FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Automated Fire Safety Sensor Distribution

Tiago Candeias Verdade

Mestrado em Engenharia Informática e Computação

Supervisor: Daniel Augusto Gama de Castro Silva

Second Supervisor: Tiago Rodrigues Vieira de Carvalho

July 31, 2022

Automated Fire Safety Sensor Distribution

Tiago Candeias Verdade

Mestrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Prof. Rosaldo José Fernandes Rossetti
External Examiner: Prof. Pedro Henriques Abreu
Supervisor: Prof. Daniel Augusto Gama de Castro Silva

July 31, 2022

Abstract

Nowadays, using technology to automate repetitive tasks previously made by humans is more relevant than ever. On that premise, there is still work that has not been made easier by technology, as is the case of fire detection planning, which we will explore in full detail.

In this thesis, we briefly explore the work of fire designers and identify some of its manual processes that could be automated. We also present the *Safe Cities - Fire Planning* project, which this work is part of and aims to automate those processes, and analyze the interaction between our program and the rest of its larger environment.

The goal of this project was to make the work of the fire designers easier. To accomplish this, we explored some state-of-the-art Wireless Sensor Networks (WSN) coverage algorithms, which is an equivalent problem to the distribution of fire sensors. With that knowledge, we focused on making a program that receives a 2D environment with obstacles and can run WSN distribution algorithms. After implementing the previous works' algorithms, we focused on creating modifications to improve those solutions. Our program's final objective was also to be able to create new algorithms as easily as possible, so we also focused on the modularity of our program for future works to be done on top.

In this work, we also evaluate our solutions, comparing them to the state-of-the-art solutions using metrics like the number of sensors used and coverage area. Our results show improvements in our modified algorithms compared to the original ones, namely improving from a coverage percentage of 68.2% to 96.5%, and from 86.3% and 79.7% to 100%. From our results and work, we believe this thesis contributes well to the WSN distribution topic and the fire equipment distribution subject.

Keywords: Distribution, Automation, Fire Safety

"Time you enjoy wasting
is not wasted time."

Marthe Troly-Curtin

Contents

1 Introduction **1**
- 1.1 Context and Motivation . . . 1
- 1.2 Objectives . . . 2
- 1.3 Methodology and Main Contributions . . . 2
- 1.4 Document Structure . . . 3

2 Related Work **5**
- 2.1 WSN Coverage algorithms . . . 5
  - 2.1.1 Virtual Force Solutions . . . 5
  - 2.1.2 Computational Geometry Solutions . . . 6
  - 2.1.3 Geometrical Patterns Solutions . . . 8
  - 2.1.4 Optimization Solutions . . . 9
- 2.2 Environment Representation . . . 11
- 2.3 WSN simulation software . . . 12
- 2.4 Conclusion . . . 13

3 Problem Description and Proposal **15**
- 3.1 Fire Designers contribution . . . 15
- 3.2 Safe Cities - Fire Planning ecosystem . . . 16
- 3.3 Ecosystem Architecture . . . 16
- 3.4 Solution Architecture . . . 18

4 Implementation **19**
- 4.1 Wireless Sensor Network Environment Framework . . . 19
  - 4.1.1 Processor Component . . . 19
  - 4.1.2 Scenarios Component . . . 20
  - 4.1.3 Metrics Component . . . 22
  - 4.1.4 Algorithms Component . . . 22
- 4.2 Scenarios . . . 24
- 4.3 Metrics . . . 26
- 4.4 Algorithms . . . 29
  - 4.4.1 Geometrical Patterns Algorithms . . . 30
  - 4.4.2 Virtual Force Algorithms . . . 32
  - 4.4.3 Computational Geometry Algorithms . . . 35

5 Results **43**
- 5.1 Miscellaneous Algorithms . . . 43
- 5.2 Geometrical Patterns Algorithms . . . 43

5.3 Virtual Force Algorithms 44
5.4 Computational Geometry Algorithms 45

6 Conclusions and Future Work **53**
6.1 Conclusions 53
6.2 Future Work 55
6.2.1 Safe Cities - Fire Planning 55
6.2.2 WSN simulators 56

A Scenarios **59**
A.1 Empty Group 59
A.2 One Obstacle Group 61
A.3 Multiple Obstacles Group 63
A.4 Multiple Rooms Group 65
A.5 Tight Corridors Group 66

Bibliography **69**

List of Figures

2.1 Summary of full coverage approaches . . . 6
2.2 Begin and end of OAVFA solution . . . 7
2.3 Voronoi regions and solution . . . 7
2.4 First and second step of Delaunay triangulation solution . . . 8
2.5 Possible robot move steps and final deployment solution . . . 9
2.6 Update of a population element in PSO solution . . . 10

3.1 Solution architecture . . . 17
3.2 General program structure . . . 18

4.1 Initial GUI . . . 20
4.2 Scenarios example folder structure . . . 21
4.3 Text and graphical representation of a scenario . . . 21
4.4 Sensor distribution in one obstacle scenarios . . . 25
4.5 Sensor distribution in multiple obstacle scenarios . . . 25
4.6 Sensor distribution in multiple rooms scenarios . . . 26
4.7 Sensor distribution in tight corridors scenarios . . . 27
4.8 Random algorithm in two scenarios . . . 30
4.9 Shorter movements made in the ORRD solution . . . 31
4.10 ORRD algorithm in two scenarios . . . 31
4.11 ORRD Modified algorithm in two scenarios . . . 32
4.12 OAVFA algorithm in two scenarios . . . 33
4.13 OAVFA Targeted algorithm in two scenarios . . . 34
4.14 Distribution of the OAVFA T and OAVFA CT algorithms in the decreasing width scenario . . . 35
4.15 OAVFA ORRD Targeted algorithm in two scenarios . . . 35
4.16 Voronoi algorithm in two scenarios . . . 36
4.17 Voronoi Random algorithm in two scenarios . . . 37
4.18 Voronoi Targeted algorithm in two scenarios . . . 37
4.19 Voronoi Targeted Unlimited algorithm in two scenarios . . . 38
4.20 Voronoi Mix algorithm in two scenarios . . . 39
4.21 Voronoi ORRD Targeted algorithm in two scenarios . . . 39
4.22 Voronoi ORRD Targeted Unlimited algorithm in two scenarios . . . 40
4.23 Voronoi ORRD Targeted Delete algorithm in two scenarios . . . 40
4.24 Voronoi Random OAVFA algorithm in two scenarios . . . 41

6.1 Decision tree for best algorithm to use giving conditions . . . 54
6.2 Comparison of multiple CVRP solutions . . . 57

A.1 150x500 scenario 59
A.2 500x150 scenario 59
A.3 300x300 scenario 60
A.4 900x900 scenario 60
A.5 1200x800 scenario 60
A.6 C scenario 61
A.7 Circle scenario 61
A.8 H scenario 61
A.9 Irregular scenario 61
A.10 PI scenario 62
A.11 T scenario 62
A.12 2 Irregular 1 scenario 63
A.13 2 Irregular 2 scenario 63
A.14 5 Circles scenario 63
A.15 5 Squares scenario 63
A.16 20 Squares scenario 64
A.17 3 Regular Rooms scenario 65
A.18 6 Rooms 1 Hall scenario 65
A.19 House Blueprint scenario 65
A.20 Irregular Rooms scenario 65
A.21 Width 150 scenario 66
A.22 Width 100 scenario 66
A.23 Width 50 scenario 66
A.24 Width 25 scenario 66
A.25 ZigZag scenario 67
A.26 Decreasing Width scenario 67

List of Tables

2.1 Environment representation table 11

5.1 All algorithms statistics with sensor radius 100 46
5.2 All algorithms statistics with sensor radius 300 48
5.3 All algorithms statistics with sensor radius 600 50

Listings

4.1 Number of sensors used metric class 23
4.2 Example WSN distribution algorithm class 24
4.3 Number of sensors used metric output 27
4.4 Percentage of area covered by at least one sensor metric output 28
4.5 Percentage of area covered by multiple number of sensors metric output 28
4.6 Sensors efficiency metric output 29
4.7 Processing time in single and group settings 29
4.8 Coverage when a sensor is loss metric output 29

Abbreviations

BIM	Building Information Modeling
CAD	Computer-Aided Design
CED	Coverage, Exploration and Deployment
CFPA	Chaotic Flower Pollination Algorithm
CLBGA	Candidate Location Based Greedy Algorithm
CVRP	Capacitated Vehicle Routing Problem
DWG	Drawing
EVFA	Exponential Virtual Force Algorithm
GA	Genetic Algorithm
GUI	Graphical User Interface
ICS	Improved Cuckoo Search
IDE	Integrated Development Environment
IVFA	Improved Virtual Force Algorithm
J-Sim	JavaSim
LRV	Least Recently Visited
MAC	Media Access Control
NS-2	Network Simulator 2
NSVA	Node Spreading Voronoi Algorithm
NSVGA	Node Spreading Voronoi Genetic Algorithm
OAVFA	Obstacle Avoidance Virtual Force Algorithm
OMNeT++	Objective Modular Network Testbed in C++
ORRD	Obstacle-Resistant Robot Deployment
PDIS	*Preparação da Dissertação* (Dissertation Planning)
PSO	Particle Swarm Optimization
SMCGA	Stable Marriage Crossover Genetic Algorithm
VFA	Virtual Force Algorithm
VRP	Vehicle Routing Problem
WSN	Wireless Sensor Networks

Chapter 1

Introduction

In the past years, with the advancement of technologies, there has been a big focus on automating tasks previously done manually [Acemoglu and Restrepo, 2018]. By automating some processes, sometimes it is possible to achieve a better result with fewer errors, as well as allow people to make better use of their time [Haight and Kecojevic, 2005].

1.1 Context and Motivation

Automation is lacking in many areas, and particularly one of them is in the fire designers' work [Mohammad, 2016]. Fire designers are responsible for planning the fire systems when plans for a building are being made. Fire systems can contain multiple types of equipment in order to be able to detect fires as early as possible and consistently [Spatar et al., 2017]. The equipment used can consist of traditional fire detectors, such as automatic fire detectors and notification appliances, special detectors, like linear heat detectors and aspirating smoke systems, and fire alarm panels to connect all gears. During the work of a fire designer, they also have to consider multiple rules and legislations that are in vigor in a building's respective country and zone relative to the distribution and arrangement of the different types of fire equipment [Beeby and Narayanan, 2005]. A fire designer's work's main file types are the *Computer-aided Design* CAD or *Building Information Modeling* BIM files, which are solutions to creating and managing 3D information. With these file types, the designers can get the main structure of the building they are designing for and add multiple pieces of equipment to the building plans. However, to the best of our knowledge, this is still a manual job, where the worker creates and assigns multiple types of equipment to the building and connects them one by one. Since this manual task takes much time to have all the equipment set up, there exists an interest in its automation, especially because of the possible increase in productivity.

1.2 Objectives

All project goals focus on making the work of the fire designers easier, specifically by making some of their work processes automated.

The project's main goal consists of creating a program that can distribute different types of equipment through a building plan always under the defined legislation rules. This program would input the 3D building plans and the specified rules and return as output the 3D building plans with all fire equipment placed and connected optimally.

In order to achieve this, we will be creating a program that is a component of a bigger project, Safe-Cities, that has that objective, as seen in section 3. Our component will be responsible for distributing the sensors in an environment.

Since our fire sensor distribution problem is equivalent to the *Wireless Sensor Network* WSN coverage problem, we will focus on multiple already implemented solutions to compare their results. After, we will focus on the question: Is it possible to implement modifications to the previous algorithms which improve their performances?

We will also focus on making the program modular so that it is easier for future work to be done on top of ours. So in terms of modularity, we will emphasize the question: Is it possible to create a WSN distribution program in which new algorithms, metrics, and scenarios can be added to run and see their performances?

1.3 Methodology and Main Contributions

To evaluate the quality of the work developed during the project, we had to focus on specific metrics related to multiple domains of the work done.

Relative to the efficient placement of the equipment through the environment, we focused primarily on the percentage of coverage of the area of interest, meaning the ratio between the area covered by the fire detectors and the total area of interest. We also considered the number of equipment necessary to have a good coverage percentage, being a better result a lower number of the equipment used for a given percentage. Also, a critical aspect of the distribution solutions is the time it takes to achieve them, which was also our focus.

In order to have and implement a good sensor distribution algorithm, we first focused on researching previous solutions already implemented. After considering their results, we determined which algorithms best suited our needs. Since the algorithms did not compare in the same environment and with the same test cases, we decided to implement a framework on which it would be possible to implement and run distribution algorithms. Since we wanted to test in the same scenarios and with the same metrics, we implemented both types in the framework and made it easy to add new ones. With the framework complete, we implemented the most relevant algorithms from our state-of-the-art to test them. Comparing the algorithms' results in the same test cases made comparing them more accessible and accurate. Afterward, we focused on improving the results of the previous works' algorithms by creating modified versions in the framework. With

the modifications, it was possible to compare all the results and see if there is, in fact, a metric performance improvement.

From all our work developed during this project, we believe that it has some good contributions. On a more technical level, it contributes with a framework of comparison of sensor distribution algorithms, which also already contains multiple implementations of some studied algorithms and some newly implemented variants. The framework also contains some implemented metrics and scenarios for comparison and analysis. On a more scientific level, this work contains the identified and implemented improved modifications of some state-of-the-art algorithms that showed the best results. Another good contribution is the developed application's potential in practical, real-life work, like fire designers.

1.4 Document Structure

In chapter 2, the state-of-the-art research is shown, where projects that try to solve problems similar to the ones we have to solve are analyzed in-depth and compared. Chapter 3 presents some context to our work and the solution architecture. Chapter 4 contains the description of the implementation work made at a high level and a low level with a description of the multiple smaller components. Chapter 5 shows the results obtained from our work and some analysis. In chapter 6, we conclude the presentation of our work by summarizing, explaining the conclusions we have reached and mentioning the future works to be done.

Chapter 2

Related Work

Since our problem consisted of distributing fire sensors in an area in order for the sensors to cover it fully, we tried to research that topic. However, from our search, we have not found previous works that focused on the distribution of fire sensors, so we had to broaden our search space. In that research, we found a problem in a different area that seemed to focus on a similar problem as ours, the maximum coverage in Wireless Sensor Networks. So to better understand the current state of the already available solutions for that problem, we researched a bit more on some aspects of wireless sensor networks.

2.1 WSN Coverage algorithms

Since the main logic of covering all the divisions with fire detection equipment is very similar to the well-known problem of maximum coverage in Wireless Sensor Networks, it is essential to explore some of its more recent and better-performing solutions.

When translating our problem to WSN coverage problems, we identified that the best co-related problem was the Full Coverage problem. The Full Coverage problem focuses on having 100% coverage of all the relevant areas while at least one node covers all points [Mohamed et al., 2017].

There are several techniques used to solve the full coverage problem, as shown in Fig. 2.1 and detailed in [Mohamed et al., 2017]. We will explore some of those solutions' categories in the following subsections.

2.1.1 Virtual Force Solutions

Virtual Force solutions are based on the behavior of electromagnetic particles. The sensor nodes create a repelling or attractive force between each other or with objects depending on their distance [Mohamed et al., 2017].

The Obstacle Avoidance Virtual Force Algorithm (OAVFA) is a proposed solution for the coverage problem in [Rout and Roy, 2016]. Initially, the nodes are randomly deployed on the relevant area, and then during multiple iterations, numerous forces interact with them. The force

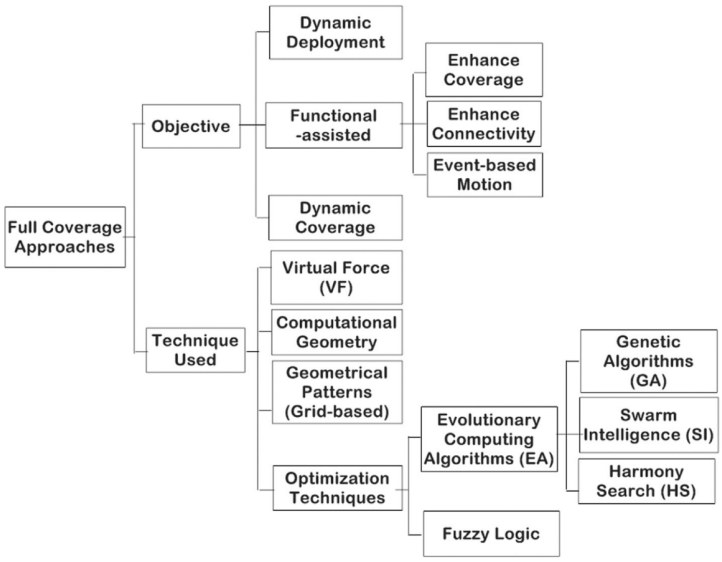


Figure 2.1: Summary of full coverage approaches (extracted from [Mohamed et al., 2017])

between two nodes is majorly affected by their distance: if they are below a threshold (in the paper $\sqrt{3}$ of nodes' radius), they are repulsive; otherwise, if above, they are attractive. The nodes also interact with objects, creating repulsive forces between them below a defined threshold. The final solution is achieved when all the sensors come to an equilibrium. In Fig. 2.2 an initial and final state of the proposed solution is shown.

The results achieved by OAVFA were promising. Their tests achieved an average 96.2% coverage rate when using 40 nodes, a bump compared to other solutions of the same type like *Improved VFA* (IVFA) and *Exponential VFA* (EVFA) [Chen et al., 2007], which achieved 91.8% and 90.7% coverage, respectively.

Virtual Force solutions seem to have good results, especially when the coverage area is ample and its obstacles are convex. The worst results appeared when the obstacles were very concave, possibly by some constrain on the movement of the nodes because of the obstacles walls. One limitation to notice is the possibility of using much time to achieve the final solution. Since the algorithm only stops when the nodes reach a stable position, in some cases, it can take many iterations to achieve it and take more time than expected.

2.1.2 Computational Geometry Solutions

Computation Geometry solutions mainly use Voronoi diagrams (Fig. 2.3a) or Delaunay triangulations to solve the problem.

A Voronoi diagram solution is found on [Eledlebi et al., 2020] to achieve maximum coverage in WSNs. It consists of iteratively adding nodes to the bottom-left corner if unexplored space exists in the region of interest. During each iteration, the current nodes move to the center of gravity of their sensed Voronoi region, which is calculated by using the intersection of their perpendicular bisectors and the node's sensing range. The end condition is being checked for each iteration,

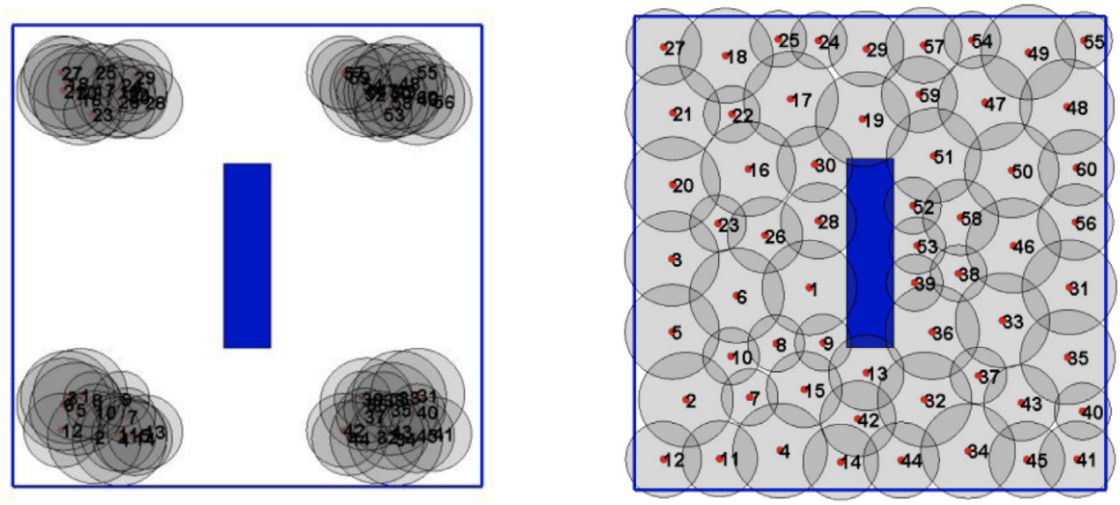
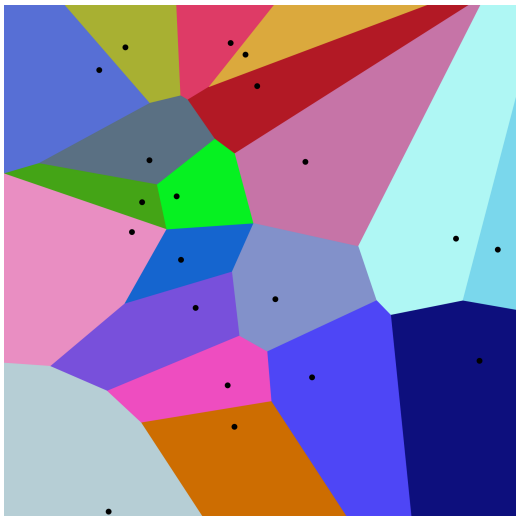


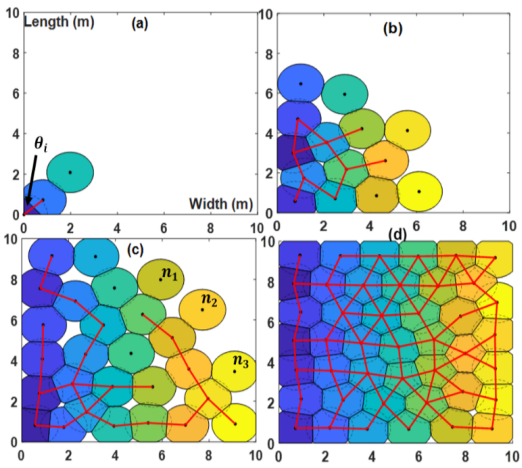
Figure 2.2: Begin and end of OAVFA solution (extracted from [Rout and Roy, 2016])

checking if the current nodes have not shifted more than a certain threshold. Fig. 2.3b shows multiple steps of the proposed solution.

This solution made it possible to achieve 99.3% area coverage with 16 nodes in 20 seconds for an empty room with a 100 meters side. For this same room, the *Node Spreading Voronoi Algorithm* (NSVA) [Zou et al., 2012] algorithm achieved 93.4% of coverage with 40 nodes in 150 seconds, and the *Node Spreading Voronoi Genetic Algorithm* (NSVGA) [Zou et al., 2012] attained 98.04% coverage with 40 nodes in 150 seconds. These results make it possible to conclude that the solution is a top performer in empty spaces. The Voronoi solution reached around 92% coverage in obstacle-rich rooms, and in environments with crevices (narrow margins), it got around 70% coverage.



(a) Voronoi regions



(b) Voronoi solution at multiple stages (extracted from [Eledlebi et al., 2020])

Figure 2.3: Voronoi regions and solution

A Delaunay triangulations solution can also be found on [Dai and Wang, 2015] to obtain complete coverage in an area with obstacles. Their solution is an iterative process that, at the start with some initial seed nodes, tries to find the Delaunay triangle with the largest coverage hole to place a node. To find this triangle, it searches all valid triangles for the one with the largest circumcircle. This process repeats until all the triangles are covered. Fig. 2.4 shows the finding of the triangle with the largest coverage hole in two steps of this Delaunay solution.

This triangulation solution achieved coverage of 100% on the six tests run, which all consisted of a room with one or two irregular obstacles inside. Making it a better performer than the *candidate location based greedy algorithm* (CLBGA) [Zhu and Wang, 2014] tested against, that only got around 98%. Also, one key advantage of this algorithm is the time to reach its final solution, which was around 4 seconds—much faster than CLBGA, which took around 600 seconds.

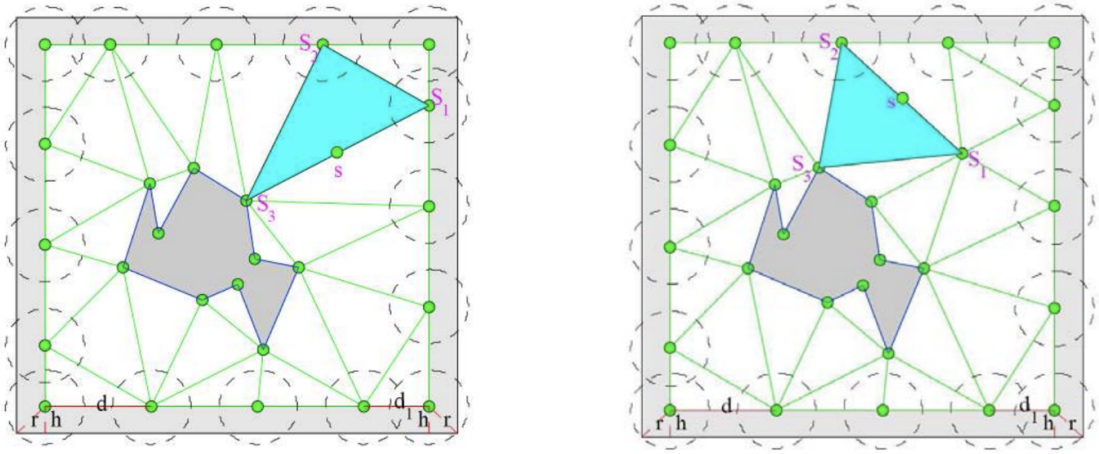


Figure 2.4: First and second step of Delaunay triangulation solution (extracted from [Dai and Wang, 2015])

Computational Geometry Solutions achieve great results while using a low amount of time to reach their final solution. Even when tested against rooms with irregular objects and obstacle-rich rooms, they presented solutions with high coverage. One possible disadvantage of using these methods is that the number of sensors used in their final solutions might be bigger than other solutions, like Geometrical Patterns Solutions, in some cases.

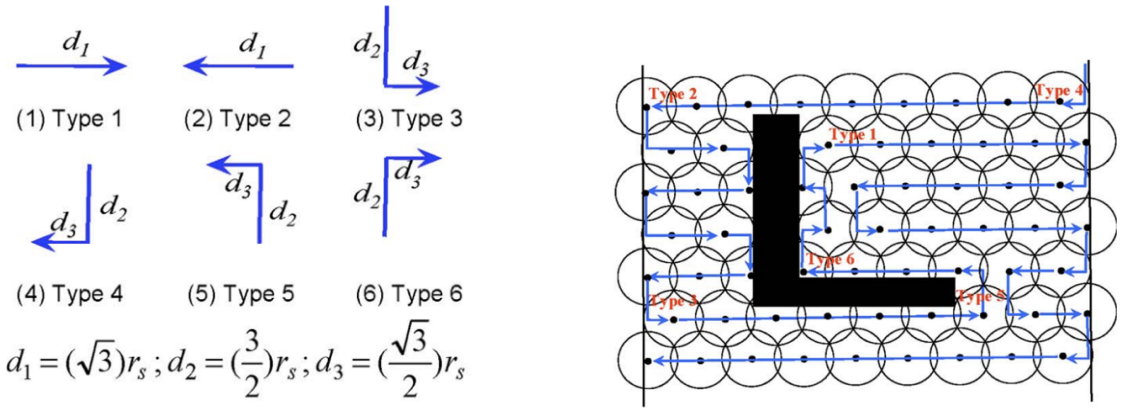
2.1.3 Geometrical Patterns Solutions

Geometrical Patterns solutions base themselves on inherently good patterns, like triangles and hexagons, for the regular deployment of the nodes in the region of interest.

In [Chang et al., 2009], a solution is presented, consisting of an algorithm that distributes nodes in a hexagon grid on the area of interest which can contain obstacles. The algorithm is planned to be run on a robot that deploys the nodes, consisting of simple ordered possible steps as can be seen in Fig. 2.5a. However, with obstacles, some additional rules had to be made to ensure complete coverage in some edge cases when the planned nodes land on obstacles or outside the

area of interest. These rules assure full coverage by adding nodes to the edges of the obstacles obstructing the predicted placement. A final solution is presented in Fig. 2.5b.

This robot solution on the six tests made, each consisting of a room with one regular obstacle, scored 100% coverage. Its scores were better than the *Coverage, Exploration and Deployment* (CED) [Batalin and Sukhatme, 2004] and *Least Recently Visited* (LRV) [Batalin and Sukhatme, 2007] algorithms tested against, which both had coverage of around 90%. One good remark was also that this solution achieved these results with fewer nodes, around 165, while the CED used around 220 and LRV around 190. One limitation identified is its use with irregular objects, which the algorithm might not work as intended. This could happen because, with irregular objects, the algorithm might not detect hidden areas that are not in the path of the possible moves, leading to uncovered areas.



(a) Possible robot move steps

(b) Final deployment solution

Figure 2.5: Possible robot move steps and final deployment solution (extracted from [Chang et al., 2009])

Geometrical Patterns solutions are able to achieve good results using a low quantity of sensors for their solutions. However, when there exists the presence of irregular objects in the area of interest, these algorithms do not have such good results, making them not achieve full coverage.

2.1.4 Optimization Solutions

Optimization solutions can use Genetic Algorithms or Swarm Intelligence techniques to improve a setup to an optimized solution.

A Genetic Algorithm (GA) solution can be found on [Yoon and Kim, 2013]. The chosen representation for each possible solution is an array with 2D coordinates of the proposed WSN nodes' positions. Initially, N/2 randomly created pairs create N/2 offspring, originating from crossover and mutation. In this solution, the crossover is generated by randomly choosing values for the offspring from an interval that considers the maximum and minimum values of the same position in both parents. Moreover, the mutation occurs to genes, which are modified by a random Gaussian number, modifying a node's coordinate position. All the individuals (parents and offspring)

are ranked based on the node's coverage area and chosen for the next generation of size N. The process of creating a new population from the best individuals of generated offspring and current population repeats itself until a fixed number of generations is reached.

In this genetic solution, the tests were on a 100mx100m space with a different subset of nodes with different coverage radii. The results presented were better in all test cases than the Multi-start VFA [Yoon and Kim, 2013] and *Stable Marriage Crossover Genetic Algorithm* (SMCGA) [Seo et al., 2008] tested against. However, since the test cases do not guarantee that it is possible to have 100% coverage, their results are a bit inconclusive aside from comparison with the other two solutions.

A Particle Swarm Optimization (PSO) solution, which is inspired by the habits of a flock of animals, can also be found on [Huynh et al., 2019]. The main difference between PSO and GA is that the individuals update themselves using the one-way information from the particles with the best results in the PSO solution. In this PSO proposal, the representation for each possible solution is an array with 2D coordinates of the proposed WSN nodes' positions. The initial population is iteratively updated in this solution by considering the current best solution for a sub-group and the current general best solution. These rankings of solutions are determined considering the area of interest covered by the WSN nodes. The process of updating an element is shown in Fig. 2.6.

In this solution, the test cases consisted of a 100mx100m space with various objects and three types of nodes with different radii. This solution achieves around 90% coverage from the overall results, better than the *Improved Cuckoo Search* (ICS) and *Chaotic Flower Pollination Algorithm* (CFPA) [Binh et al., 2016] counterparts, which reach around 80% and 85% coverage. One limitation of the proposed solution is that all the objects in the test cases are regular.

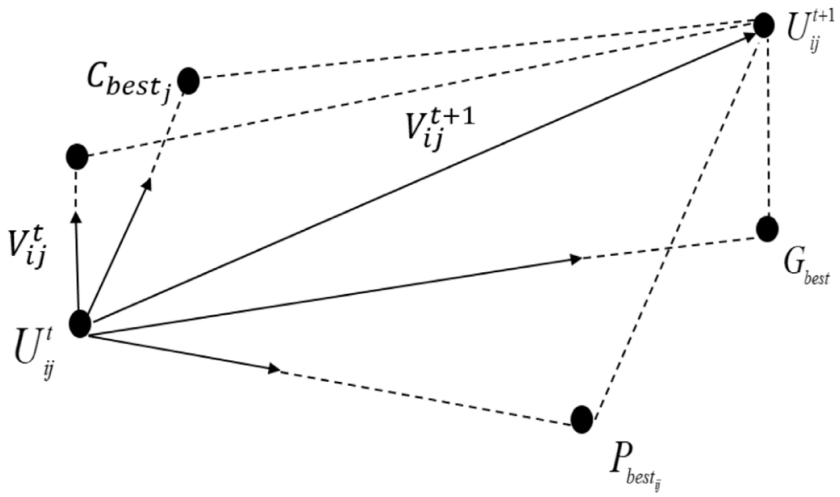


Figure 2.6: Update of a population element (extracted from [Huynh et al., 2019])

Optimization Solutions can reach relatively good results of around 90% coverage. However, these results are primarily obtained in test cases with few objects or only regular obstacles. This

type of algorithm can be inefficient to solve a problem more complex with irregular objects, making it a possible limitation.

2.2 Environment Representation

One important aspect to note about all the previous works done is their representation of the environment. All the presented scenarios contain obligatory components, such as space dimensions and sensors, and some optative components like obstacles. Even though there is not so much variety in represented components, there is an essential distinction on how those components are represented in code format. From all the papers researched, two different ways of that representation were noticeable, which will be further explored next.

The first representation is probably the one more rudimentary and less precise. It consists of representing the environment as a 2D array, where each element represents a position in the 2D space. Each 2D array element can contain an identification number that represents if in that position there exists an obstacle or a sensor. This solution can represent the real space to a certain predefined precision since the number of rows and columns on the 2D array must be defined priorly. The advantages of this solution consist of the easiness of implementation and performance when using lower precision.

The other representation is a bit more robust and complex. It consists of representing the components with decimal values. Each component can be defined as points, as is the case for obstacles, or with circles, as is the case for the sensors. In the case of obstacles, they can be represented as a list of points with x and y coordinates defining their bounds. For the sensors, they can be defined as a point with x and y coordinates as their center and a radius dimension. The advantage of this solution is the increase in precision since decimal values are used. Another important remark is that converting this format to the 2D array one is relatively easy.

In Table 2.1 it can be seen which representation was implemented on some of the papers researched.

Table 2.1: Environment representation table

Paper	**Approach**	**Representation**
[Dahmane et al., 2020]	Optimization	2D Array
[Rout and Roy, 2016]	Virtual Force	Decimal
[Eledlebi et al., 2020]	Computational Geometry	Decimal
[Dai and Wang, 2015]	Computational Geometry	Decimal
[Chang et al., 2009]	Geometrical Patterns	Decimal
[Yoon and Kim, 2013]	Optimization	Decimal
[Huynh et al., 2019]	Optimization	Decimal

Even though almost all algorithms have similar code representations for the Environments, it is possible to see from the papers that all the implementations of the environment, its logic, the

sensors and the sensors distribution are all ad-hoc solutions that the writer of the paper created. This makes it more difficult to create a project related to this subject, because before being able to create and test some sensor distribution algorithm it is necessary to create the whole infrastructure behind all the environment and sensors interaction. This is a big point that our work tries to touch on, trying to create a program that can be used as a start point for someone to develop a sensor distribution algorithm. Having an equal base point as other implementations makes it easier to test and compare solutions with each other.

2.3 WSN simulation software

Another important part of WSNs and their initial setup is testing to see the performance of their nodes and how they behave in a real-world scenario. Since it is costly to make these tests in the real scenario, WSN simulators try to mimic the real-world scenario with all its properties to test it before advancing to the real implementation [Nayyar and Singh, 2015].

In this section, we will briefly review some already implemented simulators. Our search consisted of finding the most used and conceived simulators that can be used in the area of Wireless Sensor Networks. Our research was done primarily by searching for papers that reviewed this type of simulator and then further searching individually for each one.

The *Network Simulator Version 2* NS-2 [1] focuses more on the communication aspect of the network. It provides support for multiple protocols like TCP [2], HTTP [3], and UDP [4]. From this simulator, it is possible to simulate network scenarios and analyze its outcomes based on various network factors like packet loss, delay, error, and intrusion [Downard, 2004]. However, it is impossible to run a distribution algorithm and see its performance in terms of coverage area and sensor efficiency.

The *Network Simulator Version 3* NS-3 [5], like NS-2, is also regarded as a discrete-event simulator but is not considered an extension of NS-2 since it is an entirely new simulator [Campanile et al., 2020]. The objective behind its development was to enhance research in communication networks [Nayyar and Singh, 2015]. Like the NS-2, its focus is on the simulation of network scenarios, making it impossible to simulate a WSN distribution and see its coverage and efficiency performance.

The OMNet++ [6] is an object-oriented discrete-event network simulator for WSNs. The motivation for developing OMNeT++ was to produce a powerful open-source discrete event simulation tool for the simulation of computer networks and distributed or parallel systems [Varga and Hornig,

[1] NS-2 Wiki available on http://nsnam.sourceforge.net/wiki

[2] Transmission Control Protocol (TCP) is one of the main protocols of the Internet protocol suite. TCP is connection-oriented, and a connection between client and server is established before data can be sent

[3] The Hypertext Transfer Protocol (HTTP) is an application layer protocol in the Internet protocol suite model for distributed, collaborative, hypermedia information systems.

[4] User Datagram Protocol (UDP) is one of the core members of the Internet protocol suite. With UDP, computer applications can send messages, in this case, referred to as datagrams, to other hosts on an Internet Protocol (IP) network.

[5] NS-3 Official website available on https://www.nsnam.org/

[6] OMNeT++ official website: https://omnetpp.org/

2008]. OMNeT can also provide a framework and tools for writing simulation [Priyadarshi et al., 2020]. OMNet++ focuses on the sensors' network communication aspect and cannot simulate a distribution of WSN sensors and calculate the area coverage or efficiency of the placement of the sensors.

The Castalia Simulator [7] is an open-source simulator for wireless sensor networks, and body area networks, which is widely used in the academic and research community [Pediaditakis et al., 2010]. This simulator is optimized for testing distributed algorithms and protocols and features an accurate channel/radio model, radio behavior, and other aspects of communication [Engmann et al., 2018]. Even though Castalia focuses on WSNs, its functionalities do not permit testing an algorithm that tries to distribute sensors in an environment and calculate its performance metrics like coverage %.

J-Sim [8] is considered a general-purpose simulator, which also is platform-neutral, extensible, and reusable [Sobeih et al., 2005]. It is built in Java and has a Script Interface, making it easier to run script-based languages like Perl or Python. J-Sim has been used to implement three protocols of WSNs: Localization, Geographic Routing, and Directed Diffusion. It also is a good simulator when using a large number of sensors like 1000 [Nayyar and Singh, 2015]. Even though J-Sim can use localization features and calculate some metrics like coverage, the simulator cannot run a sensors distribution algorithm to determine which sensors to use and where.

From our overall research to the best of our acknowledgment, we have not identified a WSN simulator that was able to run sensor distribution algorithms with the finality of having a sensor set up in an environment and display relevant metrics like coverage % or sensors efficiency from that distribution. These results are not that surprising since, from the research of WSN distribution algorithms in all the papers observed, all had created their ad-hoc solutions to have their environments and algorithms running. With this, it is possible to conclude that a framework that focuses on the sensor distribution aspect could be beneficial to exist.

2.4 Conclusion

From the state-of-the-art, it was possible to see some solutions relative to the distribution of sensors that show good results in terms of coverage and the number of sensors used for that coverage. However, it was also possible to see that some solutions performed poorly compared to other types of solutions. With the results from the solutions, it was possible to determine which distribution algorithms were more relevant to try to test in our work.

With the analysis of the simulators of WSNs, we also identified that there does not seem to exist a platform that focuses on the placement and distribution of the sensors as much as we wanted. The lack of this kind of platform made it clear that a platform for testing the placement and benchmarking the various algorithms in distribution was missing, which led us to focus a bit more on this topic and create such a program.

[7] Castalia Official repository available on https://github.com/boulis/Castalia
[8] J-Sim website: https://sites.google.com/site/jsimofficial/

Chapter 3

Problem Description and Proposal

In this chapter, we will explore more about the fire designers' work and how it contributes to the grand scheme of creating a building. We will also dive deep into the context into which our project is inserted, namely the Safe Cities ecosystem. After, we will present the architecture of our solution.

3.1 Fire Designers contribution

Fire designers' contribution to planning a building consists of designing and planning the necessary fire safety systems. Usually, the main portion of their work is focused on after the architect makes the building's basic plans, which contain the floor plans. For their work, they use tools that can manipulate CAD or BIM files containing the building structure to add all the necessary equipment for the fire systems.

The fire systems contain a portion of equipment focused on fire detection, which will be our focus. The fire detection equipment is composed of fire detectors, which detect the presence of fire in the area near them, fire panels, which can control what happens in the system by sounding the alarms or calling the responsible operator, and manual call points, which can be used to fire the alarms manually. In order to communicate, all the fire detectors and manual call points are connected via wiring in "loops" to a fire panel. A loop consists of multiple fire detectors and manual call points connected linearly, which form a circle. Every loop must be connected to a single fire panel, but a fire panel can contain multiple loops.

During a fire designer's work, multiple problems must be addressed to have a good fire detection system. Firstly, the fire designer must know the specific rules and legislation related to fire safety for where the building will be constructed, which can contain restrictions like the maximum area that a fire detector can cover, and the minimum space between a fire detector and a wall, and more. Secondly, they must place all the detection equipment so that the detectors completely cover all the rooms and follow the legislation as seen in the guidelines of the *European Committee for*

Standardization [CEN, 2018]. In this step, it is also essential to be as efficient as possible, planning with as few detectors as possible while always covering all areas. Finally, the fire designers must connect the equipment in loops to communicate between them. In this step, optimizing the loops to use as little wiring as possible while keeping all the equipment connected is cost beneficial.

3.2 Safe Cities - Fire Planning ecosystem

Safe Cities is a project that focuses on building urban safety, of which fire safety planning is a part. There have been works previously developed for this project which constitute the ecosystem. These programs are essential because, in our project, it will be necessary to use them in some parts of it to do additional processing.

One of the programs from this ecosystem is the one that can detect rooms given the floor plans of the building in question. This program takes as input a *Drawing* DWG file[1] with the 3D structure of the building. Then it uses an up view to create a 2D image of the upper view of the building and returns the positions of the mapped rooms according to the pixels in the image. The solution for this processing consists of using classical computer vision methods like Canny [Ding and Goshtasby, 2001] for the edge detection and Hough Transform [Illingworth and Kittler, 1988] for the line detection on the image derived from the DWG file.

Another ecosystem program focuses on determining the constitution of the fire detection loops. Fire loops are constituted of a fire panel that connects itself to several detection equipments of multiple types, forming a circle. This program makes it possible to determine the number of the different types of detection equipment that should be used in each fire loop. With the help of Google OR-Tools[2] technology, this solution uses constraint programming to achieve its results.

With the addition of this project to the Safe Cities - Fire Panning ecosystem, it will be one step closer to having all the components integrated and communicate with each other, making it possible to achieve its intended purpose.

3.3 Ecosystem Architecture

A solution architecture can help understand how the components of our solution interact with each other in the big scheme of things. A solution diagram is presented in Fig. 3.1.

Our Fire Equipment Distribution Program is part of a more extensive ecosystem, Safe Cities, where multiple programs process information and exchange it with each other.

The process begins with the information provided by the user, which consists of one CAD/BIM file with the building structure.

[1] DWG is a compact binary format that stores and describes the content of 2D and 3D design data and metadata. Format maintained by AutoDesk https://www.autodesk.com/

[2] Google OR-Tools is an open source software suite for optimization, tuned for tackling the world's toughest problems in vehicle routing, flows, integer and linear programming, and constraint programming. From Google https://developers.google.com/optimization

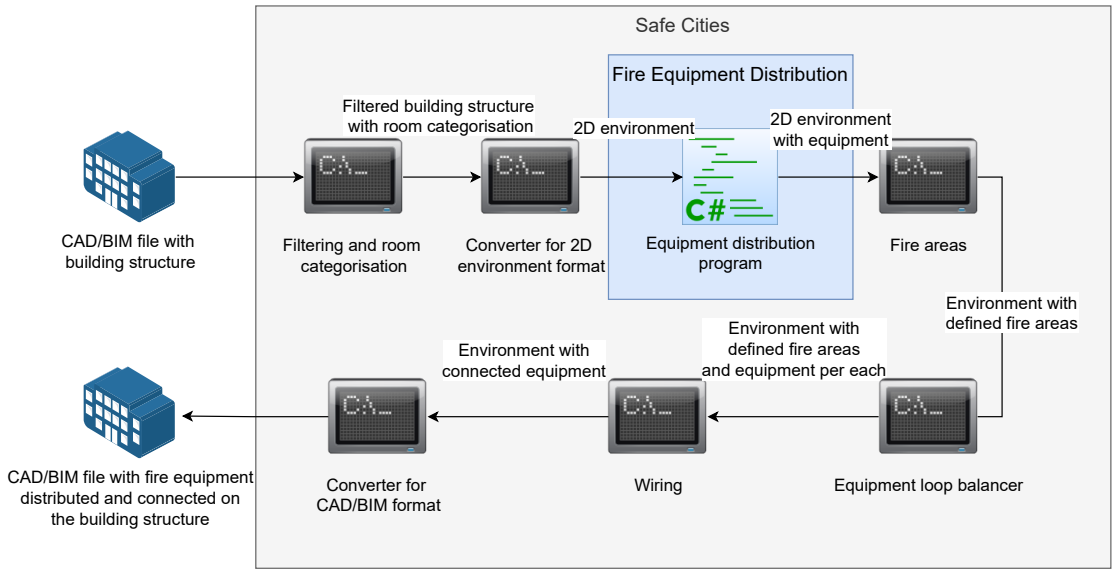


Figure 3.1: Solution architecture

The building structure information is fed to the "Filtering and room categorization program," which processes the structure, filters it to return more relevant information, and categorizes the multiple rooms present in the building.

The outcome of the categorization program is then given as input to the 2D environment converter. Here, a conversion is made to represent the environment with all its obstacles and dimensions in 2D.

Then the previous 2D environment output serves as input for the equipment distribution program, which is the focus of this thesis. In this component, a spatial equipment distribution is achieved that tries to maximize the coverage area of the fire detectors in the multiple rooms using as few pieces of equipment as possible.

After distributing the equipment in the environment, it proceeds to the fire areas component. This component defines how many fire areas will be necessary for the scene and where they will be located.

The process goes to the equipment loop balancer component with the previous information. Here, it is processed and returned the information relative to the number of each type of equipment in each loop.

Then with the information relative to the sensors, the fire areas, and the number of equipment per fire area, the wiring component can do its work. This component tries to achieve a solution of connectivity that connects all the equipment in multiple loops in each fire area while trying to minimize the wire usage required.

Finally, the information is converted back to the CAD/BIM format in the final converter. As a final result, a CAD/BIM file with the building structure and the placement of all the fire equipment and its respective wiring connections is output.

3.4 Solution Architecture

On a high level, we wanted to create a program that could distribute fire/wireless sensors in a specified environment and show statistics for its distribution results. To achieve this, our solution consists of four main components: processor, scenarios, metrics, and algorithms.

In Fig. 3.2 it is possible to see the general structure of our implemented solution.

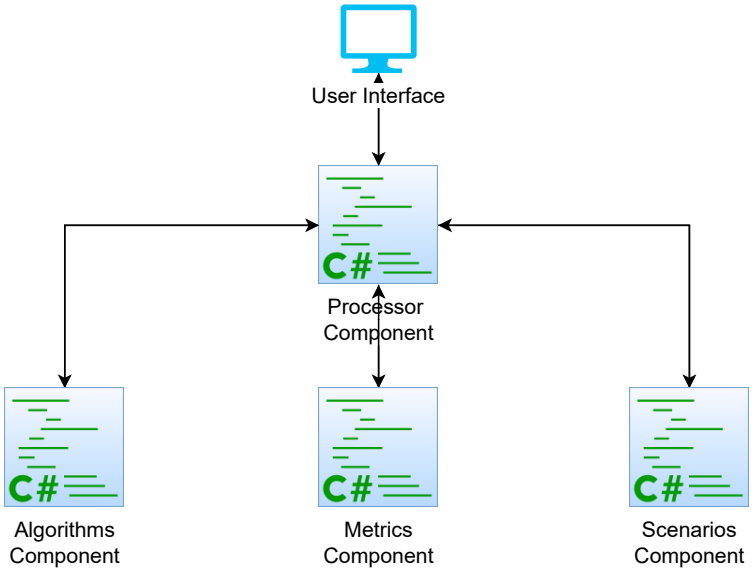


Figure 3.2: General program structure

The user interface is the main way that the user can interact with our program, and it can choose which algorithms to test with what scenarios and analyze the performance with the chosen metrics.

The processor component consists of the portion with all the base code, helper functions for the other components, and connects all the different parts of the program.

The algorithms component is where the logic of the multiple distribution algorithms is contained. Aside from the already implemented algorithms, the user can also implement new ones in our framework for testing.

The metrics component is responsible to show the performance results from the algorithms with multiple types of metrics. We provide already some metrics, but the user can add new ones to measure what is relevant to its case.

The scenarios component contains all the scenarios used to test the distribution algorithms. The user can add new scenarios to test in an environment that best correlates to its needs.

Chapter 4

Implementation

This chapter presents how our framework is structured and its functionalities and gives a detailed view of each implemented component.

4.1 Wireless Sensor Network Environment Framework

Before implementing the WSN component, we had asked authors from several related papers in the state-of-the-art section for their implementations. Since we did not get significant responses, we had to create our implementation from scratch.

In our framework implementation there exist four main components: processor, scenarios, metrics, and algorithms.

4.1.1 Processor Component

The processor component includes all the classes and code that make running and processing all scenarios, metrics and algorithms possible. This component is also the bridge to make all the other components connect among themselves.

One of the core logic implemented in this component is the representation of the obstacles and their helper functions related to them. Since two types of obstacles exist, polygons and circular, all functionalities are implemented for both types. This component is responsible for: determining if a given point is inside the obstacle or outside of it; determining if a segment between two points intersects one of the obstacles; giving a list of distances and points closer from a given point to all the segments of one polygon; giving a list of angles and vertexes positions of all the outside corners of an obstacle.

Another core aspect of this component is the geometric functions implemented that can help do some more complex operations. Almost all logic related to obstacles and the environment uses these geometric helper functions as building blocks for their operations. These utility functions are responsible for: normalizing a vector to have a unitary length; giving the Euclidean distance

between two points; giving the minimum distance between a point and a line segment; identifying if two line segments intersect, taking into account collinearity; identifying the orientation of three points which can have a clock, counter-clock, or collinear orientation; identify if a point is between two other given points.

This component also works as a bridge between the sensor distribution algorithms logic and the scenarios. To make this work, a GUI was implemented for easier control over which algorithm, scenarios, and metrics to use. In Fig. 4.1, it is possible to see the initial GUI. This user interface allows one to choose where the results are stored, which algorithm to use, which scenario to run, which metrics to calculate, and which sensor radius to use. It is also possible to run all scenarios with a specific algorithm or run all algorithms for all scenarios with a button click. In order to get all the available scenarios the program tries to find the *Scenarios* folder and, in that, searches for sub-folders that, in turn, can contain multiple scenarios. In order to get the distribution algorithms and metrics, the program tries to find classes that are available in the compiled Assembly from the namespace *SensorDistribution* and *SensorMetric* respectively. This way of detecting scenarios and algorithms makes adding new scenarios, metrics, or distribution algorithms easy without modifying existing code.

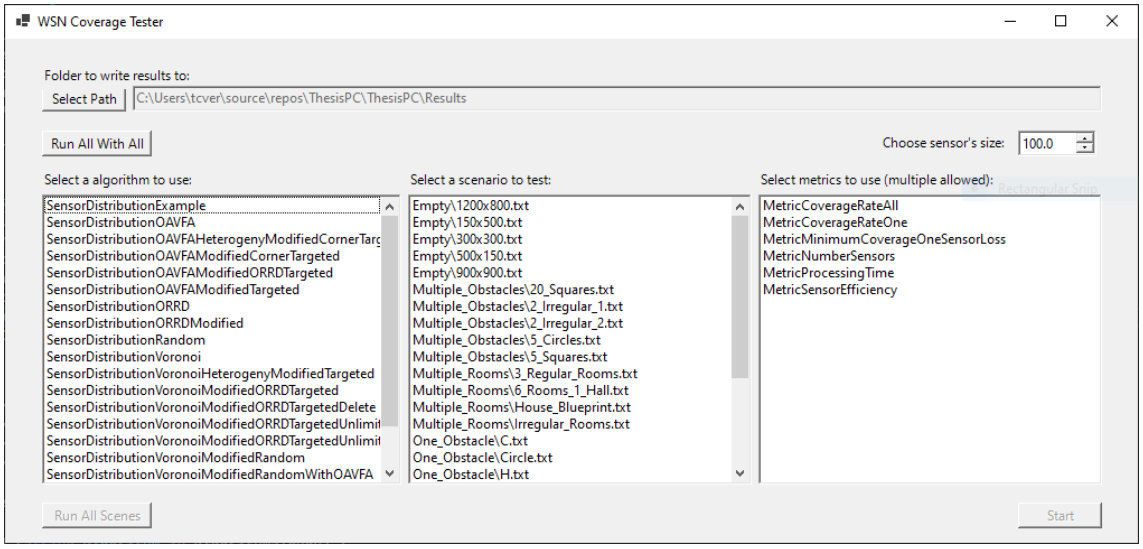


Figure 4.1: Initial GUI

4.1.2 Scenarios Component

The scenarios component is the part that contains all the scenario groups and specific scenarios. All the scenarios are located in the *Scenarios* folder at the project's root. This folder contains sub-folders that represent each group of scenarios that contain inside individual scenarios, like if it was a tree where the scene files are on the second and last level. In Fig. 4.2 it can be seen an example scenarios structure. In order to add new scenarios, either a new scenario file can be added to one of the already existing folder groups, or a new group needs to be created, and only posteriorly a new scenario file can be added to it.

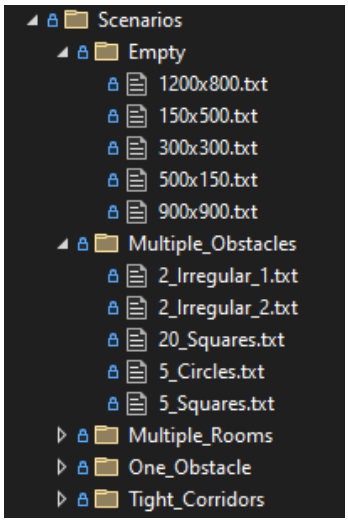


Figure 4.2: Scenarios example folder structure

All the scenarios are stored in text file format, and they can represent multiple types of spaces in the real world, like indoor/outdoor spaces, a floor of a building, or simply a room. In the scenario environment, the positions are represented by an X coordinate which indicates the horizontal position, and a Y coordinate, which indicates the vertical position, being the starting position (0,0) in the top left corner of the environment. Each scenario contains a width and a height for its dimensions, followed by the number of obstacles present and then a description of each obstacle. In the scenarios, two types of obstacles can be present, linear and circular. Linear obstacles are represented by the character *L* and a list of succeeding X and Y coordinates denoting the ordered vertices that constitute the obstacle assuming that the last point connects to the first one. Circular objects are represented by the character *C*, their X and Y center point coordinates, and the radius dimension. Fig. 4.3 shows a scenario representation in text and graphical form.

```
500.0 500.0
2
L 200.0 185.0 200.0
  350.0 300.0 350.0
  300.0 185.0 400.0
  185.0 400.0 100.0
  100.0 100.0 100.0
  185.0
C 400.0 400.0 50.0
```

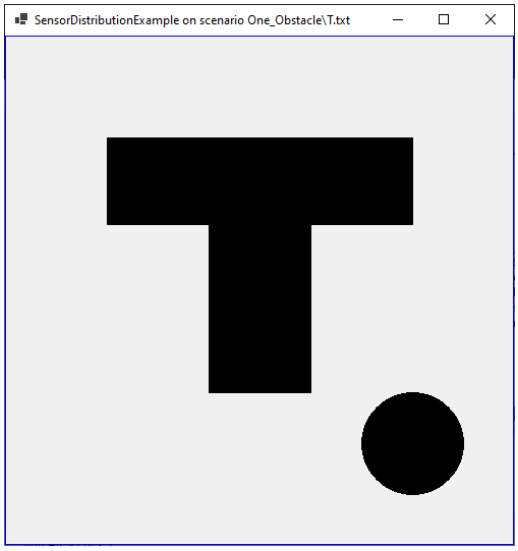


Figure 4.3: Text and graphical representation of a scenario

The main reason for this type of representation with decimal values is because it is more precise than the 2D array counterpart, as previously mentioned in Section 2.2. Also, one advantage of this representation is that it is possible to define a wider environment in a smaller space, using fractions of a unit to keep the same ratios between obstacles, sensors detection radius, and the limits of the scenario. Even though we use decimal representation for all parts of the scenario, one limitation is that when calculating the coverage area, we use a 2D grid to define if all of its points are covered by nearby sensors, which can lead to some loss in precision.

It is important to note that the implemented representation addresses the environment in a 2D way, not being able to address objects and environments in a 3D representation. This limitation makes it impossible to take into account any possible interactions between floors of a building, making the sensors on a floor unable to interact with equipment on another. However, it is still possible to address sensors' distribution in a multi-level building, but the floors need to be treated individually as a scenario each.

4.1.3 Metrics Component

The metrics component is the one that measures the performance of a specific algorithm. Each metric can provide specific information about a solution to a scene, like the number of sensors used and coverage area percentage. It also shows the aggregated results of a specific group of scenes, or in general, for the bigger picture.

We have implemented some metrics by default to be used, some inspired by previous papers on the subject like [Singh and Sharma, 2015], but the user can add more to their liking.

All the metrics implemented inherit the *Metric* abstract class, from which we can access the Coverage Processor, which is the class that processes the coverage of a scene solution, as well as the environment for further more specific information. To implement a new metric class it is necessary that the class is from the "SensorMetric" namespace and that it overrides the three functions from the abstract class *processMetric*, *printInformationOneScenario* and *printInformationAccumulated*. The function *processMetric* is supposed to process all the information and store the metric results for later. The function *printInformationOneScenario* is designed to write the metrics statistics for a specific test case, while the *printInformationAccumulated* is suppose to display the statistics of a group or general (group of groups). So to create a new metric, it is necessary to create a new class with the previous requirements in the *Metrics* folder, and it will be available in the GUI. In Listing 4.1, it is possible to see an example of a metric that displays the number of sensors used in a solution.

4.1.4 Algorithms Component

The algorithms component provides the logic to optimize the placement of the sensors to a respective scenario. With the help of the processor component, this component can define the number of sensors to use and their respective locations in the environment. These algorithms can be as simple

Listing 4.1: Number of sensors used metric class

```
namespace SensorMetric {
    public class MetricNumberSensors : Metric {

        public int numberSensors;

        public MetricNumberSensors(CoverageProcessor
           coverageProcessor) : base(coverageProcessor) { }

        public override void processMetric() {
            numberSensors = environment.getSensors().Count;
        }

        public override void
           printInformationOneScenario(StreamWriter st) {
            st.WriteLine("Sensors used: " + numberSensors);
        }

        public override void
           printInformationAccumulated(StreamWriter st,
           List<Metric> metrics) {
            double numberSensorsAccumulated = 0.0;

            foreach (MetricNumberSensors metric in metrics) {
                numberSensorsAccumulated +=
                   metric.numberSensors;
            }

            numberSensorsAccumulated /= metrics.Count;

            st.WriteLine("Mean sensors used: " +
               numberSensorsAccumulated.ToString("#.##"));
        }
    }
}
```

as randomly deploying sensors in the scenario or more complex, like using Voronoi diagrams or force-based approaches.

All the distribution algorithms implemented inherit the *SensorDistributionAlgorithm* abstract class from which can access the *env* variable that contains the environment properties. To implement a new class, it is necessary that the class is from the "SensorDistribution" namespace and that it overrides the *optimizeSensorDistribution* function, which sets the sensor nodes in the environment. So to create a new algorithm, it is necessary to create a new class with the previous requirements in the *SensorDistributionAlgorithms* folder, and it will be available in the GUI. In Listing 4.2, it is possible to see an example distribution algorithm that inserts a sensor in the

position with coordinates (100,100) of the environment.

Listing 4.2: Example WSN distribution algorithm class

```
namespace SensorDistribution {
    public class SensorDistributionExample :
       SensorDistributionAlgorithm {

        public SensorDistributionExample(Environment
           environment) : base(environment) { }

        public override void optimizeSensorDistribution() {
            env.removeSensors();

            env.addSensor(new Sensor(new Point(100, 100),
               Sensor.DEFAULT_RADIUS));
        }
    }
}
```

4.2 Scenarios

In order to test different kinds of situations during our research, we had to create multiple types of scenarios to run our algorithms. Our scenarios are grouped into five groups and contain multiple environments with different dimensions, quantities, and types of obstacles. The Empty group contains environments with no obstacles inside, with different dimensions. The One Obstacle group contains environments with only one obstacle, ranging from regular polygons to irregular and circular ones. Like the previous group, the Multiple Obstacles group contains a variety of types of obstacles, but instead of existing only one obstacle, there are multiple of them. The Multiple Rooms group contains environments where its obstacles form multiple empty spaces, which is an excellent approximation to, for example, a house structure. The Tight Corridors group contains environments where narrow obstacles form corridors. We will be exploring in more detail each group next.

The Empty group contains five scenarios, all with different dimensions without containing any obstacles. It contains environments with the following measures 150x500, 500x150, 300x300, 900x900 and 1200x800. This group can be a good way to test if a distribution algorithm can spread its sensors evenly without much overlap when not having many constraints.

The One Obstacle group contains six scenarios, all with the same dimensions but with a different obstacle each. It contains environments with obstacles in the form of a Circle, an Irregular form, and regular obstacles whose shape is similar to the letters C, H, T, and Pi. This group can be a good indicator of the performance of a distribution algorithm behavior when it encounters simple

obstacles that do not take much space from the whole environment. In Fig. 4.4 two environments can be seen in the test cases with the Circle and Irregular obstacles.

Figure 4.4: Sensor distribution in one obstacle scenarios with the Circle and Irregular obstacles

The Multiple Obstacles group contains five scenarios with the same dimensions but different forms and numbers of obstacles. It contains environments with five squares, twenty squares, five circles, two arrows, and two irregulars. This group can be a good indicator of the performance of a distribution algorithm behavior when it encounters many obstacles that can take much space from the whole environment and create many constraints. In Fig. 4.5 two scenarios can be seen in the test cases with 5 Circles and 20 Squares obstacles.

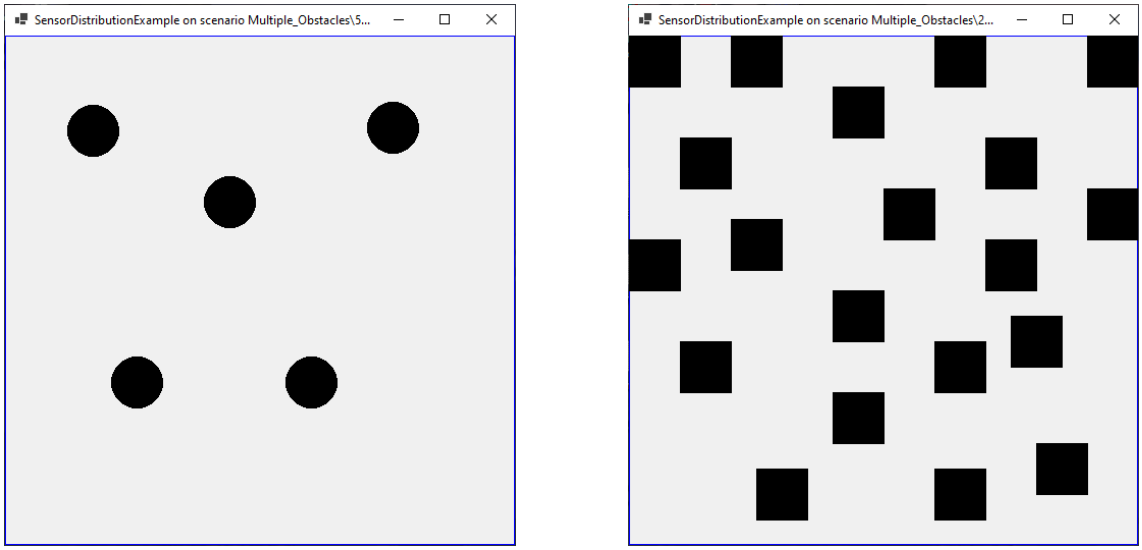


Figure 4.5: Sensor distribution in multiple obstacle scenarios with 5 Circles and 20 Squares

The Multiple Rooms group contains four scenarios, all with different dimensions and structures of rooms. It contains environments with three regular rooms, six rooms and a hall, irregular

rooms, and a house blueprint. This group can be a good indicator of the performance of a distribution algorithm behavior when it encounters a real-life building structure or a layout with empty spaces connected in some form. In Fig. 4.6 two scenes can be seen in the test cases with six rooms and a hall, and a house blueprint.

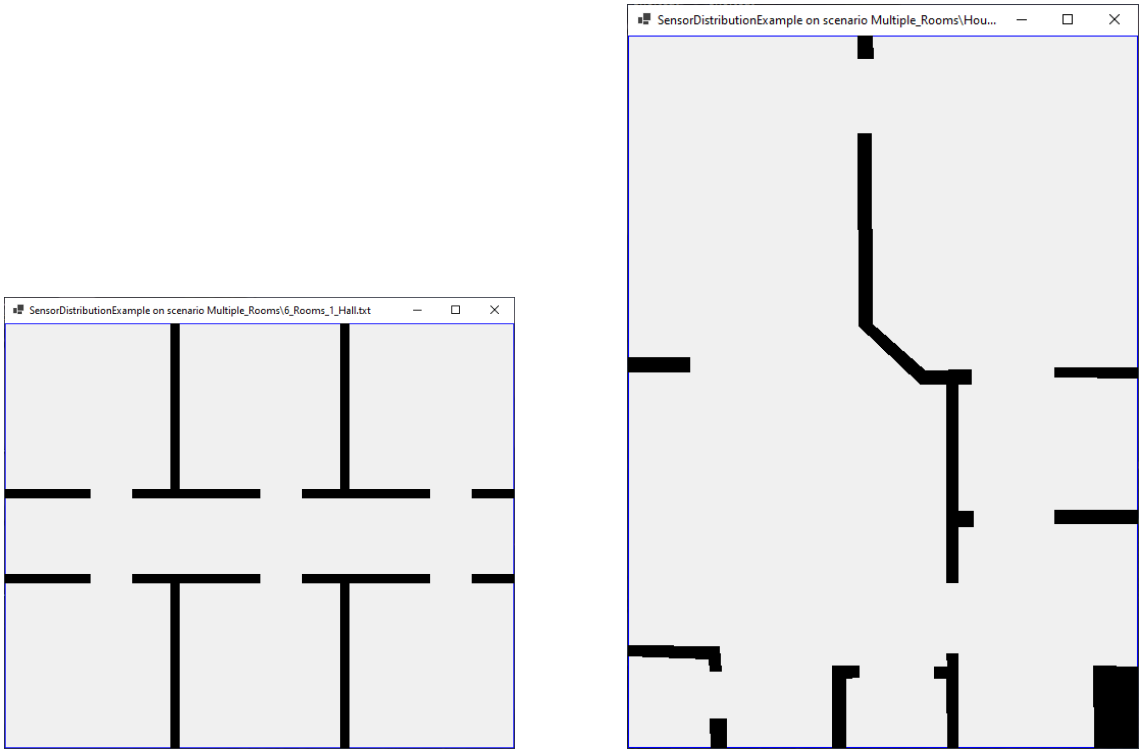


Figure 4.6: Sensor distribution in multiple rooms scenarios with 6 rooms and 1 hall, and a house blueprint

The Tight Corridors group contains six scenarios, all with different types and widths of corridors. It contains four scenarios with the same structure but widths of 150, 100, 50, and 25 for the corridors. It also contains an environment where the corridor does a zigzag shape and another where the width of the corridor decreases. This group can be a good indicator of the performance of a distribution algorithm behavior when it encounters small tight spaces that can sometimes be smaller than the radius of the sensor. In Fig. 4.7 two environments can be seen in the test cases with 25 units of corridor width and a decreasing width corridor.

4.3 Metrics

In order to better benchmark the performances of our algorithms, we had to implement relevant metrics so that we could compare results. These metrics can be used by default, some inspired by papers like [Singh and Sharma, 2015]. In this section, we will explore all the implemented metrics.

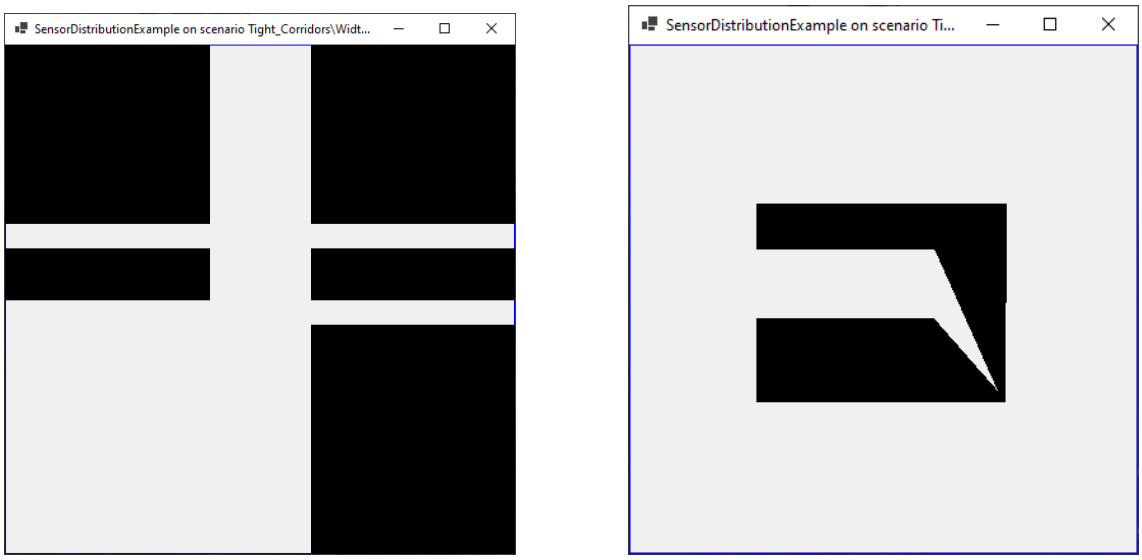


Figure 4.7: Sensor distribution in tight corridors scenarios with 25 units of corridor width and a decreasing width corridor

The first and most simple metric implemented was already shown, and it is the *MetricNumberSensors*. When used on a single environment test case, this metric displays the number of sensors placed. Furthermore, it displays the mean number of sensors used in a group of scenes when it is run. This metric is an important one to consider because it can show how expensive it would be to make this distribution. It can also be a good tiebreaker metric when the other metrics are similar, being the better one the one that uses fewer sensors.

In Listing 4.3, it is possible to see the metric results in both single and group settings.

Listing 4.3: Number of sensors used metric in single and group settings

```
Sensors used: 15

Mean sensors used: 17.19
```

Another metric implemented is the *MetricCoverageRateOne*. This metric displays the percentage of area covered by at least one sensor in a single environment setting. In a group setting, it displays the mean result from all composing test cases. This metric information is crucial because the covered area is one of the most important aspects of WSNs and our problem, in which much research is done to improve this metric results [Singh and Sharma, 2015]. In order to have full coverage of all relevant positions in the environment, at least one sensor has to cover every point, which correlates to having 100% coverage.

In Listing 4.4, it is possible to see the metric results in both single and group settings.

The next metric *MetricCoverageRateAll* is very similar to the previous one but displays a bit more information about the coverage. It displays the coverage percentage of multiple redundancy levels up to 10. This means that if a distribution has a 2-Coverage of 50%, half the points on the

Listing 4.4: Percentage of area covered by at least one sensor metric in single and group settings

```
Covered area: 74.57%

Mean covered area: 67.37%
```

scenario are covered by at least two sensors. In a group setting, it displays the same information but the average of the scenarios. This metric is the most important when there is a need to have more than one sensor covering each of the points, providing more robustness when a sensor is turned off. It can also be a good indicator if there is a lot of overlap between the sensors in the case no redundancy is needed.

In Listing 4.5, it is possible to see the metric results in both single and group settings.

Listing 4.5: Percentage of area covered by multiple number of sensors metric in single and group settings

```
1-Coverage: 75.22%
2-Coverage: 43.48%
3-Coverage: 20.15%
4-Coverage: 7.48%
5-Coverage: 1.47%
6-Coverage: 0.17%
7-Coverage: 0.00%
8-Coverage: 0.00%
9-Coverage: 0.00%
10-Coverage: 0.00%

Mean 1-Coverage: 64.50%
Mean 2-Coverage: 30.81%
Mean 3-Coverage: 10.61%
Mean 4-Coverage: 1.99%
Mean 5-Coverage: 0.22%
Mean 6-Coverage: 0.02%
Mean 7-Coverage: 0.00%
Mean 8-Coverage: 0.00%
Mean 9-Coverage: 0.00%
Mean 10-Coverage: 0.00%
```

Another more complex metric is *MetricSensorEfficiency*. This metric shows the percentage ratio of the area covered against the full potential from all sensors. This means that if, for example, only one sensor is deployed and it is on the edge of the environment, the efficiency will be near 50% since almost half the coverage potential is wasted against the environment boundary. Also, the efficiency will be lower if there is much overlap between the sensors. When the metric deals with test groups, it displays their mean efficiency. This is an important metric because it shows

how the available resources (sensors) are being used. It also can be a good tiebreaker metric between two results, for example, when the coverage percentage is the same.

In Listing 4.6, it is possible to see the metric results in both single and group settings.

Listing 4.6: Sensors efficiency metric in single and group settings

```
Sensors efficiency: 43.51%

Mean sensors efficiency: 35.01%
```

One another important metric is *MetricProcessingTime*. This metric shows the time it took to distribute all the sensors in the specific environment in milliseconds. When the metric deals with test groups, it displays their accumulated processing time. This is a relevant metric because it makes it possible to see which distribution algorithms are more suited for specific usages depending on the time it takes to achieve a solution.

In Listing 4.7, it is possible to see the metric results in both single and group settings.

Listing 4.7: Processing time in single and group settings

```
Processing time: 1068ms

Accumulated processing time: 30817ms
```

The final and most complex metric implemented is the *MetricMinimumCoverageOneSensorLoss*. This metric shows the minimum coverage possible if one of the sensors gets shut down. When the metric is run in a group, it shows the mean minimum coverage of all cases. This could be an important metric to test the robustness and redundancy of a sensor distribution.

In Listing 4.8 it is possible to see the metrics results in both single and group settings.

Listing 4.8: Coverage when a sensor is loss metric in single and group settings

```
Minimum coverage with a sensor loss: 91.58%

Mean minimum coverage with a sensor loss: 57.37%
```

4.4 Algorithms

One of the most critical parts of the whole program is the algorithms. In this component, most research is made, mainly because it is here that the most complex logic is concentrated. During the project, many WSN distribution algorithms were implemented, the majority being inspired by the state-of-the-art and some with small or big modifications from their respective papers. In this section, we will explore all the implemented algorithms in detail. However, it is essential to note

that some publications did not provide enough detail on the algorithms' implementation, which prevented reproducibility. Also, we assume that the communication distance of the sensors is big enough not to be a limitation to better focus on the area coverage of the setup.

The *Random* algorithm is the simplest of all implemented algorithms. It has an initial fixed number of sensors that will deploy. This number is calculated by dividing the area of the environment by a fraction of the coverage area of one sensor, in this case, 6/10, since it is a good approximation of the optimal coverage ratio considering the optimal overlap of sensors. After having the number of sensors, a random position inside the environment is assigned for each sensor. If a sensor's assigned random position is inside an obstacle, no new position is defined, and the sensor does not contribute to the coverage of the area. In Fig 4.8 the results from this algorithm in two scenarios, the 1200x800 empty one, and the house blueprint, can be seen. It is possible to see that the algorithm is inefficient and inconsistent, having in some portions many sensors and in others none.

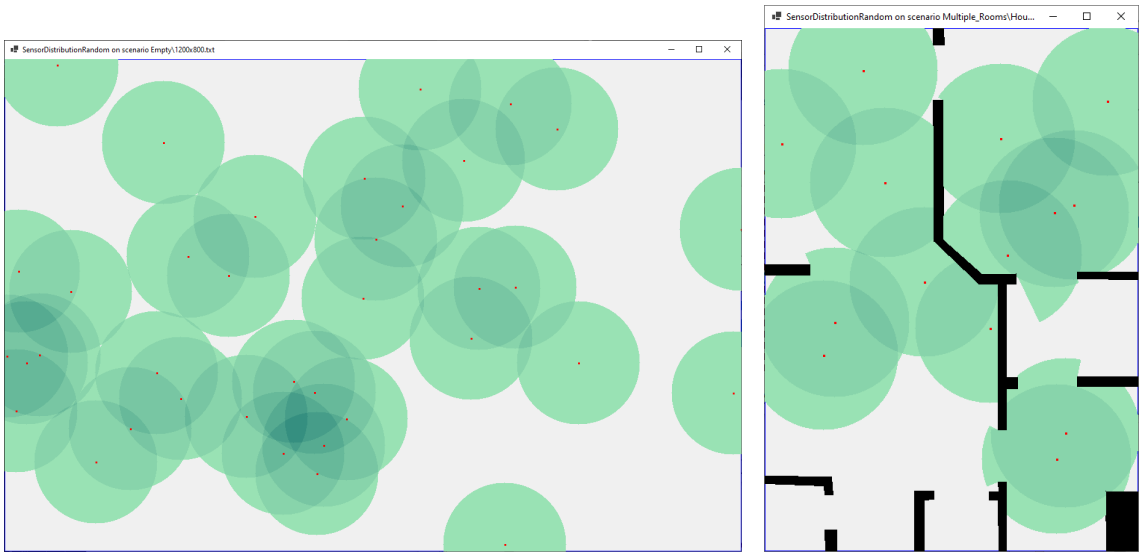


Figure 4.8: Random algorithm in two scenarios

4.4.1 Geometrical Patterns Algorithms

The *ORRD* algorithm is based on the solution developed in [Chang et al., 2009]. Since it was impossible to access the source code of their implementation, we tried to reproduce it to the best of our abilities. The algorithm consists of starting the deployment in a point of the environment, in this case, the top left, and from there, always making successive predefined moves. There are six possible moves to make, and there is an order to check if each move is possible. Fig. 2.5a shows the possible moves to make. When a move is impossible, the algorithm first tries to make a shorter move, but with a minimum length, in case of a collision with an obstacle or environment boundary. Fig. 4.9 shows some of those shorter moves being made because of the collision with an obstacle. If it is impossible to make either the normal move or the shorter one, it will try to

make the next move in order. The algorithm terminates when it is impossible to make more moves from none of the currently placed sensors.

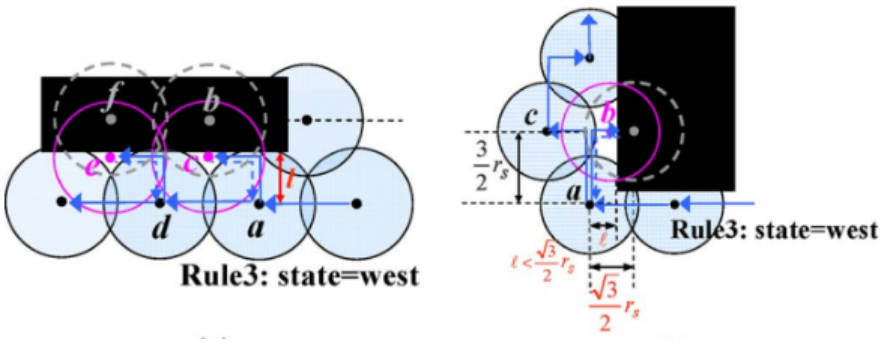


Figure 4.9: Shorter movements made in the ORRD solution

In Fig 4.10 the results from this algorithm in two scenarios can be seen. It is possible to see that the algorithm's performance is very efficient and with full coverage in the test case without any obstacles. In the House test case, the performance is lacking in coverage, possibly because certain regions exist where the algorithm could not distribute any sensor.

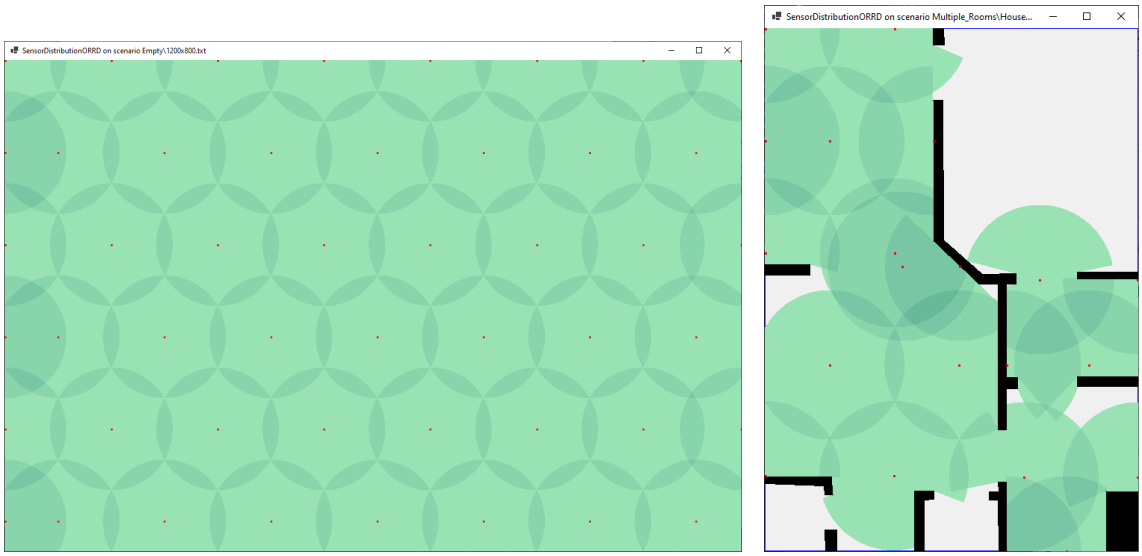


Figure 4.10: ORRD algorithm in two scenarios

To validate our implementation of the *ORRD* algorithm, we tested it in our framework in a similar environment to the original paper. The original paper was tested in an empty scenario with 400mx400m dimensions and the sensors had a radius of 20m, and it achieved complete coverage with 174 nodes. Our implementation in the same scenario also achieved 100% coverage but with 176 sensors, which is close enough to the original results to be considered a good approximation of the implementation.

The *ORRD Modified - ORRD M* is similar to the previous algorithm, also based on [Chang et al., 2009] but with a slight modification. Since, in some cases with small narrow spaces, the

algorithm is unable to progress further, as can be seen in Fig. 4.10, we modified it to try to solve this problem. Since the regular movements made in this algorithm form a grid of sensors in the environment with no obstacles, we opted to modify it to use all of those points as initial points. This modification makes it so that all the sensors do not need to be connected originating from the predefined movements.

In Fig 4.11 the results from this algorithm in two scenarios can be seen. We can see that the performance in the empty scenario is the same. In the House scenario, some regions with no sensors in the unmodified version are now covered. However, it is possible to see that some smaller zones are still without sensors.

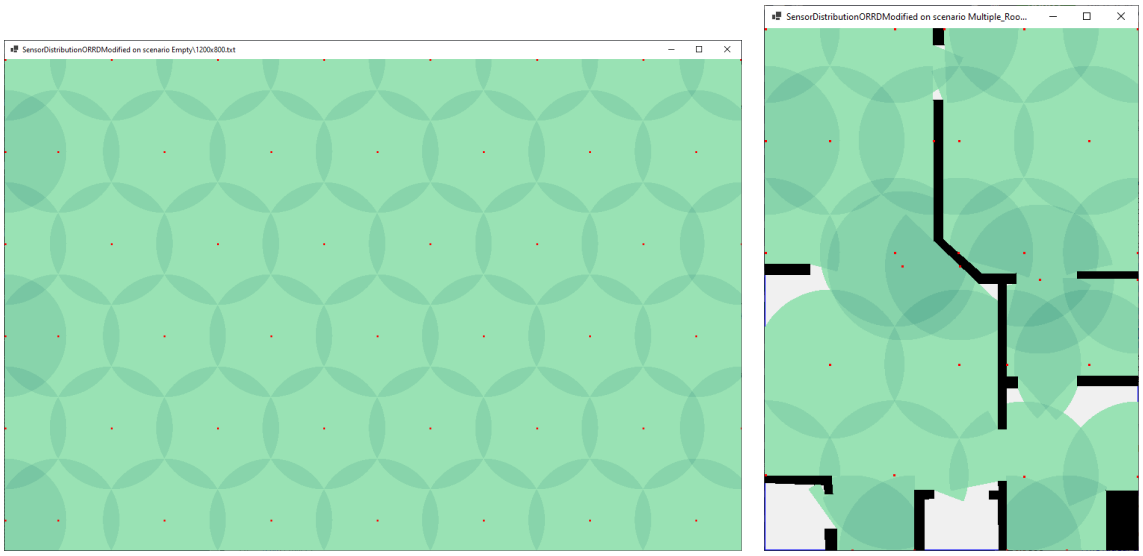


Figure 4.11: ORRD Modified algorithm in two scenarios

4.4.2 Virtual Force Algorithms

The *OAVFA* algorithm is based on the solution developed in [Rout and Roy, 2016]. Since it was impossible to access the source code of their implementation, we tried to reproduce it to the best of our abilities. The algorithm consists of starting with a specific number of sensors distributed randomly in the beginning and then modifying their positions iteratively using forces until they reach their final positions. The beginning and end of this process can be seen in Fig. 2.2. During the forces iterations, three types of forces are applied to the sensors. The first one is related to the environment boundaries, where a repulsive gradual force is applied to the sensor, depending on distance, if it is closer than a certain threshold to the boundaries. The second force is with the obstacles, where a repulsive gradual force is applied to the sensors when they are closer than a threshold, similar to the previous force. The third and final force is between the sensors themselves. This force is proportional and can be repulsive or attractive, depending on the distance between sensors. If the distance between two sensors is less than a threshold, the force is repulsive. Otherwise, if it is greater, the force is attractive. All of these forces are accumulated in an

iteration, and the sensor moves in their direction. If the sum of the forces is near zero, a stable counter is incremented, and when this counter reaches a certain number, the sensor is considered stable and in its final position. The algorithm terminates when all the sensors are stable or after a certain number of iterations.

In Fig 4.12 the results from this algorithm in two scenarios can be seen. From both test cases, it is seen that they are not fully covered. However, its efficiency seems good. In the empty scenario, it is possible to see that the sensors do not show much overlap, which is suitable for efficiency. In the House case, some parts are fully uncovered, but the covered ones seem to be spreading the sensors evenly.

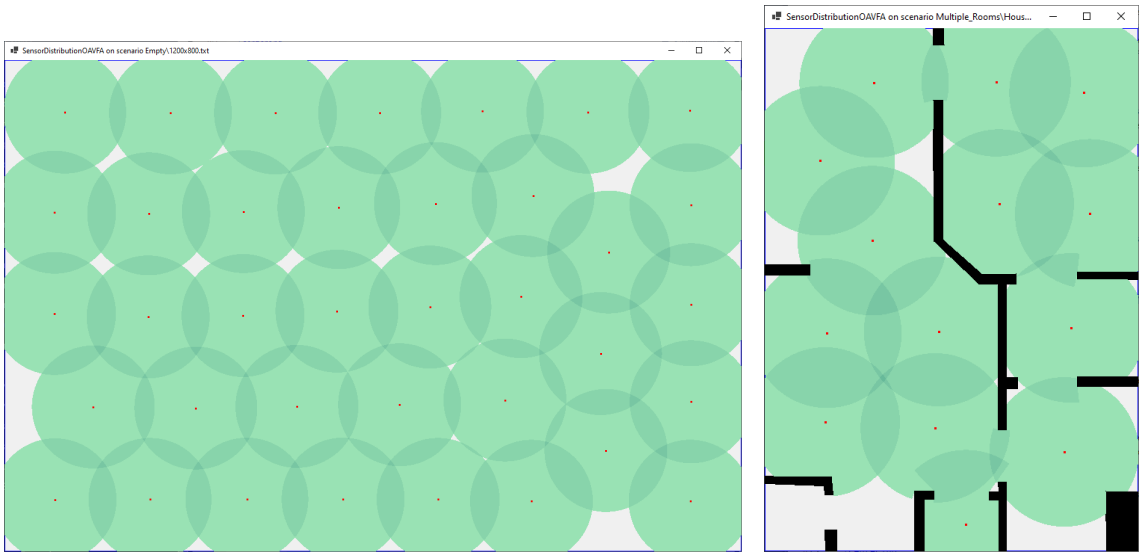


Figure 4.12: OAVFA algorithm in two scenarios

To validate our solution and implementation of the state-of-the-art *OAVFA* algorithm, we ran it in a similar environment to one in its paper. The environment consisted of an area of 100mx100m with an obstacle in the form of a *T* in the middle, and the sensor's radius was 10m. The paper solution with 40 sensors achieved a coverage rate of 98.29%, and our implementation with the same number of sensors achieved 97.07% of coverage, which is a close number to consider a good approximation to the original algorithm.

The *OAVFA Targeted - OAVFA T* is similar to the previous algorithm, also based on [Rout and Roy, 2016] with some modifications. The initial phase of the algorithm is the same as the original, where a predefined number of sensors is placed randomly on the scene. The iterations to move the sensors' locations are the same as the original. However, a new higher-level iteration exists where sensors are added in the case of existing uncovered areas in the scenario. This modification makes the algorithm add a sensor if the ones used are not enough, then proceeds to do some iterations of moving the sensors. If there are still uncovered areas, this process is repeated until the area is fully covered. In Fig 4.13 the results from this algorithm in two scenarios can be seen. Both scenarios show a 100% coverage area and appear to have a good efficiency since there does not seem to have much sensors' overlap.

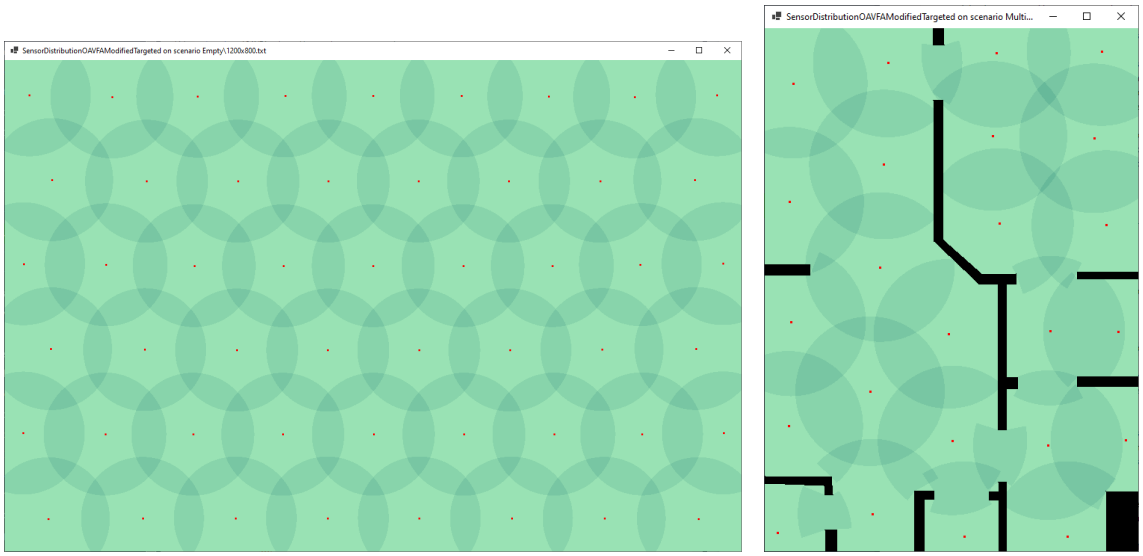


Figure 4.13: OAVFA Targeted algorithm in two scenarios

The *OAVFA Corner Targeted - OAVFA CT* is a modification of the *OAVFA T* algorithm, which is based on [Rout and Roy, 2016]. The difference between this algorithm and the previous is that it creates an attractive force between obstacles and the sensors in some scenarios. This new attractive force is created when a sensor is closer than a certain threshold to an inner corner of an obstacle, in this case, any corner that makes less than 180º degrees outside. This force also considers the degree of the corner, this being inverted proportional to the degrees of the corner. If the degree is more acute, the force is stronger, and if it is more obtuse, it is less noticeable. This force tries to fight against the normal repelling force from the obstacles, making it easier for sensors to move into narrow corners of the environment.

In Fig 4.14 the results from the parent algorithm *OAVFA T* and the *OAVFA CT* algorithm in the Decreasing Width scenario can be seen respectively. From the shown distribution, it is possible to see that the Corner modification is more efficient, possibly because of the existence of such an acute angle that pushes the sensors away from it on the unmodified version. Since the Corner version also has an attractive force from the inner angles, this pushing effect is better neutralized.

The *OAVFA ORRD Targeted - OAVFA OT* is a modification of the *OAVFA T* algorithm, which is based on [Rout and Roy, 2016], with a modification inspired by [Chang et al., 2009]. The algorithm is equal to *OAVFA T* in all aspects except in the initial distribution of sensors. Instead of using a random distribution as the original, in this modification, we try to place the nodes in the clear places that are part of the grid, as can be seen in Fig. 2.5b. Since the grid from the ORRD solution is almost optimal in avoiding overlapping zones, it could be a good starting point for the algorithm. In Fig 4.15 the results from this algorithm in two scenarios can be seen, which don't seem to be very different from the parent algorithm.

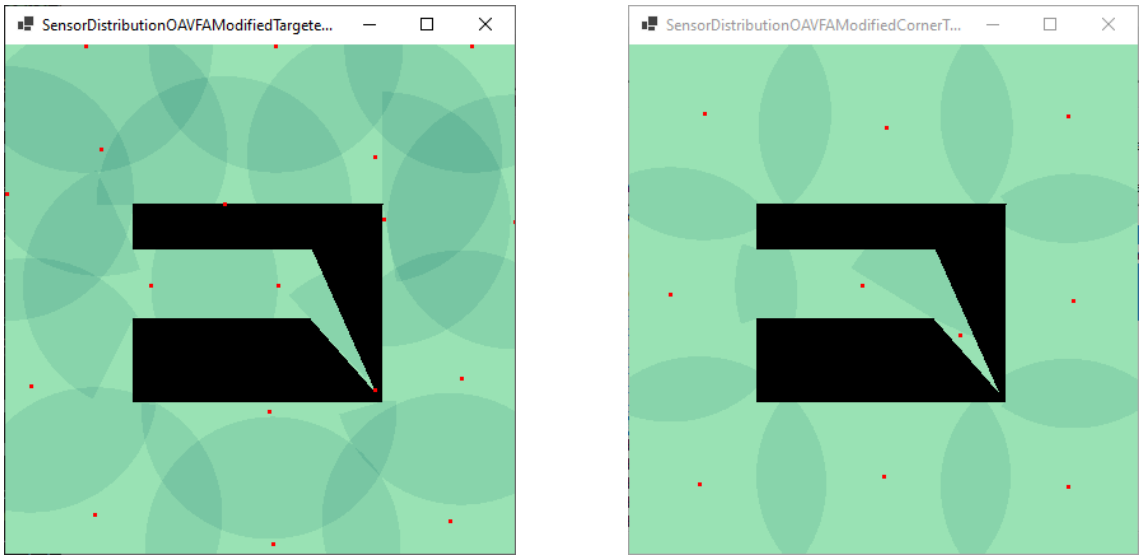


Figure 4.14: Distribution of the OAVFA T and OAVFA CT algorithms in the decreasing width scenario

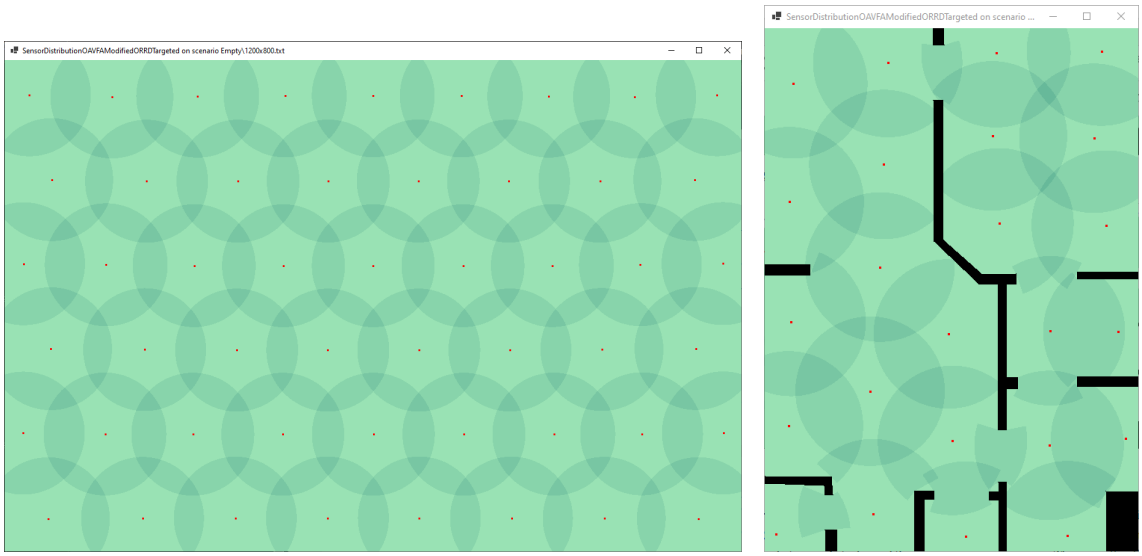


Figure 4.15: OAVFA ORRD Targeted algorithm in two scenarios

4.4.3 Computational Geometry Algorithms

The *Voronoi* algorithm is based on the solution developed in [Eledlebi et al., 2020]. In this algorithm, there exists an injection point, in this case, bottom left, where it adds new sensors if there is no sensor covering that point or there is a sensor that is disconnected from all other nodes. During each iteration, the placed sensors also move themselves taking into account their Voronoi regions between each other and the collisions with obstacles. To calculate a sensor's next position it is necessary to use the center point of the intersection between the Voronoi regions, the sensor's range, and the obstacles. The algorithm iteratively moves the sensors and adds new ones while there is an amount of movement from the sensors above a certain threshold or when the number

of iterations is less than a counter. The iteration counter is reset every time a new max coverage rate is accomplished, making sure the algorithm is only stopped when it does not achieve better results.

In Fig 4.16 the results from this algorithm in two scenarios can be seen. It is possible to see that the algorithm has a good coverage performance in the empty scenario. However, it struggles with obstacles in the environment, leaving many areas uncovered.

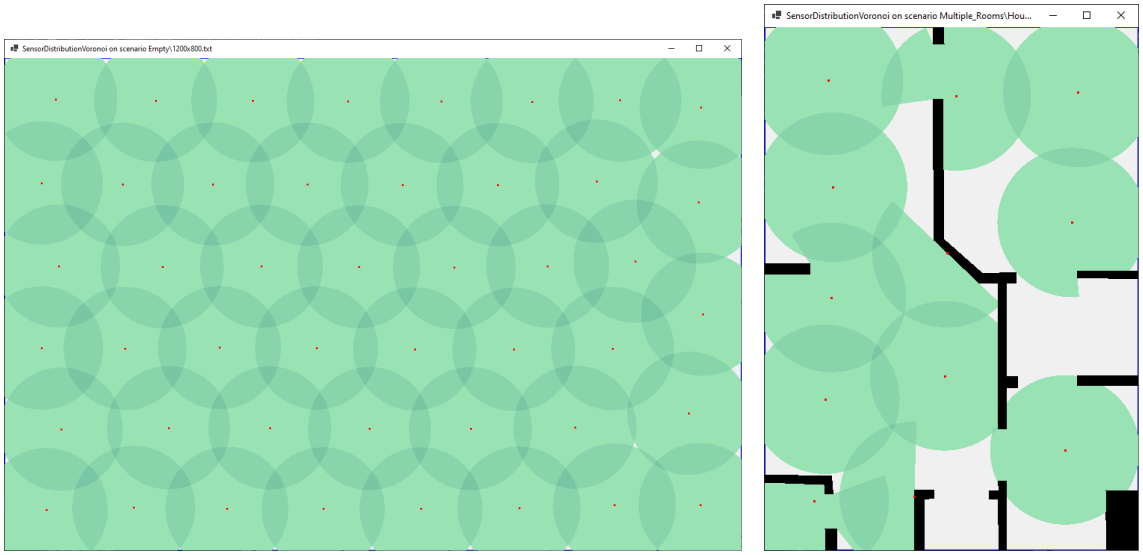


Figure 4.16: Voronoi algorithm in two scenarios

We tested using the same scenario to ensure our implementation had close results to the one in the *Voronoi* original paper, which used a 100mx100m scenario with a sensor radius of 16m. The original implementation achieved a 99.3% coverage in 20 seconds with 16 nodes. Our implementation achieved 98.49% coverage in 14 seconds with 17 nodes, which are similar results making it a good approximation of the original algorithm.

The *Voronoi Random - Voronoi R* is similar to the previous algorithm, also based on [Eledlebi et al., 2020] with some modifications. The big modification from the original algorithm is that the injection point is randomly assigned as a random point in the environment each iteration instead of always being the same bottom-left point. This modification could make the sensors reach some difficult points and make it easier for the sensors to spread through the environment.

In Fig 4.17 the results from this algorithm in two scenarios can be seen. The coverage in the empty scenario seems a bit better but is still not fully covered. In the house scenario, we can see a big improvement in the coverage, covering some small areas left without sensors in the unmodified version.

The *Voronoi Targeted - Voronoi T* is also based on [Eledlebi et al., 2020] with some modifications relative to the way of inserting nodes. In this algorithm, the way the sensors move is the same as the original. However, during the iterations, sensors are added differently. In this case, when a new sensor is to be added because the whole area is still not covered, the algorithm searches for

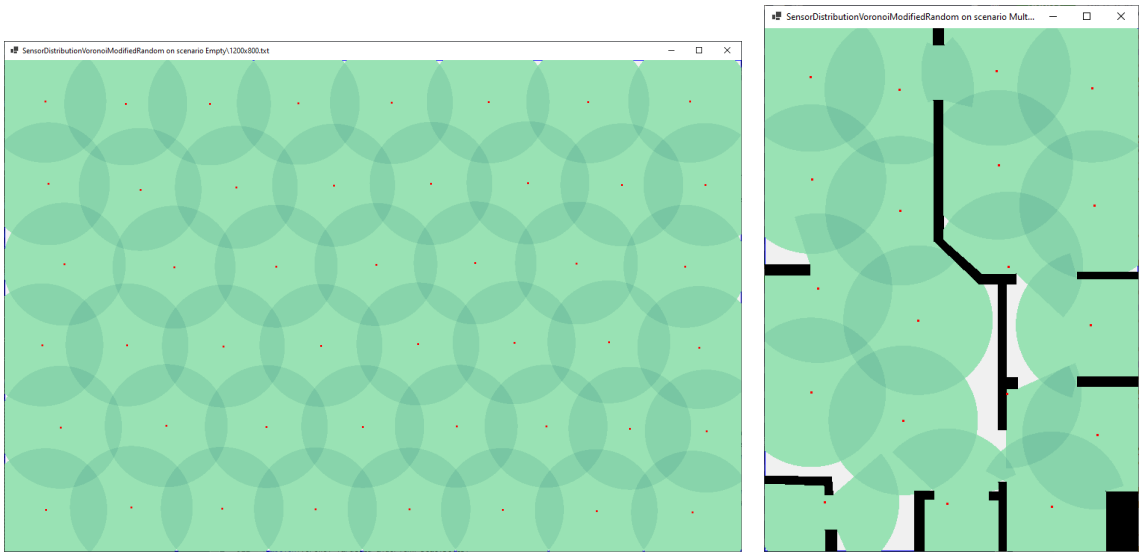


Figure 4.17: Voronoi Random algorithm in two scenarios

a non-covered point and adds a sensor there. This modification was expected to guarantee 100% coverage of all areas by the sensors.

In Fig 4.18 the results from this algorithm in two scenarios can be seen. It is possible to see that both scenarios show a distribution with full coverage. It is also noticeable that in the empty scenario, the efficiency of the sensors seems high without much overlap. However, we see much overlap in the scenario with obstacles, which diminishes efficiency.

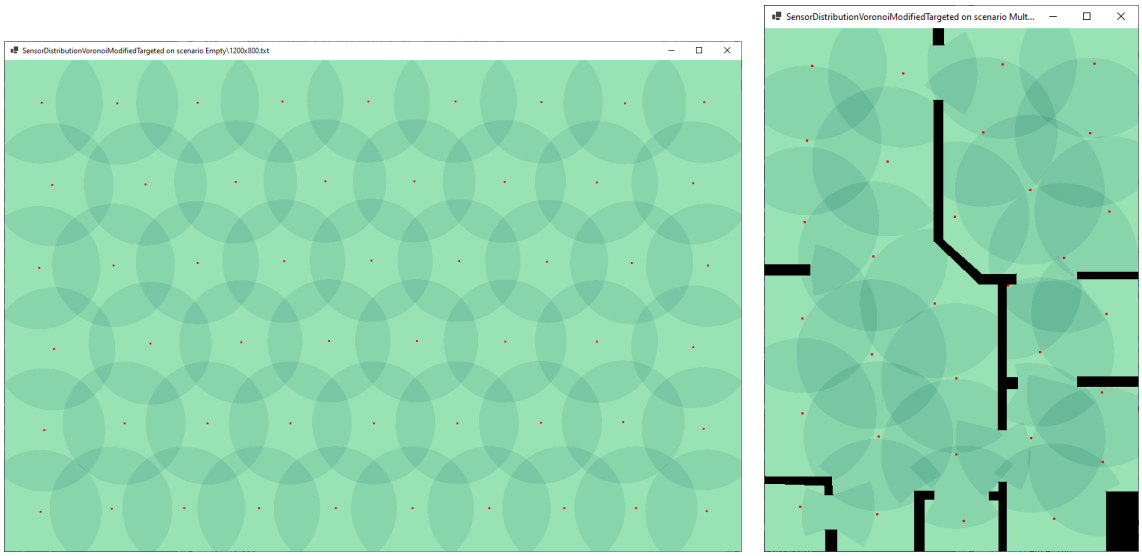


Figure 4.18: Voronoi Targeted algorithm in two scenarios

The *Voronoi Targeted Unlimited - Voronoi TU* is a modification of the algorithm *Voronoi T*, also based on [Eledlebi et al., 2020]. The difference between this algorithm and the parent and original one is that when calculating the next position of a sensor, it does not consider the sensor's range. This modification ensures that the sensor's next position is the centroid of the area created

by the intersection of the sensor's Voronoi region with the area without obstacles. This modification aims to make a more evenly spread of the sensors through all the scene areas and use fewer sensors while having 100% coverage. In Fig 4.19 the results from this algorithm in two scenarios can be seen, which don't seem to be very different from the parent algorithm.

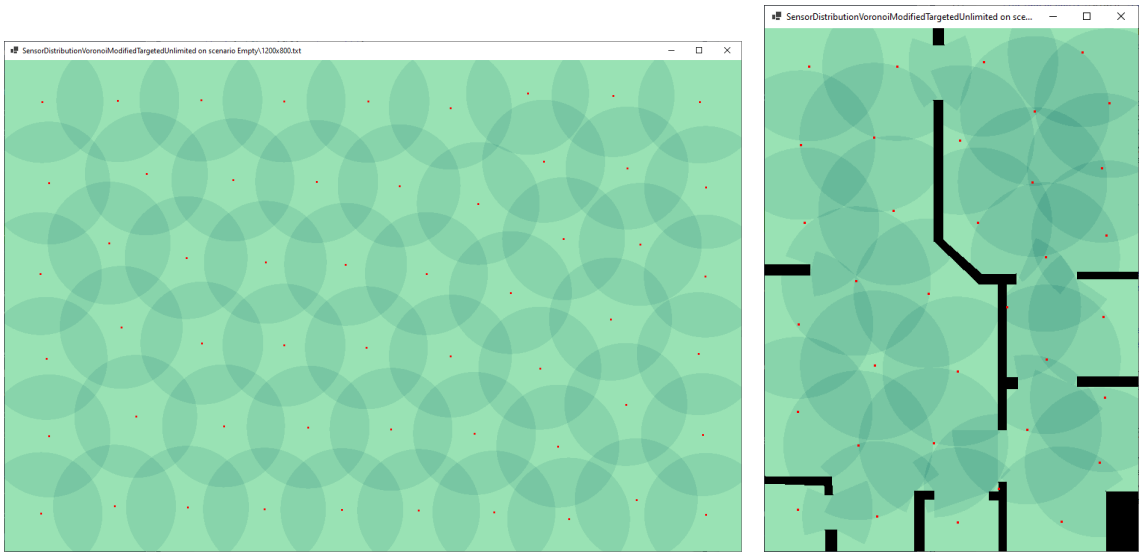


Figure 4.19: Voronoi Targeted Unlimited algorithm in two scenarios

The *Voronoi Mix - Voronoi M* is a modification that mixes the algorithms *Voronoi T* and *Voronoi R*, all based on [Eledlebi et al., 2020]. In this algorithm, the beginning is run based on the Random variant, where in each iteration, a new random point is assigned as the injection point. If no sensor covers that point, a new sensor is added at that position. When the algorithm achieves 99% area coverage by all sensors, it switches to the Targeted variant. In the Targeted variant, after several iterations, the algorithm does a linear search of the environment to check for uncovered points and add sensors to it. This algorithm came to life because the Random variant had relatively fewer sensors with almost 100% coverage but not reaching it, so it tries to take advantage of its seemingly good performance but with 100% coverage.

In Fig 4.20 the results from this algorithm in two scenarios can be seen. The efficiency performance in the empty scenario seems almost the same as the parent algorithm. However, the house scenario seems less efficient, having more overlap in the sensors and some even being redundant.

The *Voronoi ORRD Targeted - Voronoi OT* is a modification of the algorithm *Voronoi T* based on [Eledlebi et al., 2020] with a modification inspired by [Chang et al., 2009]. The algorithm is equal to *Voronoi T* in all aspects except in the initial distribution of sensors. Instead of starting without any sensor in the beginning iterations, in this modification, we try to place the nodes in the clear places that are part of the grid, as can be seen in Fig. 2.5b. Since the grid from the ORRD solution is almost optimal in avoiding overlapping zones, it could be a good starting point for the algorithm. In Fig 4.21 we can see the performance of this algorithm in two scenarios.

The *Voronoi ORRD Targeted Unlimited - Voronoi OTU* is a modification of the algorithm *Voronoi OT* and *Voronoi TU* based on [Eledlebi et al., 2020] with a modification inspired by

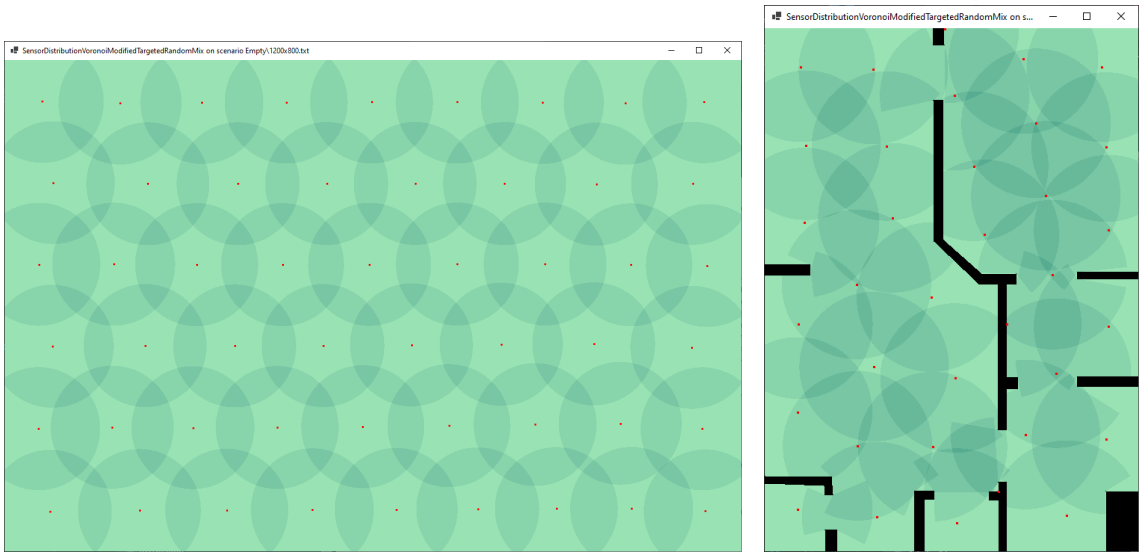


Figure 4.20: Voronoi Mix algorithm in two scenarios

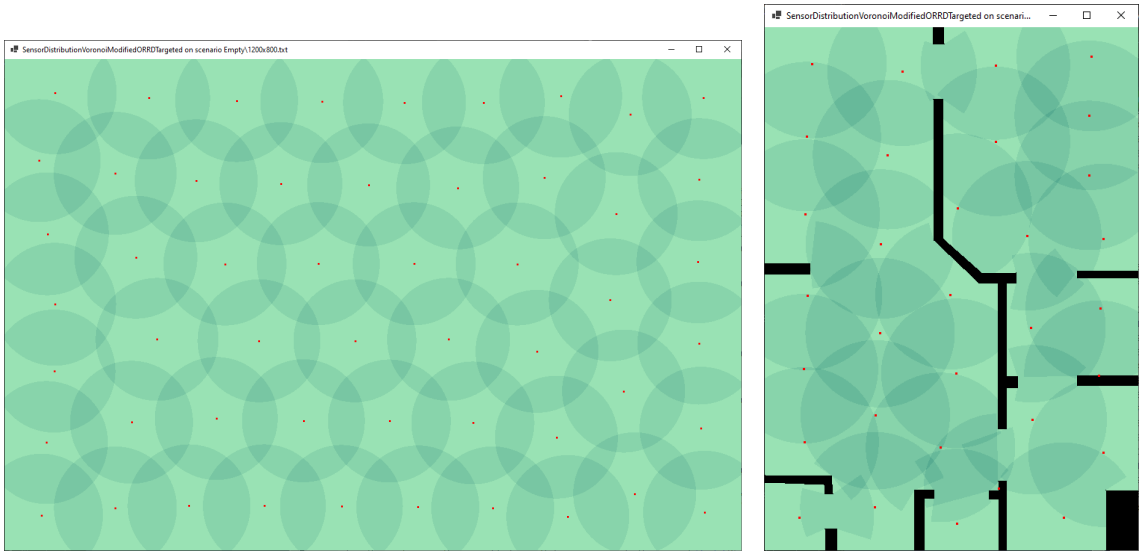


Figure 4.21: Voronoi ORRD Targeted algorithm in two scenarios

[Chang et al., 2009]. The algorithm is equal to *Voronoi OT* in all aspects, except when calculating the next position of a sensor because it does not take into account the sensor's range, as is the case in the *Voronoi TU* algorithm. In Fig 4.22 two sensor distributions from this algorithm can be seen.

The *Voronoi ORRD Targeted Delete - Voronoi OTD* is a modification of the algorithm *Voronoi OT* based on [Eledlebi et al., 2020] with a modification inspired by [Chang et al., 2009]. The algorithm is equal to *Voronoi OT* in all aspects except that in the end, after the coverage is 100%, it tries to remove sensors while trying to keep the coverage. This modification could delete some redundant sensors, which would make the algorithm more efficient. In Fig 4.23 we can see how this algorithm performs in two scenarios.

The *Voronoi Random OAVFA - Voronoi RF* is a combination of the algorithms *Voronoi R* based

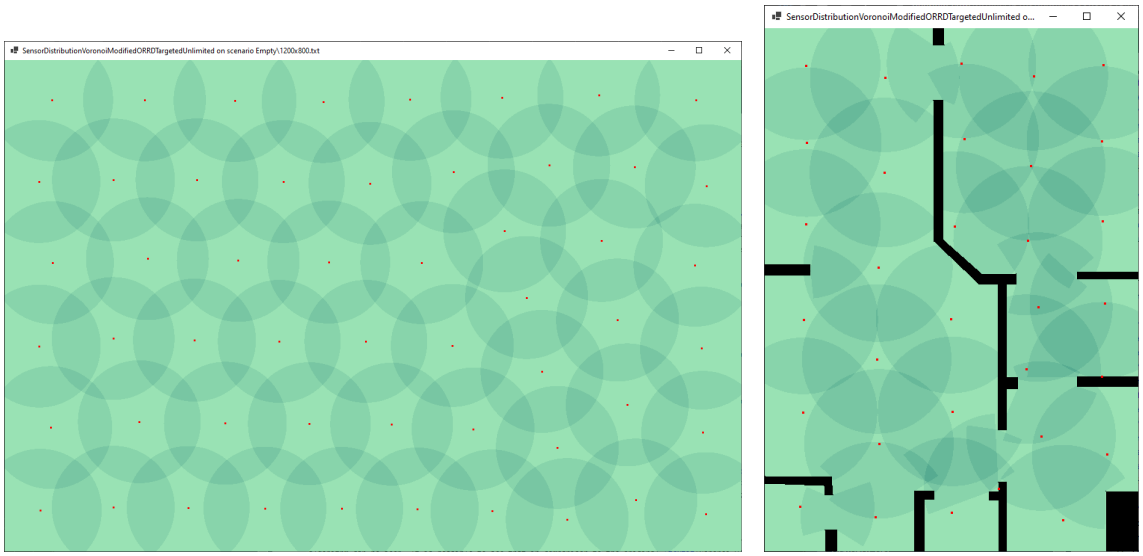


Figure 4.22: Voronoi ORRD Targeted Unlimited algorithm in two scenarios

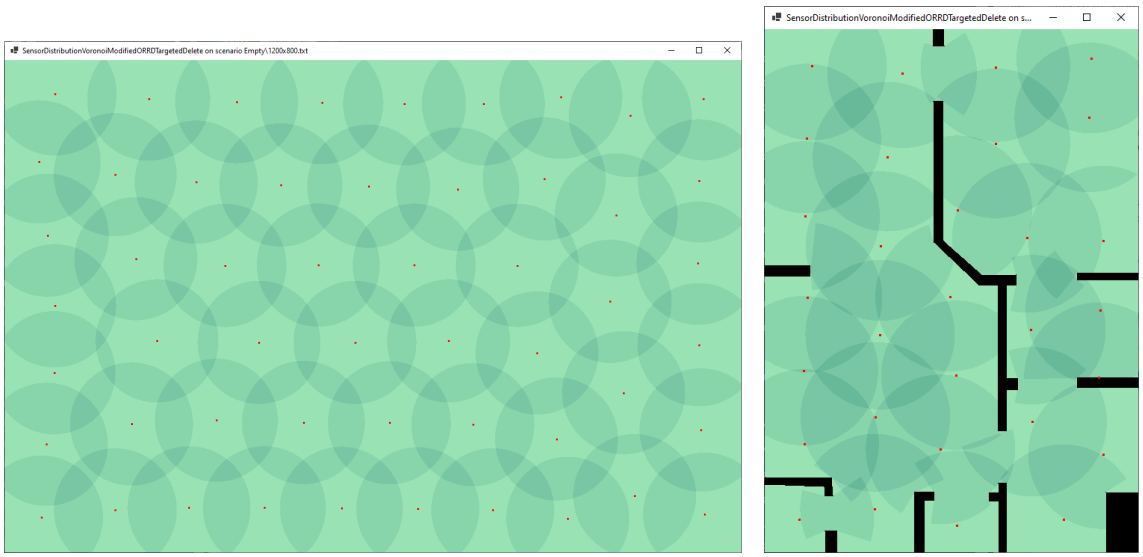


Figure 4.23: Voronoi ORRD Targeted Delete algorithm in two scenarios

on [Eledlebi et al., 2020] and the *OAVFA OT* based on [Rout and Roy, 2016]. The algorithm first runs the *Voronoi R* algorithm unmodified, which usually reaches close to 100% coverage. When this ends, it runs the *OAVFA OT* algorithm with the previous setup of sensors. This algorithm tries to take the good results but not perfect from the first algorithm and complement them with the second one. In Fig 4.24 the results from this algorithm in two scenarios can be seen. It is possible to see that the efficiency performance is good in the empty scenario, but it has a bad performance in the scenario with obstacles.

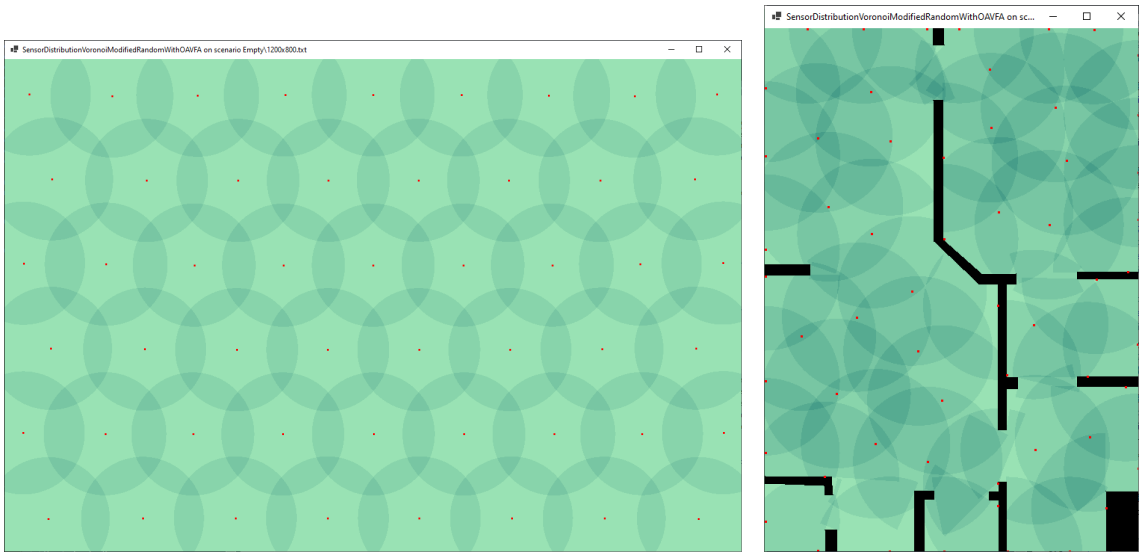


Figure 4.24: Voronoi Random OAVFA algorithm in two scenarios

Chapter 5

Results

Since we have implemented a good amount of algorithms and variations we provide Tables 5.1, 5.2, and 5.3 which summarize the statistics for all algorithms when using different radius sizes of sensors (100, 300, 600) in the different groups and in general with the aggregated results for ease of comparison. We will also analyze the algorithms' results and compare them with each other.

5.1 Miscellaneous Algorithms

The results from the *Random* algorithm are expected. It is possible to see its best results when there are little or no environmental obstacles, possibly because the algorithm does not take them into account. The statistics also show that the overall coverage performance worsens with the increase of the radius, possibly because with a bigger radius, fewer sensors are used, making it more essential to place the few sensors in good positions, which might not happen.

5.2 Geometrical Patterns Algorithms

For the *ORRD* algorithm, we can see that it works best in empty environments or more simple environments without narrow paths. These results make sense because if a narrow path occurs and the algorithm cannot pass it to get to the rest of the environment, a portion of the scene is not covered. One of these cases can be seen in Fig. 4.10 where the algorithm cannot advance to a part of the scenario. It is also possible to see that with the increase in the radius of the sensor, the algorithm performance gets worse in terms of coverage, possibly because it makes the paths more narrow in comparison, making the algorithm unable to go to more zones.

Relative to the *ORRD M*, it is possible to see a significant improvement in coverage compared to the original *ORRD*, especially in the groups where it struggled. However, the algorithm still works best in simple scenarios where there are not a lot of obstacles or small spaces, possibly making it less viable for the remaining types of scenarios. Similar to the original algorithm, the

coverage % is worse with a bigger radius, possibly because more uncovered areas appear because of the smaller obstacles in comparison.

All the *Geometrical Patterns* algorithms show a good coverage % performance in the empty and with one obstacle scenarios, having the prior 100% coverage with the radius of 100. Since these algorithms require a lot less computing time than the majority of the other types of solutions, they could be taken as a good initial solution and, in case of poor performance in coverage, be replaced by some more intensive solutions.

5.3 Virtual Force Algorithms

The *OAVFA* algorithm shows relatively good and stable results in terms of coverage in all test groups, especially when using a smaller radius. Its performance might not be as good because there might not be enough nodes to cover the whole area, especially in more complex environments. When using a bigger radius, this effect increases, and the results decline.

The results from the *OAVFA T* algorithm are excellent compared to the original algorithm *OAVFA*. This algorithm can achieve the 100% coverage percentage in all the scenarios with all the tested sensor radii, which makes sense since the nodes are added to the uncovered areas detected during its iterations. It is possible to see that its performance in terms of efficiency seems better when the radius is bigger compared to the other algorithms in general. Also, it is a fast algorithm to run, which can be an important point compared to, for example, the *Voronoi OTD* algorithm.

Relative to the results of the *OAVFA CT* algorithm, they are a bit lacking compared to the original algorithm *OAVFA T*. In some cases, it has better efficiency performance, but it is counterproductive to have this modification in general. This performance also does not change with different radii, possibly indicating that the forces between sensors and obstacles were already in good balance in the original algorithm.

For the *OAVFA OT* algorithm, it is possible to see that its results are very similar to the original *OAVFA T*. However, in the case of using a smaller radius, this algorithm is better in terms of efficiency than its counterpart but loses its advantage with the bigger radius. These results could indicate that the initial deployment is more beneficial when using a smaller radius and prejudicing otherwise.

The *Virtual Force* algorithms seem consistent algorithms that show promising results in terms of coverage % having a lot of them 100% on all test cases. Its efficiency is best when using a bigger sensor radius of 300 or 600 when compared to the other types of algorithm families. In terms of processing time, it is much worse than the *Geometric Patterns* counterpart, but it is similar to the *Computational Geometry* family. So, in conclusion, this family of algorithms should be used when the sensor radius is big in proportion to the environment since it has the best average results of all the families in terms of sensor coverage and efficiency.

5.4 Computational Geometry Algorithms

The *Voronoi* algorithm shows promising results in some groups, but none stand out that much. One aspect worth noting is that the algorithm's efficiency is high even when there are a lot of sensors and high coverage, making it possibly have significant potential to have good results. However, adding new sensors to the scenarios seems complicated, especially when using a bigger radius, which could be a limitation.

When it comes to the *Voronoi R* algorithm, it is possible to see that the results have significantly improved from the original, having high coverage and a good efficiency score for all groups. This increase is more noticeable in the smaller radius cases, where all the groups have 95+% coverage than in the bigger ones. However, even though the results are good, it cannot guarantee full area coverage.

The results from the *Voronoi T* algorithm are excellent compared to the original *Voronoi* algorithm. As expected, it achieves 100% coverage in all test cases with all radii since it adds sensors to unmonitored zones. This algorithm, in terms of efficiency, has better results when using smaller sensor radii, and it falls a bit behind when using bigger ones compared to similar algorithms.

Relative to the results from *Voronoi TU*, it is possible to see that the results are almost the same as the original algorithm *Voronoi T*. It also achieves full coverage, and the number of sensors used and its efficiency is very similar. These results could indicate that the change to not taking into account the radius of the sensor in the movement phase does not affect the results significantly.

For the *Voronoi M* algorithm, its results are almost equal and, in some cases, a bit worse in terms of efficiency than one of the parent's algorithms *Voronoi T*. Even though the algorithm tried to get better results using the Random variant from the *Voronoi R*, it did not have any improvement from it. So, in the end, this modification was not that successful since it has the same distribution results as its original algorithm, with the disadvantage that it is now much slower in processing time.

The *Voronoi OT* algorithm shows a bit of efficiency improvement when using a smaller sensor radius than the parent algorithm *Voronoi T*. However, when used in situations with a bigger sensor radius, its improvement is diminished, possibly because its initial placement is not as good, as seen in the *OAVFA OT* algorithm. These variations indicate that the initial setup influences the obtained results and that having a good initial placement is crucial.

When it comes to the *Voronoi OTU* algorithm, it is possible to see across the tables that the algorithm has a bit worst performance than its parent algorithms *Voronoi OT* and *Voronoi TU* in terms of efficiency. We can conclude that the mix between the two algorithms does not improve any tested cases with the tested sensors' radii.

The results from the *Voronoi OTD* algorithm are the best achieved in all algorithms when taking coverage percentage, the number of sensors used, and efficiency into account. These results are also great, considering that its performance is good across all the sensor radii. These stats make sense since the algorithm, in the end, tries to remove sensors that might be redundant, which can

increase efficiency. One disadvantage of this algorithm is the time it takes to distribute the sensors, which can be almost 30 times slower than the parent algorithm *Voronoi OT*.

Finally, it is possible to see that the *Voronoi RF* algorithm has mixed results. When using small radii, the algorithm has worst results in terms of efficiency and processing time than, for example, its *Voronoi T* counterpart. However, when using bigger radii has improved the efficiency metric performance. These results might make sense since this algorithm also uses forces, and the results of the *OAVFA* force algorithm show better results with a big radius. When using the smaller radius, the force algorithm part might bottleneck the results of this algorithm.

From all the *Computational Geometry* algorithms, it is possible to see that they have incredible results in terms of coverage % and efficiency. Most of the algorithms achieve 100% coverage in all scenarios with all sizes of sensors and are more efficient than the rest of the families when using a small sensor radius. Their processing times are a bit more dispersed but, on average, are similar to the *Virtual Force* ones. So this family of distribution algorithms should be used when the sensor's radius is small in comparison to the scene since they are the ones that have the best efficiency in those types of situations.

Table 5.1: All algorithms statistics with sensor radius 100. The mean results followed by the standard deviation of five runs are presented in each cell. The results are categorized by the algorithms and the different groups run. The groups are Empty, Multiple Obstacles, Multiple Rooms, One Obstacle, and Tight Corridors. The General column shows the aggregated results of all the groups.

Algorithm	Metric	Empty	M. Obst.	M. Rooms	1 Obst.	T. Corr.	General
Random	Cov%	67.9 | 3.5	60.6 | 3.0	57.0 | 4.6	57.0 | 4.0	57.1 | 3.9	59.8 | 0.6
	Num.S.	15.8 | 0.0	8.8 | 0.0	11.5 | 0.0	10.0 | 0.0	9.0 | 0.0	10.9 | 0.0
	Eff%	53.4 | 2.7	43.0 | 2.2	40.7 | 3.1	38.8 | 2.7	30.4 | 1.9	40.8 | 0.6
	T(s)	0.0 | 0	0.0 | 0	0.0 | 0	0.0 | 0	0.0 | 0	0.0 | 0
ORRD	Cov%	100 | 0.0	70.1 | 0.0	36.0 | 0.0	98.4 | 0.0	31.7 | 0.0	68.2 | 0.0
	Num.S.	23.4 | 0.0	12.2 | 0.0	10.8 | 0.0	21.2 | 0.0	6.0 | 0.0	14.8 | 0.0
	Eff%	42.7 | 0.0	29.2 | 0.0	26.8 | 0.0	32.0 | 0.0	7.6 | 0.0	27.1 | 0.0
	T(s)	0.0 | 0	0.0 | 0	0.0 | 0	0.0 | 0	0.0 | 0	0.0 | 0
ORRD M	Cov%	100 | 0.0	95.9 | 0.0	94.8 | 0.0	99.6 | 0.0	92.1 | 0.0	96.5 | 0.0
	Num.S.	23.6 | 0.0	19.0 | 0.0	26.5 | 0.0	22.2 | 0.0	13.5 | 0.0	20.5 | 0.0
	Eff%	42.5 | 0.0	32.3 | 0.0	29.4 | 0.0	31.1 | 0.0	33.0 | 0.0	33.7 | 0.0
	T(s)	0.0 | 0	0.0 | 0	0.0 | 0	0.0 | 0	0.0 | 0	0.1 | 0
OAVFA	Cov%	97.0 | 0.2	80.7 | 1.8	78.6 | 1.0	90.8 | 1.8	82.8 | 3.3	86.3 | 1.1
	Num.S.	15.8 | 0.0	8.8 | 0.0	11.5 | 0.0	10.0 | 0.0	9.0 | 0.0	10.9 | 0.0
	Eff%	76.4 | 0.1	57.3 | 1.2	56.0 | 0.9	61.7 | 1.2	42.9 | 1.8	58.5 | 0.7
	T(s)	0.2 | 0	1.9 | 0	2.6 | 0	0.7 | 0	0.3 | 0	5.7 | 1

OAVFA T	Cov%	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0
	Num.S.	22.5 ± 0.1	18.9 ± 0.2	26.8 ± 0.7	15.8 ± 0.1	17.3 ± 0.4	19.7 ± 0.1
	Eff%	50.8 ± 0.1	36.5 ± 0.5	30.9 ± 0.7	43.3 ± 0.3	26.9 ± 0.9	37.7 ± 0.3
	T(s)	23.9 ± 4	21.0 ± 3	72.7 ± 10	17.0 ± 1	9.7 ± 1	144 ± 10
OAVFA CT	Cov%	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0
	Num.S.	22.5 ± 0.1	18.6 ± 0.2	26.6 ± 1.1	15.8 ± 0.3	18.1 ± 0.8	19.8 ± 0.1
	Eff%	50.9 ± 0.1	36.8 ± 0.8	31.4 ± 1.2	43.3 ± 0.7	25.9 ± 2.1	37.7 ± 0.3
	T(s)	21.9 ± 3	21.7 ± 4	76.2 ± 10	16.9 ± 1	11.8 ± 2	149 ± 12
OAVFA OT	Cov%	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0
	Num.S.	22.5 ± 0.1	19.4 ± 0.5	24.9 ± 1.5	16.7 ± 0.5	19.5 ± 0.9	20.2 ± 0.3
	Eff%	50.8 ± 0.1	34.9 ± 1.1	33.5 ± 2.1	41.9 ± 0.9	24.5 ± 1.3	37.0 ± 0.5
	T(s)	24.0 ± 3	19.4 ± 2	52.2 ± 3	16.6 ± 2	12.4 ± 1	125 ± 8
Voronoi	Cov%	99.7 ± 0.0	71.0 ± 0.0	51.9 ± 0.0	92.2 ± 0.0	76.2 ± 0.0	79.7 ± 0.0
	Num.S.	19.0 ± 0.0	7.0 ± 0.0	6.8 ± 0.0	10.2 ± 0.0	5.2 ± 0.0	9.6 ± 0.0
	Eff%	62.8 ± 0.0	62.3 ± 0.0	63.5 ± 0.0	61.7 ± 0.0	68.1 ± 0.0	63.8 ± 0.0
	T(s)	138 ± 10	38.5 ± 3	78.1 ± 9	72.1 ± 9	45.8 ± 13	372 ± 15
Voronoi R	Cov%	99.9 ± 0.0	99.4 ± 0.1	96.2 ± 0.9	99.9 ± 0.0	99.8 ± 0.1	99.2 ± 0.2
	Num.S.	19.6 ± 0.4	12.3 ± 0.3	16.4 ± 0.7	12.9 ± 0.3	9.6 ± 0.2	13.8 ± 0.2
	Eff%	60.3 ± 1.9	50.8 ± 1.1	48.2 ± 1.8	52.6 ± 1.0	47.8 ± 0.8	52.0 ± 0.5
	T(s)	85.0 ± 16	108 ± 19	314 ± 79	136 ± 30	58.1 ± 7	701 ± 60
Voronoi T	Cov%	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0
	Num.S.	22.8 ± 0.3	15.2 ± 0.2	23.8 ± 0.0	15.2 ± 0.0	12.2 ± 0.0	17.3 ± 0.1
	Eff%	54.8 ± 0.3	43.4 ± 0.6	36.0 ± 0.0	45.5 ± 0.0	38.4 ± 0.0	43.8 ± 0.1
	T(s)	36.0 ± 6	17.8 ± 1	48.1 ± 4	15.5 ± 1	9.8 ± 2	127 ± 7
Voronoi TU	Cov%	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0
	Num.S.	23.4 ± 0.0	15.1 ± 0.1	25.2 ± 0.0	15.2 ± 0.0	11.7 ± 0.0	17.5 ± 0.0
	Eff%	54.2 ± 0.0	43.2 ± 0.3	33.6 ± 0.0	45.5 ± 0.0	39.4 ± 0.0	43.5 ± 0.1
	T(s)	20.7 ± 4	11.1 ± 1	45.0 ± 5	13.3 ± 4	6.6 ± 1	96.6 ± 12
Voronoi M	Cov%	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0
	Num.S.	22.8 ± 0.1	15.9 ± 1.3	24.2 ± 0.5	15.1 ± 0.1	11.6 ± 0.5	17.3 ± 0.2
	Eff%	54.4 ± 1.0	42.4 ± 1.2	35.5 ± 0.5	45.4 ± 0.4	40.0 ± 1.1	43.8 ± 0.3
	T(s)	36.4 ± 5	36.1 ± 25	68.9 ± 10	20.6 ± 2	13.1 ± 4	175 ± 32
Voronoi OT	Cov%	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0
	Num.S.	22.8 ± 0.0	17.8 ± 0.2	24.0 ± 0.0	15.3 ± 0.0	11.7 ± 0.0	17.7 ± 0.0
	Eff%	54.8 ± 0.0	41.7 ± 0.3	35.7 ± 0.0	44.9 ± 0.0	39.5 ± 0.0	43.5 ± 0.1
	T(s)	33.9 ± 7	41.3 ± 6	49.9 ± 8	19.7 ± 2	10.7 ± 3	156 ± 7

Voronoi OTU	Cov%	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0
	Num.S.	23.0 ± 0.0	17.6 ± 0.7	24.5 ± 0.0	15.2 ± 0.0	11.3 ± 0.0	17.7 ± 0.1
	Eff%	54.6 ± 0.0	41.0 ± 0.4	34.3 ± 0.0	45.5 ± 0.0	40.5 ± 0.0	43.5 ± 0.1
	T(s)	19.1 ± 4	27.9 ± 4	51.5 ± 7	15.4 ± 2	6.8 ± 1	121 ± 15
Voronoi OTD	Cov%	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0
	Num.S.	22.8 ± 0.0	15.5 ± 0.2	23.5 ± 0.0	15.0 ± 0.0	11.5 ± 0.0	17.1 ± 0.0
	Eff%	54.8 ± 0.0	43.3 ± 0.2	36.2 ± 0.0	45.9 ± 0.0	40.2 ± 0.0	44.3 ± 0.0
	T(s)	932 ± 13	1424 ± 268	1734 ± 15	302 ± 30	87.2 ± 13	4480 ± 295
Voronoi RF	Cov%	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0
	Num.S.	22.6 ± 0.2	20.4 ± 0.7	29.0 ± 1.4	16.0 ± 0.4	18.6 ± 1.4	20.7 ± 0.3
	Eff%	50.8 ± 0.2	34.5 ± 0.9	30.2 ± 1.7	43.0 ± 0.7	25.1 ± 1.9	36.8 ± 0.4
	T(s)	137 ± 28	97.1 ± 17	360 ± 58	139 ± 10	64.0 ± 6	797 ± 48

Table 5.2: All algorithms statistics with sensor radius 300. The mean results followed by the standard deviation of five runs are presented in each cell. The results are categorized by the algorithms and the different groups run. The groups are Empty, Multiple Obstacles, Multiple Rooms, One Obstacle, and Tight Corridors. The General column shows the aggregated results of all the groups.

Algorithm	**Metric**	**Empty**	**M. Obst.**	**M. Rooms**	**1 Obst.**	**T. Corr.**	**General**
Random	Cov%	76.8 ± 4.4	55.5 ± 3.6	42.3 ± 7.1	51.9 ± 3.8	51.0 ± 7.7	55.7 ± 2.4
	Num.S.	2.4 ± 0.0	1.6 ± 0.0	2.0 ± 0.0	2.0 ± 0.0	1.7 ± 0.0	1.9 ± 0.0
	Eff%	32.6 ± 2.9	25.6 ± 1.4	19.6 ± 3.0	19.6 ± 1.4	16.7 ± 1.8	22.6 ± 1.0
	T(s)	0.0 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0
ORRD	Cov%	84.3 ± 0.0	73.5 ± 0.0	36.7 ± 0.0	78.9 ± 0.0	28.8 ± 0.0	60.9 ± 0.0
	Num.S.	4.6 ± 0.0	2.8 ± 0.0	2.2 ± 0.0	3.7 ± 0.0	1.0 ± 0.0	2.9 ± 0.0
	Eff%	21.3 ± 0.0	14.3 ± 0.0	15.7 ± 0.0	17.3 ± 0.0	4.6 ± 0.0	14.3 ± 0.0
	T(s)	0.0 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0
ORRD M	Cov%	100 ± 0.0	86.7 ± 0.0	75.9 ± 0.0	92.7 ± 0.0	84.2 ± 0.0	88.4 ± 0.0
	Num.S.	5.8 ± 0.0	3.8 ± 0.0	6.5 ± 0.0	4.7 ± 0.0	3.3 ± 0.0	4.7 ± 0.0
	Eff%	18.6 ± 0.0	15.5 ± 0.0	10.6 ± 0.0	15.6 ± 0.0	12.9 ± 0.0	14.8 ± 0.0
	T(s)	0.0 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0
OAVFA	Cov%	85.3 ± 3.3	64.5 ± 1.1	46.2 ± 5.1	65.6 ± 5.4	61.4 ± 3.7	65.2 ± 1.2
	Num.S.	2.4 ± 0.0	1.6 ± 0.0	2.0 ± 0.0	2.0 ± 0.0	1.7 ± 0.0	1.9 ± 0.0
	Eff%	36.4 ± 1.7	29.2 ± 0.5	21.6 ± 2.0	24.8 ± 2.0	19.0 ± 1.2	26.0 ± 0.4
	T(s)	0.0 ± 0	0.1 ± 0	0.1 ± 0	0.1 ± 0	0.0 ± 0	0.3 ± 0

OAVFA T	Cov%	100 | 0.0	100 | 0.0	100 | 0.0	100 | 0.0	100 | 0.0	100 | 0.0
	Num.S.	5.0 | 0.2	9.1 | 0.6	7.7 | 0.3	5.7 | 0.3	5.3 | 0.3	6.4 | 0.2
	Eff%	24.3 | 0.5	9.9 | 0.4	13.0 | 0.9	13.8 | 0.9	9.8 | 0.5	14.0 | 0.3
	T(s)	2.7 | 1	9.0 | 2	14.4 | 3	6.6 | 0	3.6 | 1	36.4 | 2
OAVFA CT	Cov%	100 | 0.0	100 | 0.0	100 | 0.0	100 | 0.0	100 | 0.0	100 | 0.0
	Num.S.	5.3 | 0.2	9.3 | 0.3	8.3 | 0.2	5.5 | 0.2	5.3 | 0.2	6.6 | 0.1
	Eff%	22.9 | 1.2	10.0 | 0.6	11.7 | 0.3	14.2 | 0.6	9.8 | 0.3	13.7 | 0.2
	T(s)	3.4 | 1	11.2 | 2	24.3 | 3	8.4 | 1	3.7 | 0	51.0 | 3
OAVFA OT	Cov%	100 | 0.0	100 | 0.0	100 | 0.0	100 | 0.0	100 | 0.0	100 | 0.0
	Num.S.	4.4 | 0.1	9.1 | 0.7	8.3 | 0.5	5.6 | 0.3	4.9 | 0.1	6.3 | 0.2
	Eff%	26.8 | 1.3	9.8 | 0.5	11.5 | 0.7	13.9 | 0.7	11.8 | 0.6	14.7 | 0.4
	T(s)	2.6 | 0	8.1 | 2	18.1 | 2	9.0 | 2	2.9 | 0	40.8 | 5
Voronoi	Cov%	99.5 | 0.0	68.0 | 0.0	56.1 | 0.0	43.3 | 0.0	66.1 | 0.0	66.1 | 0.0
	Num.S.	2.6 | 0.0	1.4 | 0.0	2.0 | 0.0	1.2 | 0.0	1.3 | 0.0	1.6 | 0.0
	Eff%	41.7 | 0.0	33.7 | 0.0	24.8 | 0.0	28.4 | 0.0	25.2 | 0.0	30.7 | 0.0
	T(s)	8.3 | 2	21.6 | 4	35.2 | 4	29.8 | 4	22.3 | 1	117 | 6
Voronoi R	Cov%	100.0 | 0.0	76.9 | 1.7	66.5 | 3.0	82.6 | 2.9	76.1 | 8.8	80.9 | 2.5
	Num.S.	3.0 | 0.1	2.0 | 0.1	2.6 | 0.2	2.5 | 0.1	2.0 | 0.1	2.4 | 0.1
	Eff%	37.7 | 1.6	29.8 | 1.1	24.0 | 1.5	26.4 | 0.7	18.9 | 1.6	27.1 | 0.6
	T(s)	10.2 | 3	27.1 | 5	66.7 | 18	38.2 | 5	25.0 | 7	167 | 24
Voronoi T	Cov%	100 | 0.0	100 | 0.0	100 | 0.0	100 | 0.0	100 | 0.0	100 | 0.0
	Num.S.	3.2 | 0.0	12.6 | 0.8	10.2 | 0.1	5.7 | 0.0	5.0 | 0.0	7.1 | 0.2
	Eff%	36.5 | 0.0	6.7 | 0.1	12.8 | 0.0	13.8 | 0.0	13.5 | 0.0	16.5 | 0.0
	T(s)	1.6 | 0	17.8 | 5	21.3 | 2	5.7 | 1	3.6 | 0	50.0 | 7
Voronoi TU	Cov%	100 | 0.0	100 | 0.0	100 | 0.0	100 | 0.0	100 | 0.0	100 | 0.0
	Num.S.	3.2 | 0.0	11.6 | 0.3	10.1 | 0.1	5.5 | 0.0	5.0 | 0.0	6.8 | 0.0
	Eff%	36.5 | 0.0	6.8 | 0.1	12.8 | 0.0	14.2 | 0.0	13.5 | 0.0	16.7 | 0.0
	T(s)	1.4 | 0	11.3 | 1	18.9 | 3	4.5 | 0	2.5 | 0	38.6 | 4
Voronoi M	Cov%	100 | 0.0	100 | 0.0	100 | 0.0	100 | 0.0	100 | 0.0	100 | 0.0
	Num.S.	3.2 | 0.1	12.1 | 0.4	10.2 | 0.0	5.6 | 0.2	5.9 | 0.2	7.2 | 0.1
	Eff%	36.8 | 0.6	6.8 | 0.1	12.8 | 0.0	14.0 | 0.5	10.4 | 1.5	16.0 | 0.5
	T(s)	2.2 | 0	50.4 | 13	225 | 31	33.1 | 11	22.9 | 7	333 | 30
Voronoi OT	Cov%	100 | 0.0	100 | 0.0	100 | 0.0	100 | 0.0	100 | 0.0	100 | 0.0
	Num.S.	3.2 | 0.0	12.4 | 0.5	10.1 | 0.1	5.7 | 0.0	5.0 | 0.0	7.0 | 0.1
	Eff%	36.5 | 0.0	6.8 | 0.1	12.8 | 0.0	13.8 | 0.0	13.5 | 0.0	16.6 | 0.0
	T(s)	1.7 | 0	15.5 | 2	23.1 | 3	5.9 | 1	3.6 | 0	49.8 | 5

Voronoi OTU	Cov%	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0
	Num.S.	3.2 ± 0.0	11.4 ± 0.5	10.2 ± 0.1	5.5 ± 0.0	5.0 ± 0.0	6.8 ± 0.1
	Eff%	36.5 ± 0.0	6.9 ± 0.1	12.8 ± 0.0	14.2 ± 0.0	13.5 ± 0.0	16.7 ± 0.0
	T(s)	1.5 ± 0	12.0 ± 1	17.4 ± 2	5.0 ± 0	2.9 ± 0	38.8 ± 2
Voronoi OTD	Cov%	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0
	Num.S.	3.2 ± 0.0	7.4 ± 0.1	6.5 ± 0.0	5.0 ± 0.0	4.0 ± 0.0	5.1 ± 0.0
	Eff%	36.5 ± 0.0	10.9 ± 0.1	16.0 ± 0.0	15.5 ± 0.0	14.9 ± 0.0	18.6 ± 0.0
	T(s)	16.7 ± 0	384 ± 10	629 ± 11	65.3 ± 1	38.5 ± 0	1133 ± 18
Voronoi RF	Cov%	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0
	Num.S.	3.6 ± 0.5	8.8 ± 0.5	8.2 ± 0.8	5.0 ± 0.3	5.3 ± 0.3	6.0 ± 0.2
	Eff%	34.8 ± 1.9	10.2 ± 0.3	13.2 ± 1.9	15.6 ± 0.9	10.2 ± 0.7	16.6 ± 0.6
	T(s)	10.7 ± 1	34.4 ± 7	85.3 ± 23	47.9 ± 6	27.2 ± 6	205 ± 24

Table 5.3: All algorithms statistics with sensor radius 600. The mean results followed by the standard deviation of five runs are presented in each cell. The results are categorized by the algorithms and the different groups run. The groups are Empty, Multiple Obstacles, Multiple Rooms, One Obstacle, and Tight Corridors. The General column shows the aggregated results of all the groups.

Algorithm	**Metric**	**Empty**	**M. Obst.**	**M. Rooms**	**1 Obst.**	**T. Corr.**	**General**
Random	Cov%	90.6 ± 5.6	52.3 ± 9.8	32.9 ± 5.1	40.2 ± 7.2	34.3 ± 18.2	49.7 ± 5.9
	Num.S.	1.2 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
	Eff%	21.3 ± 3.6	9.3 ± 1.4	7.8 ± 1.3	7.6 ± 1.4	4.5 ± 2.2	9.9 ± 1.3
	T(s)	0.0 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0
ORRD	Cov%	57.5 ± 0.0	68.9 ± 0.0	28.3 ± 0.0	72.3 ± 0.0	21.4 ± 0.0	50.3 ± 0.0
	Num.S.	2.0 ± 0.0	2.2 ± 0.0	1.6 ± 0.0	1.0 ± 0.0	2.0 ± 0.0	1.5 ± 0.0
	Eff%	7.7 ± 0.0	6.0 ± 0.0	3.3 ± 0.0	6.8 ± 0.0	1.3 ± 0.0	5.0 ± 0.0
	T(s)	0.0 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0
ORRD M	Cov%	100 ± 0.0	78.8 ± 0.0	45.1 ± 0.0	89.2 ± 0.0	29.4 ± 0.0	68.7 ± 0.0
	Num.S.	3.0 ± 0.0	2.4 ± 0.0	2.0 ± 0.0	3.0 ± 0.0	1.0 ± 0.0	2.3 ± 0.0
	Eff%	9.8 ± 0.0	4.5 ± 0.0	5.1 ± 0.0	5.6 ± 0.0	1.2 ± 0.0	5.1 ± 0.0
	T(s)	0.0 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0	0.0 ± 0
OAVFA	Cov%	91.1 ± 2.8	61.2 ± 3.7	34.7 ± 4.0	46.8 ± 3.9	52.3 ± 4.8	57.5 ± 1.5
	Num.S.	1.2 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
	Eff%	21.7 ± 2.0	10.7 ± 0.7	8.2 ± 1.0	8.9 ± 0.7	6.7 ± 0.5	11.1 ± 0.5
	T(s)	0.0 ± 0	0.0 ± 0	0.1 ± 0	0.0 ± 0	0.0 ± 0	0.1 ± 0

OAVFA T	Cov%	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0
	Num.S.	2.0 ± 0.2	8.5 ± 0.5	7.0 ± 0.3	4.6 ± 0.1	4.4 ± 0.2	5.2 ± 0.1
	Eff%	13.3 ± 1.4	2.8 ± 0.3	3.6 ± 0.2	4.2 ± 0.2	3.1 ± 0.2	5.4 ± 0.3
	T(s)	1.2 ± 0	11.1 ± 1	17.5 ± 2	5.8 ± 1	3.2 ± 0	38.8 ± 2
OAVFA CT	Cov%	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0
	Num.S.	2.0 ± 0.0	8.4 ± 0.4	7.5 ± 0.4	4.8 ± 0.3	4.2 ± 0.2	5.2 ± 0.1
	Eff%	13.3 ± 0.0	2.7 ± 0.1	3.5 ± 0.2	4.0 ± 0.3	3.3 ± 0.1	5.3 ± 0.0
	T(s)	1.0 ± 0	11.9 ± 2	23.9 ± 7	7.5 ± 2	3.4 ± 0	47.7 ± 7
OAVFA OT	Cov%	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0
	Num.S.	1.9 ± 0.1	8.6 ± 0.7	7.0 ± 0.2	4.7 ± 0.2	4.4 ± 0.5	5.2 ± 0.2
	Eff%	14.7 ± 1.3	2.7 ± 0.2	3.7 ± 0.2	4.1 ± 0.2	3.2 ± 0.4	5.6 ± 0.2
	T(s)	1.0 ± 0	10.2 ± 3	15.7 ± 2	6.0 ± 1	3.3 ± 1	36.3 ± 5
Voronoi	Cov%	100 ± 0.0	62.9 ± 0.0	45.1 ± 0.0	51.4 ± 0.0	59.1 ± 0.0	63.8 ± 0.0
	Num.S.	1.4 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.1 ± 0.0
	Eff%	19.9 ± 0.0	11.0 ± 0.0	10.4 ± 0.0	9.7 ± 0.0	7.6 ± 0.0	11.6 ± 0.0
	T(s)	5.0 ± 1	14.6 ± 2	38.8 ± 10	24.6 ± 2	22.2 ± 4	105 ± 6
Voronoi R	Cov%	100 ± 0.0	63.8 ± 3.1	45.1 ± 0.0	43.5 ± 4.5	59.1 ± 0.0	62.1 ± 1.3
	Num.S.	1.4 ± 0.1	1.1 ± 0.1	1.0 ± 0.0	1.1 ± 0.1	1.0 ± 0.0	1.1 ± 0.0
	Eff%	19.4 ± 1.1	10.8 ± 0.3	10.4 ± 0.0	7.4 ± 0.7	7.6 ± 0.0	10.9 ± 0.4
	T(s)	4.9 ± 1	20.8 ± 7	31.4 ± 6	30.8 ± 7	19.8 ± 4	108 ± 10
Voronoi T	Cov%	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0
	Num.S.	1.4 ± 0.0	10.6 ± 0.2	9.8 ± 0.2	5.5 ± 0.0	4.8 ± 0.0	6.2 ± 0.1
	Eff%	19.9 ± 0.0	2.0 ± 0.0	3.3 ± 0.0	3.5 ± 0.0	3.5 ± 0.0	6.4 ± 0.0
	T(s)	1.0 ± 0	12.6 ± 1	21.0 ± 3	5.7 ± 0	3.5 ± 0	43.8 ± 3
Voronoi TU	Cov%	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0
	Num.S.	1.4 ± 0.0	10.4 ± 0.3	9.9 ± 0.1	5.5 ± 0.0	4.8 ± 0.0	6.2 ± 0.1
	Eff%	19.9 ± 0.0	2.1 ± 0.0	3.3 ± 0.0	3.5 ± 0.0	3.5 ± 0.0	6.4 ± 0.0
	T(s)	0.9 ± 0	9.8 ± 1	19.2 ± 2	4.8 ± 0	2.7 ± 0	37.5 ± 2
Voronoi M	Cov%	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0
	Num.S.	1.4 ± 0.1	11.1 ± 0.5	9.8 ± 0.1	5.7 ± 0.0	5.2 ± 0.0	6.4 ± 0.1
	Eff%	19.4 ± 1.1	1.9 ± 0.0	3.3 ± 0.0	3.4 ± 0.0	3.5 ± 0.0	6.2 ± 0.2
	T(s)	1.1 ± 0	76.1 ± 10	241 ± 18	98.6 ± 14	34.8 ± 4	452 ± 33
Voronoi OT	Cov%	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0
	Num.S.	1.4 ± 0.0	10.4 ± 0.2	10.0 ± 0.0	5.5 ± 0.0	4.8 ± 0.0	6.2 ± 0.0
	Eff%	19.9 ± 0.0	2.1 ± 0.0	3.3 ± 0.0	3.5 ± 0.0	3.5 ± 0.0	6.4 ± 0.0
	T(s)	1.0 ± 0	11.9 ± 1	19.3 ± 1	6.0 ± 0	3.7 ± 0	41.8 ± 2

Voronoi OTU	Cov%	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0
	Num.S.	1.4 ± 0.0	10.5 ± 0.2	9.9 ± 0.1	5.5 ± 0.0	4.8 ± 0.0	6.2 ± 0.0
	Eff%	19.9 ± 0.0	2.1 ± 0.0	3.3 ± 0.0	3.5 ± 0.0	3.5 ± 0.0	6.4 ± 0.0
	T(s)	1.0 ± 0	10.6 ± 1	20.6 ± 3	5.3 ± 0	2.9 ± 0	40.4 ± 3
Voronoi OTD	Cov%	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0
	Num.S.	1.4 ± 0.0	6.9 ± 0.1	6.5 ± 0.1	4.2 ± 0.0	3.2 ± 0.0	4.3 ± 0.0
	Eff%	19.9 ± 0.0	3.2 ± 0.1	4.1 ± 0.0	4.5 ± 0.0	4.6 ± 0.0	7.2 ± 0.0
	T(s)	2.2 ± 0	439 ± 3	648 ± 38	80.9 ± 1	31.0 ± 0	1202 ± 41
Voronoi RF	Cov%	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0	100 ± 0.0
	Num.S.	1.4 ± 0.0	8.2 ± 0.4	7.6 ± 0.3	4.7 ± 0.3	4.0 ± 0.2	5.0 ± 0.1
	Eff%	19.9 ± 0.0	2.8 ± 0.2	3.4 ± 0.2	4.1 ± 0.3	3.9 ± 0.3	6.7 ± 0.1
	T(s)	5.5 ± 1	23.7 ± 4	49.5 ± 9	34.6 ± 5	23.3 ± 3	137 ± 15

Chapter 6

Conclusions and Future Work

This last chapter presents a small summary of what has been done and the results of our work. To finish some possible future work is proposed.

6.1 Conclusions

Even though, to the best of our knowledge, our work is a first of its kind regarding the automatic distribution of fire equipment, from our research, it is possible to see that some of its problems are similar to some very well-known ones. Our search for already implemented solutions consisted of WSN coverage problem solutions where we determined some problem context-related information, such as datasets used, obtained results, benchmark metrics, and limitations. This information allowed us to determine the best solutions to test and implement for our problem. After implementing some of those solutions, we tested some modifications to improve their results in different metrics, which seem to have been successful considering their results.

From our results, we have created a decision tree in Fig. 6.1 that shows the best solution to use in each case, taking into account some variables. The main diversion paths are the presence of obstacles in the scenario, the size of the sensor radius compared to the scenario, and the time to achieve a solution. The recommended algorithms are chosen based on the results of the coverage percentage, the number of sensors used, and the time to find a solution. From the tree, it is possible to observe that the Voronoi algorithms perform better when obstacles exist, varying in the modification chosen depending on the sensor's radius and time limitation. In the empty scenarios, it is possible to see that when sensors with smaller are used, the *OAVFA* modifications show the best results. Still, when the radius is bigger, the *Voronoi* algorithms perform best.

We believe that our work is a good contribution to the WSN topic. Since we have implemented a program that can run multiple algorithms in the same environment, in our opinion, it is a better way to compare solutions than running the different algorithms in their ad-hoc way. Because in this way, it could prevent some incongruent results caused by the different ways of implementing

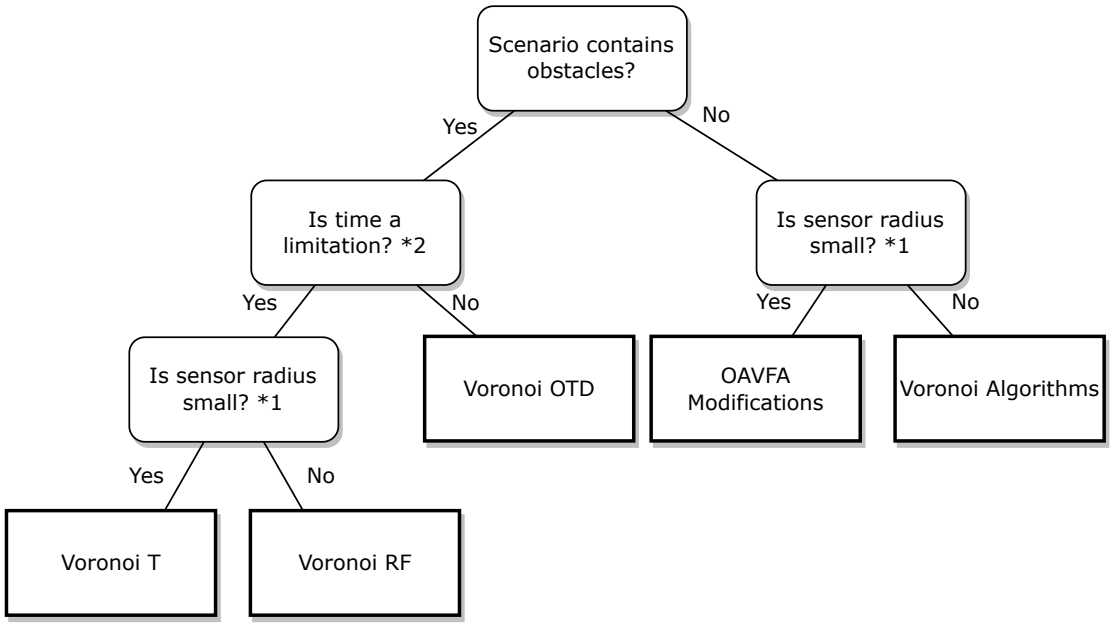


Figure 6.1: Decision tree for best algorithm to use giving conditions

the structural platform. Also, since we tried to implement some previous solutions and got their results in our test cases, it is a good contribution in comparing algorithms that have not previously been compared. Also, our work on creating different modifications to the different researched algorithms was a good contribution because we have improved some of the results that the algorithms showed. The improved performance can be seen in multiple examples such as: in the *ORRD* algorithm from a paper, to its modified algorithm *ORRD M* went from a 68.2% coverage percentage to 96.5% with relatively the same time of processing; in the *OAVFA* algorithm from a publication, to one of its modifications *OAVFA OT* there was an improvement from 86.3% to 100% in terms of coverage but with a significant increase, approximately 70 times, in processing time; in the *Voronoi* algorithm also from a paper to a modification *Voronoi OT*, the results changed from 79.7% to 100% coverage and with an approximately 40% time reduction. Also, since we have focused on the ease of creating new algorithms/test cases/metrics in our program, we believe it could be a good platform for creating new distribution algorithms in the future, making a good contribution to this subject.

From our study, it was also possible to understand that it should be possible to accomplish our primary goal of automating fire equipment distribution in buildings. Making this work a step in the right direction for the Fire Safety project that tries to achieve this goal.

6.2 Future Work

After all our work, we have identified some key topics that could be the subject of future works related to this project. Some of these topics are already planned to be made in the future.

6.2.1 Safe Cities - Fire Planning

Some components in the Safe Cities program will be focusing in the following topics. But a research was made to better understand what are some of the key concepts and features required to create and develop those fragments.

6.2.1.1 BIM and CAD projects

Since our developed work mainly focuses on distribution of the sensors in a 2D environment that has a specific representation which is not normally used in the real world building plans, it makes sense to explore a way to convert the most used formats to ours.

Since most civil engineering projects involve the plans of buildings designed using BIM and CAD files, it would make sense to explore and use them for the conversion.

The use of BIM for fire safety is not something new. In [Wang et al., 2015] the BIM file type is used for evacuation assessment, escape route planning, safety education, and equipment maintenance. Since BIM has a lot of information on the building, like the building structure and details about current components and equipment, it is perfect for planning and maintenance.

Another project that also uses BIM and focuses on the automatic distribution of equipment is [Dahmane et al., 2020]. In this paper, BIM is used to get the building's main structure, which includes walls. Posteriorly, they try to distribute WSN nodes on the whole building to achieve maximum coverage. The process described in this paper is very similar to our work. However, there are some limitations in their work, as they use a discrete environment using a 2D table where each cell represents a one-meter square of the scene and the sensors have a three-meter radius, which can lead to some loss of accuracy. Also, considering that their distribution algorithm is genetically based, although the processing time in the paper is not mentioned, it is expected to be slower than other types of methods like the geometric or force ones.

This topic is a bit touched on in the initial phase of the Safe Cities - Fire Planning project, where a DWG file is used to create a 2D up view image of the floors, and the information relative to the multiple rooms is presented in dimensions related to the pixels of said image. However, this conversion is not detailed enough and not in the same format as our input, so additional work would have to be done on this topic.

6.2.1.2 Sensor Wiring

Another aspect that wasn't much explored in this project is the connection between the sensors.

After having all the required equipment positions, creating "loops" that connect the equipment in disjunctive groups is necessary. Since the main objective is to minimize the quantity of wire

used to connect all the equipment in groups, it would be important to research *Vehicle Routing Problem* VRP algorithms that might be a good problem approximation.

One variation of the Vehicle Routing Problem that fits the problem is the *Capacitated Vehicle Routing Problem* CVRP. The CVRP consists in the optimization of the distribution of goods from a single depot to a given set of customers with known demand using a given number of vehicles of fixed capacity [Augerat et al., 1995]. The CVRP can be translated for our context. A vehicle route represents a wiring "loop," the vehicle's capacity is the number of equipment in a "loop," and the objective is to minimize wiring usage.

The CVRP has been a relevant explored subject of research, which makes it have a lot of resources developed for its study. One of these resources is defined test cases, which make a comparison between multiple solutions easier to access. In Fig. 6.2, it is possible to compare the quality of the results and the time to achieve it from multiple solutions.

Aside from the paper solutions presented in Fig. 6.2, a library implementation also exists that makes it possible to achieve a solution for the CVRP, the Google-OR tools [Perron and Furnon, 2022]. A significant advantage of using this tool is that it would probably be easier and faster to implement it. Also, since this is a tool supported by Google, updates to its algorithm might happen in the future, making it easier to always have the most up-to-date solution for this problem. A disadvantage is that it is impossible to make minor modifications to the solving algorithm.

This topic will also be addressed in the late stages of the Safe Cities - Fire Planning pipeline. The possible connections between the sensors need to be calculated, and an optimal wiring structure must be defined to use as little wiring as possible.

6.2.2 WSN simulators

Another possible future work is exporting the distribution of the WSN sensors and the environment to some of the most used WSN simulators.

Since the WSN simulators are used to test the performance of the network in multiple aspects, like, for example, energy or bandwidth, before the WSN is implemented in the real world, it would make sense to implement a way to export the results from our program to them for future testing.

This future work would make it easier to do the complete process of implementing a WSN from the start.

References	ADFB[a]	CPU time[b]	Computer
Rochat and Taillard (1995)	0.00	N/A	Silicon Graphics 100 MHz
Nagata and Bräysy (2009)	0.00	13.80	Opteron 2.4 GHz
Mester and Bräysy (2007)	0.03	2.80	Pentium IV 2.8 GHz
Marinakis et al. (2010)	0.03	0.80	Pentium M750 1.86 GHz
Taillard (1993)	0.05	N/A	Silicon Graphics 100 MHz
Prins (2009)	0.07	0.27	Pentium IV 2.8 GHz
Pisinger and Ropke (2007)	0.11	17.50	Pentium IV 3 GHz
Chen et al. (2010)	0.13	10.90	Pentium IV 2.93 GHz
Yu et al. (2009)	0.14	2.91	Pentium 1000 MHz
Tarantilis (2005)	0.20	5.63	Pentium 400 MHz
Derigs and Kaiser (2007)	0.21	5.84	Celeron 2.4 GHz
Tarantilis and Kiranoudis (2002)	0.23	5.22	Pentium 2 400 MHz
Ergun et al. (2006)	0.23	28.91	Pentium 733 MHz
Prins (2004)	0.24	5.19	Pentium 1 GHz
Lin et al. (2009)	0.35	8.21	Pentium IV 2.8 GHz
Szeto et al.	**0.46**	4.40	Pentium 1.73 GHz
Reimann et al. (2004)	0.48	3.63	Pentium 900 MHz
Berger and Barkoui (2003)	0.49	21.25	Pentium 400 MHz
Ho and Gendreau (2006)	0.54	4.13	Pentium 2.53 GHz
Rego and Roucairol (1996)	0.55	24.65	4 Sun Sparc IPC
Cordeau et al. (2001)	0.56[c]	24.62	Pentium IV 2 GHz
Baker and Ayechew (2003)	0.56	29.11	Pentium 266 MHz
Toth and Vigo (2003)	0.64	3.84	Pentium 200 MHz
Gendreau et al. (1994)	0.86	46.80	Silicon Graphics 36 MHz
Ai and Kachitvichyanukul (2009)	0.88	2.72	Intel P4 3.4 GHz
Yurtkuran and Emel (2010)	1.04	2.20	Intel Core2 Duo, 2 GHz
Bullnheimer et al. (1999)	1.51	18.44	Pentium 100 MHz
Rego (1998)	1.54	2.32	HP 9000/712
Osman (1993)	2.11	151.35	VAX 8600

[a] Average deviation from best known results.
[b] Average computing time in minutes.
[c] Computational results obtained from Cordeau et al. (2005).

Figure 6.2: Comparison of multiple CVRP solutions (extracted from [Szeto et al., 2011])

Appendix A

Scenarios

In this appendix, we show all the scenarios that we have implemented and used in our work. The scenarios are divided into groups, which contain scenarios that share some common characteristics.

A.1 Empty Group

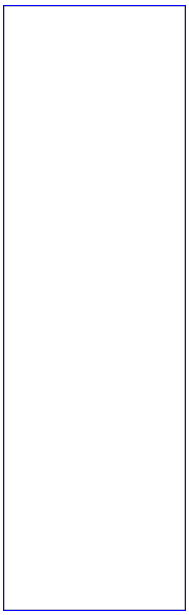


Figure A.1: 150x500 scenario

Figure A.2: 500x150 scenario

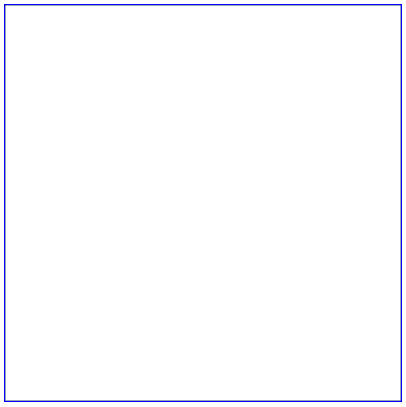


Figure A.3: 300x300 scenario

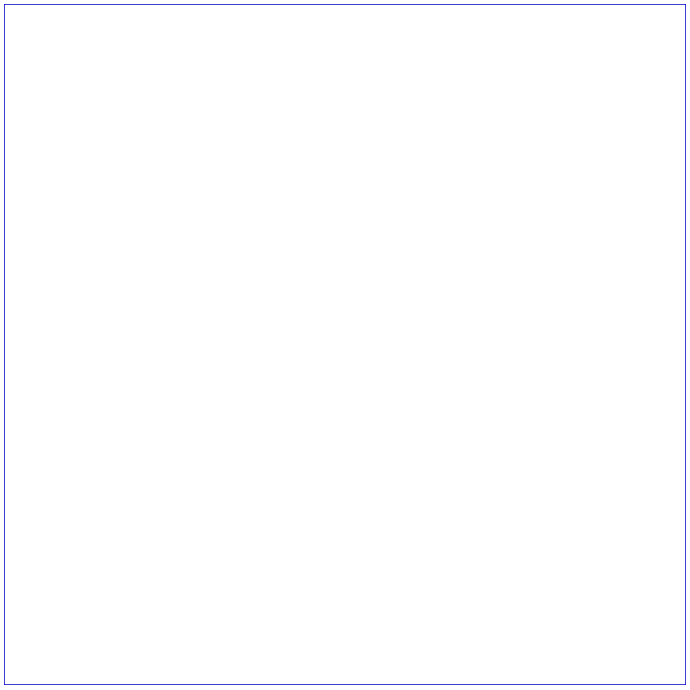


Figure A.4: 900x900 scenario

Figure A.5: 1200x800 scenario

A.2 One Obstacle Group

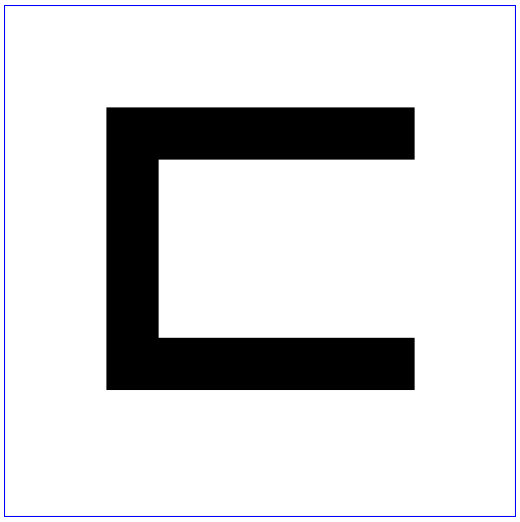


Figure A.6: C scenario

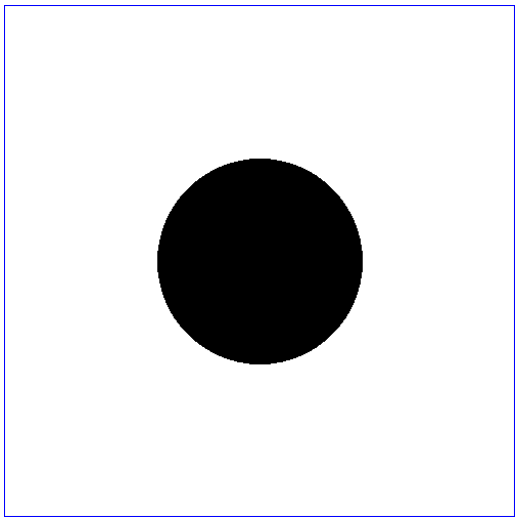


Figure A.7: Circle scenario

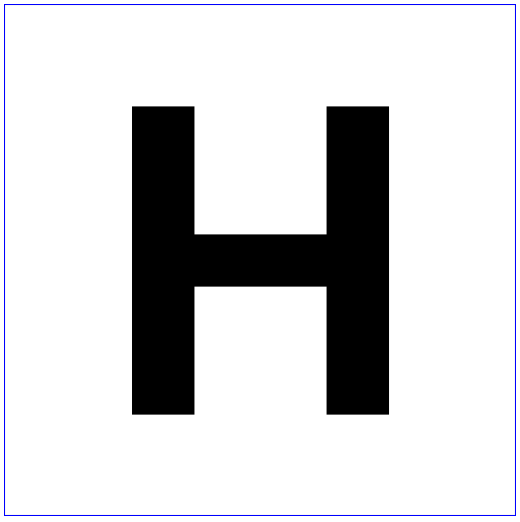


Figure A.8: H scenario

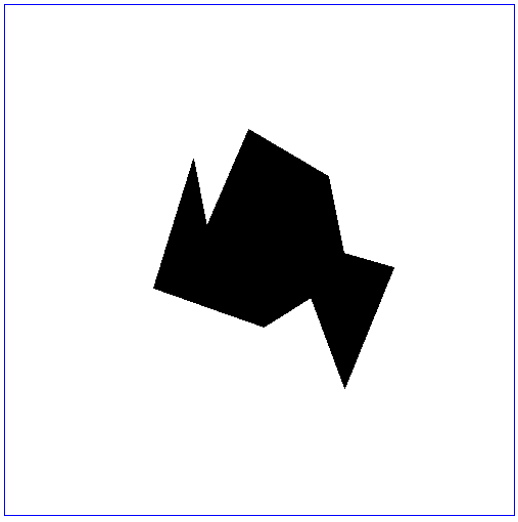


Figure A.9: Irregular scenario

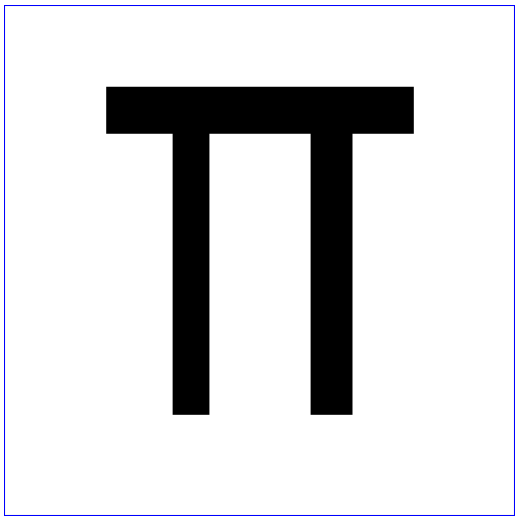


Figure A.10: PI scenario

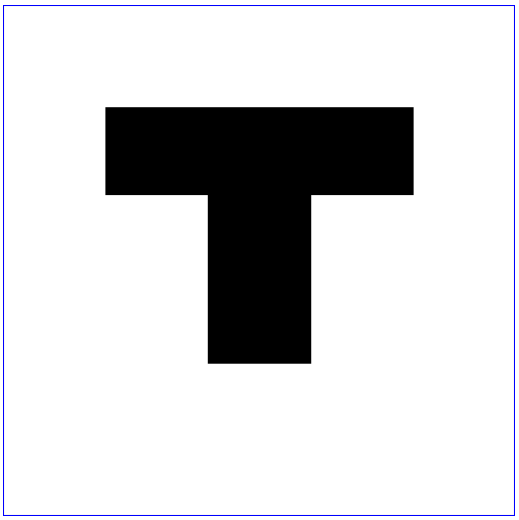


Figure A.11: T scenario

A.3 Multiple Obstacles Group

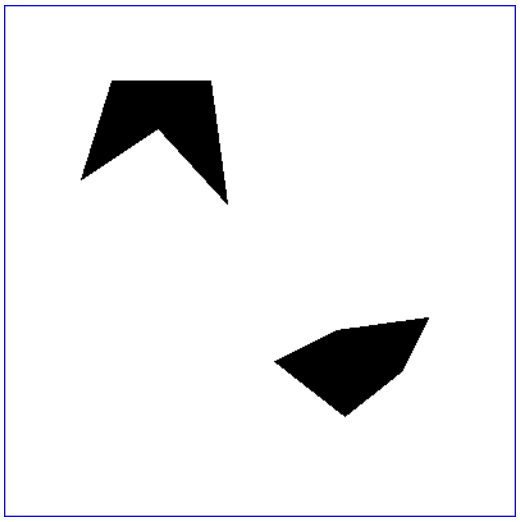


Figure A.12: 2 Irregular 1 scenario

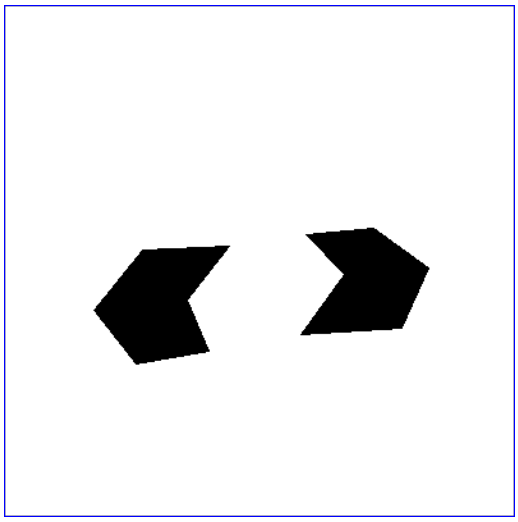


Figure A.13: 2 Irregular 2 scenario

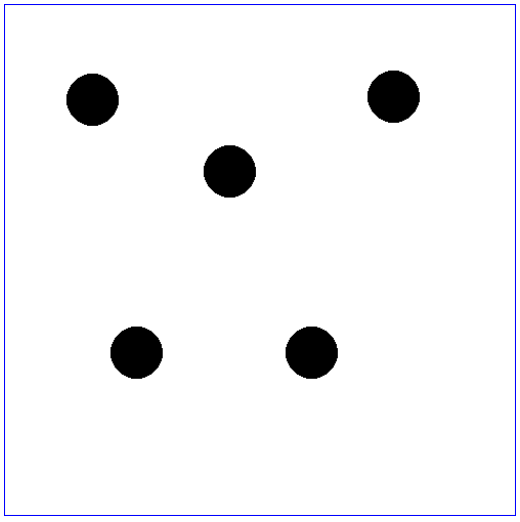


Figure A.14: 5 Circles scenario

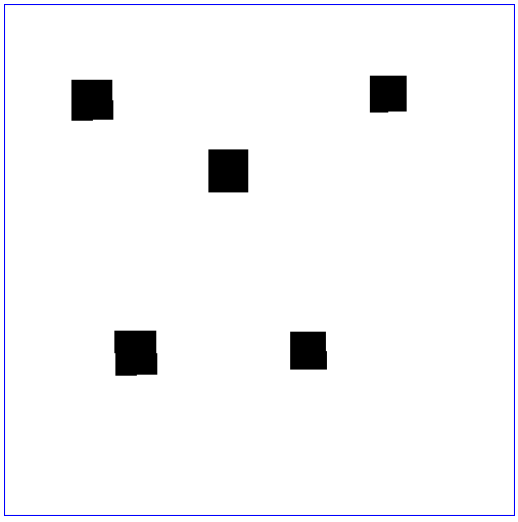


Figure A.15: 5 Squares scenario

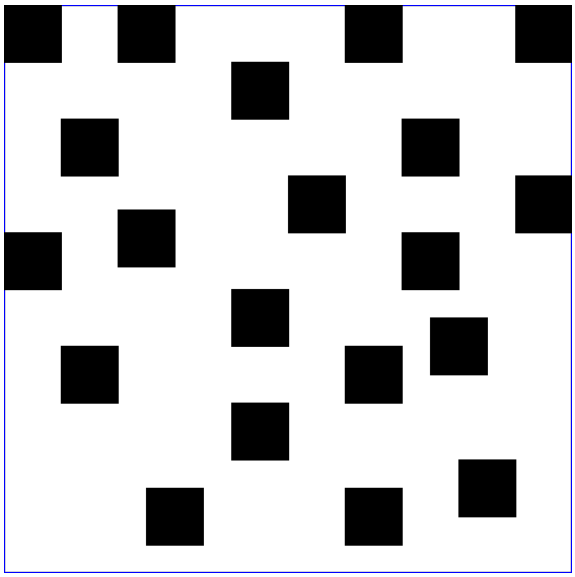


Figure A.16: 20 Squares scenario

A.4 Multiple Rooms Group

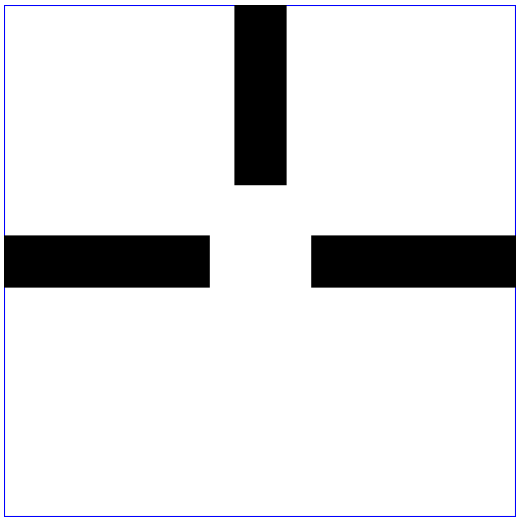


Figure A.17: 3 Regular Rooms scenario

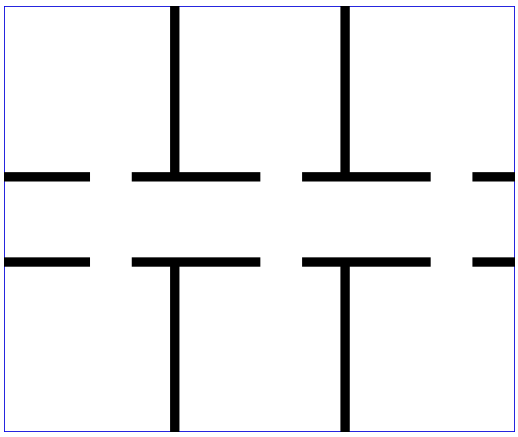


Figure A.18: 6 Rooms 1 Hall scenario

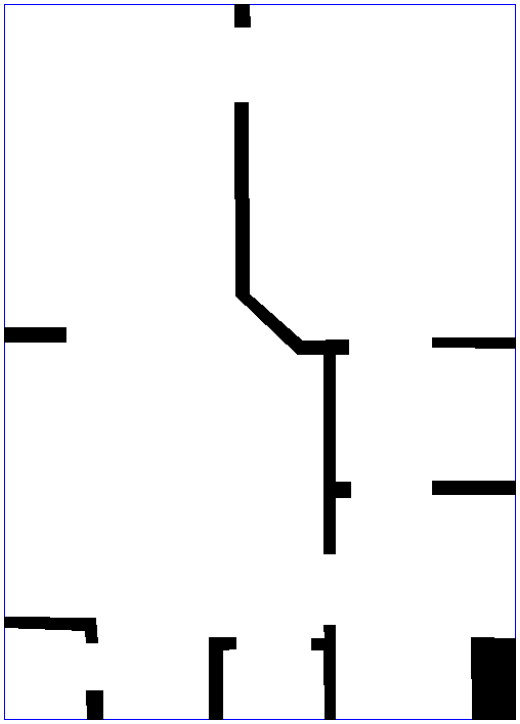


Figure A.19: House Blueprint scenario

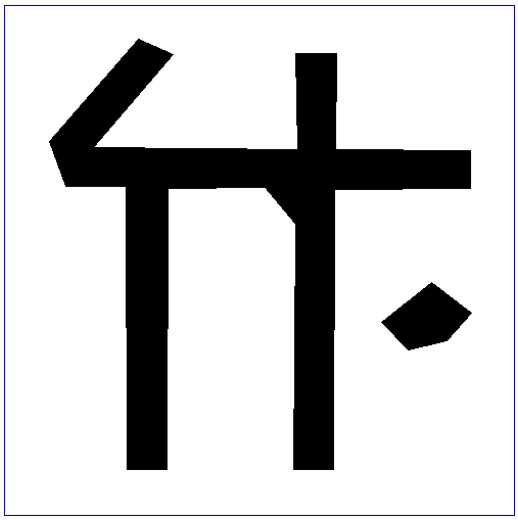


Figure A.20: Irregular Rooms scenario

A.5 Tight Corridors Group

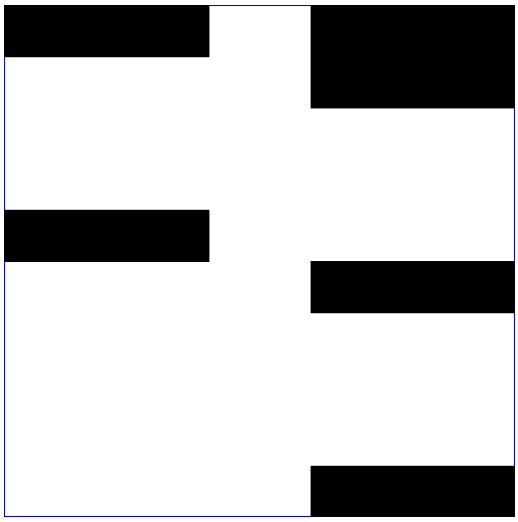


Figure A.21: Width 150 scenario

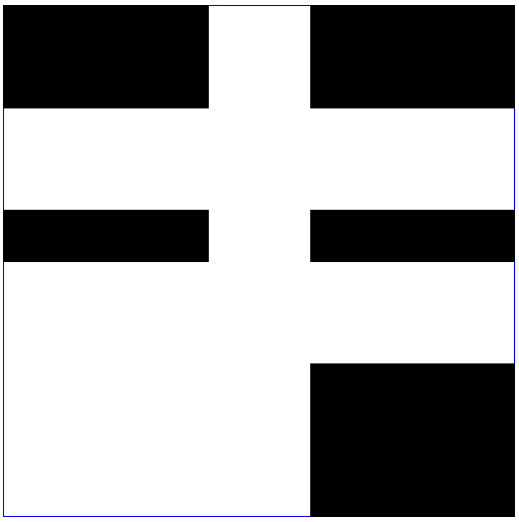


Figure A.22: Width 100 scenario

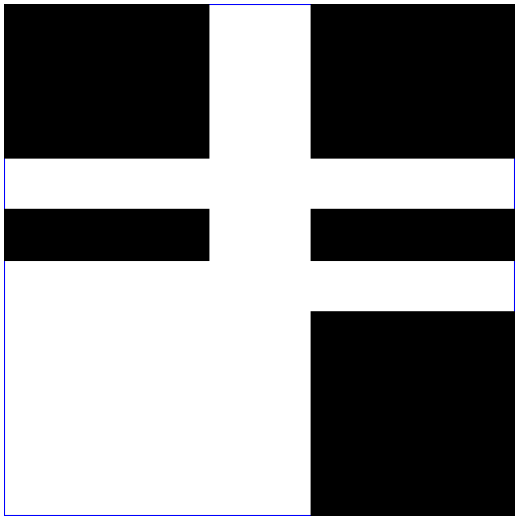


Figure A.23: Width 50 scenario

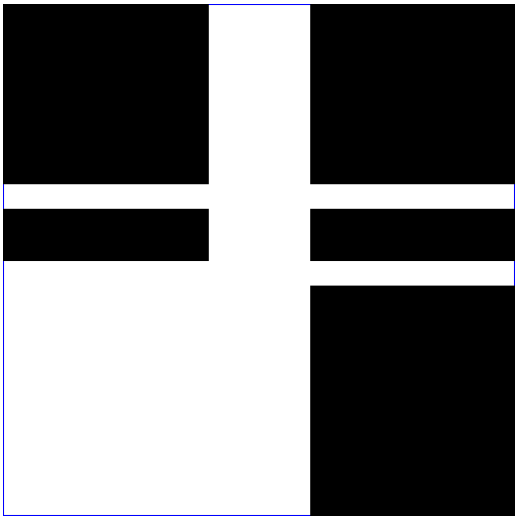


Figure A.24: Width 25 scenario

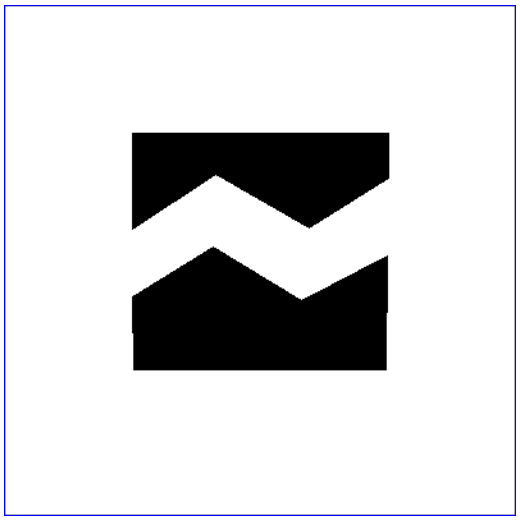


Figure A.25: ZigZag scenario

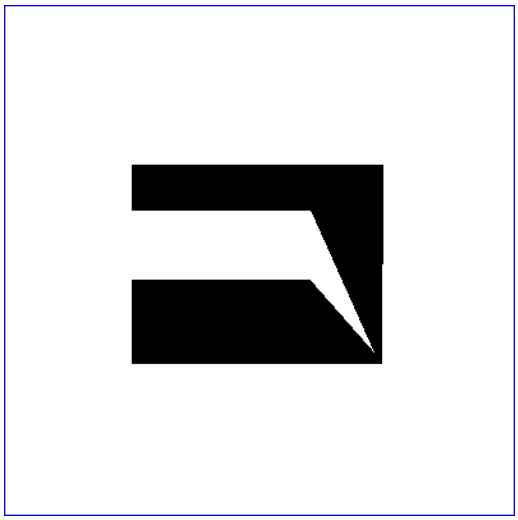


Figure A.26: Decreasing Width scenario

Bibliography

[Acemoglu and Restrepo, 2018] Acemoglu, D. and Restrepo, P. (2018). Artificial intelligence, automation, and work. In *The economics of artificial intelligence: An agenda*, pages 197–236. University of Chicago Press. ISBN: 9780226613338.

[Augerat et al., 1995] Augerat, P., Naddef, D., Belenguer, J., Benavent, E., Corberan, A., and Rinaldi, G. (1995). Computational results with a branch and cut code for the capacitated vehicle routing problem. Technical Report INPG-RR-949-M, Institut National Polytechnique.

[Batalin and Sukhatme, 2004] Batalin, M. A. and Sukhatme, G. S. (2004). Coverage, exploration and deployment by a mobile robot and communication network. *Telecommunication Systems*, 26(2):181–196. DOI: 10.1023/B:TELS.0000029038.31947.d1.

[Batalin and Sukhatme, 2007] Batalin, M. A. and Sukhatme, G. S. (2007). The design and analysis of an efficient local algorithm for coverage and exploration based on sensor network deployment. *IEEE Transactions on Robotics*, 23(4):661–675. DOI: 10.1109/TRO.2007.903809.

[Beeby and Narayanan, 2005] Beeby, A. W. and Narayanan, R. (2005). *Designers' Guide to EN 1992-1-1 and EN 1992-1-2. Eurocode 2: Design of Concrete Structures: General Rules and Rules for Buildings and Structural Fire Design*, volume 17. Thomas Telford. ISBN: 072773105X.

[Binh et al., 2016] Binh, H., Hanh, N., and Dey, N. (2016). Improved cuckoo search and chaotic flower pollination optimization algorithm for maximizing area coverage in wireless sensor networks. *Neural Computing and Applications*, 30(7):1–13. DOI: 10.1007/s00521-016-2823-5.

[Campanile et al., 2020] Campanile, L., Gribaudo, M., Iacono, M., Marulli, F., and Mastroianni, M. (2020). Computer network simulation with ns-3: A systematic literature review. *Electronics*, 9(2):272. DOI: 10.3390/electronics9020272.

[CEN, 2018] CEN (2018). Fire detection and fire alarm systems - part 14: Guidelines for planning, design, installation, commissioning, use and maintenance. Technical Report CENTS_54-14, European Committee for Standardization.

[Chang et al., 2009] Chang, C.-Y., Chang, C.-T., Chen, Y.-C., and Chang, H.-R. (2009). Obstacle-resistant deployment algorithms for wireless sensor networks. *IEEE Transactions on Vehicular Technology*, 58(6):2925–2941. DOI: 10.1109/TVT.2008.2010619.

[Chen et al., 2007] Chen, J., Li, S., and Sun, Y. (2007). Novel deployment schemes for mobile sensor networks. *Sensors*, 7(11):2907–2919. DOI: 10.3390/S7112907.

[Dahmane et al., 2020] Dahmane, W. M., Brahmia, M.-e.-A., Dollinger, J.-F., and Ouchani, S. (2020). A bim-based framework for an optimal wsn deployment in smart building. In *Proceedings of the 11th International Conference on Network of the Future (NoF), 12-14 Oct. 2020, Bordeaux, France*, pages 110–114. IEEE. DOI: 10.1109/NoF50125.2020.9249099.

[Dai and Wang, 2015] Dai, L. and Wang, B. (2015). Sensor placement based on delaunay triangulation for complete confident information coverage in an area with obstacles. In *Proceedings of the 34th IEEE – International Performance Computing and Communications Conference (IPCCC), 14-16 Dec. 2015, Nanjing, China*, pages 1–8. IEEE. DOI: 10.1109/PCCC.2015.7410290.

[Ding and Goshtasby, 2001] Ding, L. and Goshtasby, A. (2001). On the canny edge detector. *Pattern recognition*, 34(3):721–725. DOI: 10.1016/S0031-3203(00)00023-6.

[Downard, 2004] Downard, I. T. (2004). Simulating sensor networks in ns-2. Technical Report NRL/FR/5522-04-10073, Naval Research Lab Washington DC.

[Eledlebi et al., 2020] Eledlebi, K., Ruta, D., Hildmann, H., Saffre, F., Alhammadi, Y., and Isakovic, A. (2020). Coverage and energy analysis of mobile sensor nodes in obstructed noisy indoor environment: A voronoi-approach. *IEEE Transactions on Mobile Computing*, page 17 pages. DOI: 10.1109/TMC.2020.3046184.

[Engmann et al., 2018] Engmann, F., Katsriku, F. A., Abdulai, J.-D., Adu-Manu, K. S., and Banaseka, F. K. (2018). Prolonging the lifetime of wireless sensor networks: a review of current techniques. *Wireless Communications and Mobile Computing*, 2018(2):1–23. DOI: 10.1155/2018/8035065.

[Haight and Kecojevic, 2005] Haight, J. M. and Kecojevic, V. (2005). Automation vs. human intervention: What is the best fit for the best performance? *Process Safety Progress*, 24(1):45–51. DOI: 10.1002/prs.10050.

[Huynh et al., 2019] Huynh, B., Nguyen, H., Quan, L., Nghia, N., and Dey, N. (2019). Metaheuristics for maximization of obstacles constrained area coverage in heterogeneous wireless sensor networks. *Applied Soft Computing*, 86:105939. DOI: 10.1016/j.asoc.2019.105939.

[Illingworth and Kittler, 1988] Illingworth, J. and Kittler, J. (1988). A survey of the hough transform. *Computer vision, graphics, and image processing*, 44(1):87–116. DOI: 10.1016/S0734-189X(88)80033-1.

[Mohamed et al., 2017] Mohamed, S., Hamza, H., and Saroit, I. (2017). Coverage in mobile wireless sensor networks (m-wsn): A survey. *Computer Communications*, 110:133–150. DOI: 10.1016/j.comcom.2017.06.010.

[Mohammad, 2016] Mohammad, S. M. (2016). Continuous integration and automation. *International Journal of Creative Research Thoughts*, 4(3):938–944.

[Nayyar and Singh, 2015] Nayyar, A. and Singh, R. (2015). A comprehensive review of simulation tools for wireless sensor networks (wsns). *Journal of Wireless Networking and Communications*, 5(1):19–47.

[Pediaditakis et al., 2010] Pediaditakis, D., Tselishchev, Y., and Boulis, A. (2010). Performance and scalability evaluation of the castalia wireless sensor network simulator. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques (SIMUTools '10), 15-19 March 2010, Torremolinos Malaga, Spain*, pages 1–6. DOI: 10.4108/ICST.SIMUTOOLS2010.8727.

[Perron and Furnon, 2022] Perron, L. and Furnon, V. (2022). Google or-tools. online: https://developers.google.com/optimization (Last access: 30/06/2022).

[Priyadarshi et al., 2020] Priyadarshi, R., Gupta, B., and Anurag, A. (2020). Deployment techniques in wireless sensor networks: a survey, classification, challenges, and future research issues. *The Journal of Supercomputing*, 76(9):7333–7373. DOI: 10.1007/s11227-020-03166-5.

[Rout and Roy, 2016] Rout, M. and Roy, R. (2016). Dynamic deployment of randomly deployed mobile sensor nodes in the presence of obstacles. *Ad Hoc Networks*, 46(C):12–22. DOI: 10.1016/j.adhoc.2016.03.004.

[Seo et al., 2008] Seo, J.-H., Kim, Y.-H., Ryou, H.-B., Cha, S.-H., and Jo, M. (2008). Optimal sensor deployment for wireless surveillance sensor networks by a hybrid steady-state genetic algorithm. *IEICE Transactions on Communications*, E91-B(11):3534–3543. DOI: 10.1093/ietcom/e91-b.11.3534.

[Singh and Sharma, 2015] Singh, S. and Sharma, R. M. (2015). Some aspects of coverage awareness in wireless sensor networks. *Procedia Computer Science*, 70:160–165. DOI: 10.1016/j.procs.2015.10.065.

[Sobeih et al., 2005] Sobeih, A., Chen, W.-P., Hou, J., Kung, L.-C., Li, N., Lim, H., Tyan, H.-Y., and Zhang, H. (2005). J-sim: a simulation environment for wireless sensor networks. In *Proceedings of the 38th Annual Simulation Symposium (ANSS-38 2005), 4-6 April 2005, San Diego, CA, USA*, pages 175–187. DOI: 10.1109/ANSS.2005.27.

[Spatar et al., 2017] Spatar, E., Fedorova, L., and Kipriyanova, N. (2017). Principles of designing information fire systems in multi-apartment buisdings. *Components of Scientific and Technological Progress*, 2(32):30–33.

[Szeto et al., 2011] Szeto, W., Wu, Y., and Ho, S. C. (2011). An artificial bee colony algorithm for the capacitated vehicle routing problem. *European Journal of Operational Research*, 215(1):126–135. DOI: 10.1016/j.ejor.2011.06.006.

[Varga and Hornig, 2008] Varga, A. and Hornig, R. (2008). An overview of the omnet++ simulation environment. In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops (SIMUTools '08), 3-7 March 2008, Marseille, France*, pages 1–10. DOI: 10.5555/1416222.1416290.

[Wang et al., 2015] Wang, S.-H., Wang, W.-C., Wang, K.-C., and Shih, S.-Y. (2015). Applying building information modeling to support fire safety management. *Automation in Construction*, 59:158–167. DOI: 10.1016/j.autcon.2015.02.001.

[Yoon and Kim, 2013] Yoon, Y. and Kim, Y.-H. (2013). An efficient genetic algorithm for maximum coverage deployment in wireless sensor networks. *IEEE Transactions on Cybernetics*, 43(5):1473–1483. DOI: 10.1109/TCYB.2013.2250955.

[Zhu and Wang, 2014] Zhu, J. and Wang, B. (2014). Sensor placement algorithms for confident information coverage in wireless sensor networks. In *Proceedings of the 23rd International Conference on Computer Communication and Networks (ICCCN), 4-7 Aug. 2014, Shanghai, China*, pages 1–7. IEEE. DOI: 10.1109/ICCCN.2014.6911802.

[Zou et al., 2012] Zou, J., Kusyk, J., Uyar, M. U., Gundry, S., and Sahin, C. S. (2012). Bio-inspired and voronoi-based algorithms for self-positioning autonomous mobile nodes. In *Proceedings of the 31st IEEE Military Communications Conference (MILCOM), 29 Oct. - 1 Nov. 2012, Orlando, FL, USA*, pages 1–6. IEEE. DOI: 10.1109/MILCOM.2012.6415806.