A grid-based maze approach to humanitarian logistics

T Olivier

orcid.org 0000-0001-6273-4438

Dissertation accepted in partial fulfilment of the requirements for the degree *Master of Science in Computer Science* at the North-West University

Supervisor: Prof HA Kruger

Graduation May 2021

24988596

Preface

The completion of this study is only made possible by the strength and determination provided by our Heavenly Father, to whom I give all the honour and glory.

Secondly, I give thanks to my supervisor, Prof Hennie Kruger, for pushing and motivating me to endure through the difficulties of the study and supporting me throughout its lifespan. Thank you for also providing valuable feedback without which this study would not have been possible.

Thank you to my family who continuously supported me and motivated me to pursue my studies further.

I lastly want to thank Dr Isabel Swart for providing language editing for this dissertation.

Abstract

Humanitarian logistics is the planning and implementation of cost-effective and efficient procedures to manage the flow and storage of relief items between an origin, such as a relief or medical station, and the people affected by an event, such as a natural disaster. However, in practice numerous problems and practical difficulties may occur that will prevent the smooth operation of a humanitarian logistics chain. A problem that often occurs is accessibility – following a natural disaster, certain areas may be inaccessible, and it may be difficult or impossible to reach people trapped in these areas. In addition, there is a need to establish sufficient relief facilities in a disaster area to maintain a humanitarian relief supply chain that can provide shelter, medicine, food and other emergency items.

In this study, the use of maze generation and maze-solving techniques together with discrete facility location models that can be used to assist humanitarian logistics in disaster situations is proposed. Different maze generation algorithms are used to develop a maze that represents a real-world disaster-stricken area. The maze is then solved, using different maze-solving algorithms that produce optimal traversable routes. Discrete facility location models are also formulated to determine the extent to which mathematical models can assist with the decision-making process of establishing relief facilities. To implement and demonstrate the proposed techniques and algorithms, a software application is developed that enables users to perform the computations in a fast and efficient manner. The proposed techniques and models are applied in a real-world disaster situation and data obtained from Hurricane Katrina that occurred in 2005 in New Orleans in the United States of America is used. Results obtained from the application of the models and algorithms suggest that the proposed methodology does indeed produce valuable and useful results that are typically required in a humanitarian logistics scenario.

Keywords: Facility location models; Grid-based maze; Humanitarian logistics; Natural disaster; Optimal route.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction and problem description

1.1. Introduction

Natural disasters are a worldwide phenomenon that can cause great destruction and loss of lives. The Merriam Webster Dictionary (Merriam-Webster Inc., 2020) defines a natural disaster as "*a sudden and terrible event in nature (such as a hurricane, tornado or flood) that usually results in serious damage and many deaths*". There are normally five broad categories of natural disaster which include

- Geophysical (earthquakes, landslides, tsunamis, volcanic activity),
- Hydrological (avalanches and floods),
- Climatological (extreme temperatures, droughts, wildfires),
- Meteorological (cyclones, storms, wave surges),
- Biological (disease epidemics, insect/animal plagues).

These categories and others are detailed in the work of Shaluf (2007).

Although the number of people who are killed annually in natural disasters is declining, there is still an average death rate of 60 000 people who are killed each year globally. Historically, droughts and floods were the most fatal disaster events, but the deadliest events today tend to be earthquakes (Ritchie and Roser, 2019). To illustrate the global deaths from natural disasters, Figure 1.1 presents a graphical display of deaths for the period 1900-2016.

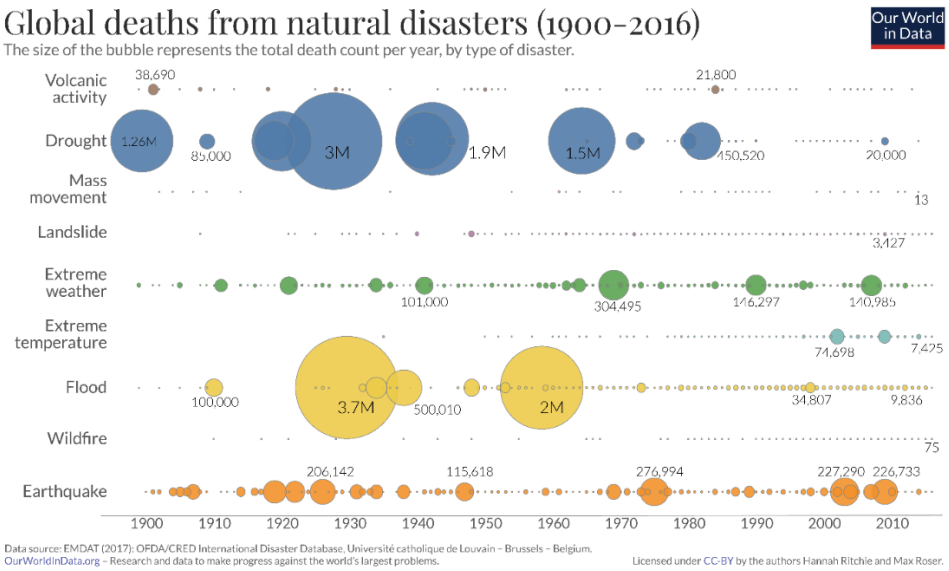


Figure 1.1 Global deaths due to natural disasters for the period 1900-2016

To further highlight the demolishing capabilities of natural disasters, examples of a few recent disasters can be quoted. The recent wildfires that spread across the South Wales region in Australia prompted the Australian Government to declare a state of emergency in November 2019 (Calma, 2020). Thousands of people had to be evacuated and at least 3 000 houses were either completely destroyed or seriously damaged. According to estimates one billion animals also lost their lives in the disaster. Even the smoke became a disaster on its own with smoke pollution reaching New Zealand, 1 000 miles away. Figure 1.2 shows a picture of the fire and smoke.

Figure 1.2 Australia wildfire 2019

Another natural disaster occurred early in 2019 in Japan and cause widespread evacuations. Typhoon Hagibis caused major upheaval in Japan, leaving more than 138 000 households without water and nearly 500 000 without power. According to the Japan Fire and Disaster Management Agency, about 38 000 people had to evacuate their homes due to large scale flooding (Olano, 2019). Also in Japan, a tsunami caused by an offshore earthquake that was classified as a magnitude-9 earthquake hit the east coast of Japan in 2011. More than a million houses were left uninhabitable with about 120 000 completely destroyed and a death toll of approximately 16 000. Some of the damage can be seen in Figure 1.3 which depicts the east coast town, Sukuiso, a week after the earthquake. According to the Japanese government, the total economic cost that was caused by the earthquake reached up to $235 billion (Orskin, 2017).

In the United States of America, Hurricane Sandy was regarded as one of the most devastating natural disasters to hit the USA and parts of the Caribbean during the 2012 Atlantic hurricane season. Sandy raged on for nine days before calming down and caused the death of approximately 220 people (Gibbens, 2019).

Figure 1.3 Sukuiso, Japan – one week after the tsunami

It was estimated by the National Oceanic and Atmospheric Administration that the hurricane caused at least $70 billion in damages. More than 600 000 housing units were destroyed in the New York and New Jersey areas with an estimated cost of $19 billion. In Figure 1.4 part of the damage can be seen.

Figure 1.4 Damage caused by Hurricane Sandy in the USA

Natural disasters and the large-scale destruction caused by them is an important subject of study and many researchers aim to understand the nature of natural disasters while others focus on the support and evacuation of disaster victims, e.g. humanitarian logistics. Examples of such studies can be found in Ahmed *et al.,* (2019), Boustan *et al.,* (2020), Massazza *et al.,* (2019) and Munyaka and Yadavalli (2020).

One of the primary tasks during and immediately after a disaster is to make provision for relief and medical teams to assist victims and to be able to send medical, food and other relief items to specific locations. According to Pradhananga *et al.* (2016), the immediate provision of emergency supplies in the case of large-scale natural disasters, such as tornados, hurricanes and floods is a critical task. This critical task is managed through a humanitarian logistics chain. Duran *et al.* (2013) define humanitarian logistics as the planning and implementation of a process to control flow and storage of goods between their origin and the people affected by the disaster in the most efficient and cost-effective manner. However, in a disaster situation, a myriad of problems and practical difficulties may occur that will inhibit the smooth operation of a humanitarian logistics chain. One such problem is accessibility to certain areas to reach the disaster victims. Bíl *et al.* (2015) describe the enormous impact natural disasters have on roads and pointed out that transportation infrastructure is often demolished which cause residents or other victims to be cut off from the outside world and from any help. Addressing the problem of transporting resources to people in need in areas with no or little infrastructure is the core of humanitarian logistics and often leads to the demand for new and innovative ways of reaching people in need and evacuating them to appropriate and safe locations. In addition to finding optimum traversable routes to reach people, it is also common practice to assist disaster victims by establishing emergency medical or rescue facilities in a disaster-stricken area (Bíl *et al.,* 2015).

In this study, the aim is to make a contribution to the humanitarian logistics problem of finding suitable routes in a disaster-stricken area, as well as to determine appropriate locations for the establishment of relief facilities. The primary focus of the study is the development of a grid-based maze that can be used to represent traversable routes in a disaster-stricken area. Linked to this focus is the formulation of mathematical models that can be used to determine a suitable number and corresponding locations for emergency and relief facilities

A maze can be viewed as a grid that is divided into a collection of paths with an entrance (starting cell) and an exit (end cell). To solve a maze, a path must be found that connects the entrance to the exit. Doing so becomes increasingly more difficult, since many potential paths may exist, most leading to dead ends (incorrect paths). In a disaster area, mazes become more difficult, as damaged areas need to be taken into account when constructing the maze which in turn will have an impact on the possible routes. In this study, a grid-based maze will be constructed in a real-world disaster area. The maze will then be evaluated, using different algorithms to solve the basic humanitarian logistics problem of finding the optimum traversable route between different pre-specified points. Two different solution

options will be considered, namely the Lee algorithm (Lee, 1961) and the A-star algorithm (Hart *et al.*, 1968). In addition, two facility location models will be formulated for the same real-world disaster area. The first model (a set-covering model) will illustrate how a minimum number of relief facilities (covering the entire disaster area) may be determined while the second model (a maximal covering model) will demonstrate the various options of covering a maximum population, given a fixed number of relief facilities to be established. The data from the 2005 hurricane, Katrina, which hit New Orleans in the United States of America, will be used.

This chapter serves as an introduction to the study and the remainder of the chapter is organised as follows. A research question is formulated followed by the research objectives and aims. The methods used to conduct the study will be highlighted and an outline of the chapters will be presented. The chapter is then concluded with a brief summary statement.

1.2. Research question

The focus of the study is to support the humanitarian logistics chain in a disaster-stricken area by formulating and implementing algorithms and mathematical models that can be used in the identification of optimal routes and suitable relief facility locations. A grid-based structure is proposed as an approach to solve the humanitarian logistics problem explained earlier. Consequently, the research question for this study is formulated as follows: To what extent can a grid-based maze approach, linked with a facility location problem, support the activities of a humanitarian logistics chain in a disaster-stricken area?

1.3. Research aims and objectives

The primary objective of this research is encapsulated in the research question and aims to determine to what extent a mathematical modelling approach can support the activities of a humanitarian logistics chain in a disaster-stricken area. It is therefore proposed that a grid-based maze approach, linked with a facility location problem, be investigated as a solution strategy.

To achieve the primary aim of the study, the following secondary objectives must be achieved:

- Conduct a literature review on maze generation algorithms, solving strategies and facility location modelling problems;
- Design and develop a grid-based maze generator to represent traversable routes in a real-world disaster-stricken area;

- Evaluate and apply two different optimum path-finding algorithms to the grid-based maze in the real-world disaster-stricken area; and
- Formulate and implement appropriate facility location models to address relief and rescue station location problems in the real-world scenario.

1.4. Research design

The research project is conducted in a positivistic paradigm and the applicable characteristics of a positivistic approach are taken into account. The study entails an experimental design to create a grid-based maze model that can be utilised in a real-world natural disaster scenario. In addition, specific selected algorithms are evaluated to solve the grid-based maze while mathematical models are formulated and solved to address the problem of locating relief facilities.

The data used include publicly available data on a real-world natural disaster and include data regarding the damage caused by the disaster (to generate and solve the maze), as well as the geographical regions or neighbourhoods in the disaster area and the population of each region (to formulate and solve the relief facility location problem). The real-world natural disaster data used in this study is data obtained from Hurricane Katrina that hit the New Orleans area in the United States of America in 2005.

1.5. Ethical considerations

The research proposal was presented to the ethics committee of the Faculty of Natural and Agricultural Sciences for ethical clearance. The study was approved as a no risk study and the ethics number NWU-01455-20-A9 was issued on 28 May 2020.

There are no humans or other participants involved in the study and the data used in the study is publicly available online.

1.6. Outline of chapters

Apart from this introductory chapter, this dissertation consists of a further six chapters which will be briefly discussed here.

Chapter 2 – Generation of grid-based mazes

The objective of Chapter 2 is to provide background knowledge of maze generation algorithms. Different algorithms are reviewed and insight from the literature on previous implementations of these algorithms is provided.

Chapter 3 – Solution strategies for grid-based mazes

While Chapter 2 presents details on maze generation algorithms, in this chapter, details of algorithms that can be employed to solve a maze are provided. Selected maze-solving algorithms are briefly reviewed and related to the appropriate literature.

Chapter 4 – Discrete facility location models

The focus of Chapter 4 is to present an introduction to the formulation of different discrete facility location models. Aspects such as a taxonomy and classification of the different models as well as model descriptions and formulations are presented. Relevant literature resources on these topics are also presented.

Chapter 5 – Grid-based maze model development: a real-world case study

In Chapter 5, details of a real-world natural disaster case study, based on the techniques and algorithms presented in Chapters 2 and 3 are provided. A software solution that implements the specific algorithms is described and a demonstration of how these algorithms may be used in humanitarian logistics is presented.

Chapter 6 – Relief facility location

A proposed solution strategy for the placement of relief facilities, based on standard mathematical programming models, is presented in Chapter 6.

Chapter 7 – Conclusion

In Chapter 7, the study is concluded and details of how the set goals were achieved will be presented. Limitations of the study, as well as opportunities for further research will also be highlighted.

1.7. Summary

In this chapter, the reader was introduced to the research study. A general description of natural disasters and the accompanying problems were presented. Based on this, a research question was formulated, and specific research aims and objectives were identified. The chapter was concluded with an outline and organisation of the chapters of the dissertation.

Chapter 2 Generation of grid-based mazes

2.1. Introduction

In this study, the feasibility of using a maze as an aid in humanitarian logistics will be investigated. A maze will be developed that can be used in a disaster-stricken area to determine different access routes and the best paths to follow. These paths must allow disaster relief teams to navigate the disaster terrain in the quickest way possible, allowing them to deliver supplies to affected civilians and to transport injured individuals to the closest functioning healthcare facility. Because a wide variety of maze generation algorithms exists, it is important to gain proper background knowledge of the most influential maze generation algorithms. The goal of this chapter is firstly, to supply some background information on maze generation algorithms. Four well-known maze generation algorithms were chosen for this review. Secondly, a short insight from the literature will be given about previous implementations of these maze generation algorithms. Definitions of a maze and associated concepts, such as grid-based structures will first be given. The four selected maze-generation algorithms will then be explained. This will also include a brief literature review of how the algorithms were applied and implemented in different studies.

2.2. Grid-based maze structure

In this section, the structure of a maze that is generated using a grid map will be discussed. Firstly, a grid map will be defined followed by a discussion of the concept of a maze. The section will conclude with a discussion of how these concepts can be used together to produce a grid-based maze; this will be illustrated with data from a disaster-stricken area.

Grid maps

The use of grid maps in scientific study is widely acknowledged as a useful method to represent any environment. In layman's terms, a grid map is a map of any location with a grid placed over it to represent data in the real world. Harabor and Grastien (2011) define grid maps as a graphical representation of an environment that is divided into equal sections where each section represents a measured distance. Grid maps are used in several fields, stretching from leisurely games to scientific research. Grid maps are further used in situations where more needs to be known about a location or where a form of navigation between points on a map is required. This is accomplished by using a grid map to capture data of a certain location in a graphical manner which makes it easier for the reader to read and understand.

A graphical example of a grid map is represented in Figure 2.1 where a section of London, England, was taken and a 6x5 grid placed over it. The grid map is read by determining where the rows and columns intersect on the map. For example, the first cell (top left cell) will be referenced as A1, since row A intersects with column 1. This notation can now be applied to facilitate locating certain points and recording their location. Referring to Figure 2.1, the different locations of landmarks can be identified with greater ease; for example, the Planetarium is located at B3 and Buckingham Palace in D4.

Figure 2. 1 A 6x5 grid map of London, England

Many uses for grid maps exist. While Figure 2.1 represents a basic use of grid maps in pathfinding by determining the location of different landmarks, there are many other implementations as well. Some of these applications will be briefly mentioned in the following paragraphs.

The method of using grid maps as a pathfinding tool is elaborated on by Bekker and Schmid (2006) with their research on ships traveling safely through a minefield. In this study, a grid

map is used to plan paths that avoid minefields in the studied location. The geographical location of each minefield is recorded on a grid map. They used two pathfinding algorithms to determine the shortest and safest path through the minefield. Dijkstra's algorithm is used alongside a generic algorithm to achieve a method where two optimisation algorithms interact with each other to determine an optimal path that considers both the dangers of the sea mines and the surrounding environment. By using the grid map with all this collected data and different pathfinding algorithms, they are able to determine the optimal path through the minefield while still avoiding the detonation of any mine which makes their methods applicable for sea mine avoidance.

Nelson and Smith (2016) discuss the use of grid maps in gaming. A grid map is represented with tiles that are laid down next to each other, both vertically and horizontally, forming a grid structure. Different objects are then placed on the grid structure, such as walls, buildings, NPCs (Non-Player Characters) or any environmental item, such as trees, mountains, rocks or bodies of water to form the desired virtual environment of the game. Any further interaction with the grid will then be determined by the game's mechanics. The mechanics will determine which tiles (cells) are traversable and what actions can be performed within this virtual environment.

Batista e Silva *et al.* (2013) used grid maps to represent the population of Europe over larger geographical areas. Graphically representing large quantities of data on populations is historically difficult; however, advances in technology and scientific methods, such as dasymetric mapping techniques (using symbols to classify large volumes of data according to a location) have increased the accuracy with which the data can be displayed over larger geographical areas. Many countries in Europe originally collected data only spanning their municipal area, but this was inadequate for analyses in many fields. To represent the population data over the entire Europe, they used the population data of 2006 along with a spatial resolution of 100 x 100 meters (divided segments of the graphical map) to validate the data against reference data (gathered through study) to produce a final map of Europe's population.

Fankhauser and Hutter (2016) developed a universal grid map library that can be used as a mapping framework for mobile robotics. The library was designed to provide a wide variety of applications with examples, such as online surface reconstruction and terrain interpretation for rough terrain navigation.

Grid maps can further be used to store arbitrary data about the given location. Konrad *et al.,* (2011) present an approach where grid maps are used to map large geographical areas

while limiting the memory used to process the data, making it real time capable. Each cell of the grid map can store data gathered from arbitrary sensors, such as laser scanner data, calculated grey values from video images and intensities from imaging radar sensors. With this data, a digital road map (like the maps of a GPS) can match a laser scanner grid map.

Examples of other studies that were conducted using grid maps are presented in Table 2.1.

Table 2.1 Additional literature examples for grid maps

Authors	Year	Title
Kim, J.H., Min, K.S. and Yeo, W.Y.	2014	A design of irregular grid map for large-scale Wi-Fi LAN fingerprint positioning systems
Marın-Plaza, P., Beltrán, J., Hussein, A., Musleh, B., Martın, D., de la Escalera, A. and Armingol, J.M.	2016	Stereo vision-based local occupancy grid map for autonomous navigation in ROS
Saeedi, S., Paull, L., Trentini, M. and Li, H.	2015	Occupancy grid map merging for multiple robot simultaneous localisation and mapping
Sturtevant, N.R.	2012	Benchmarks for grid-based pathfinding

From the above-mentioned studies, it is clear that grid maps form an important basis for much scientific research. It does not only allow for data to be represented more accurately but also provides additional functionality that can be applied to research, as indicated in the above studies. One advantage of a grid is that it may be used to generate a maze. In the subsequent paragraphs the concept of a maze and its use will be briefly introduced.

Mazes

A maze, also referred to as a labyrinth, is a puzzle that is designed with the intension that it should be solved. An average maze usually consists of an area that is divided into a collection of paths with an entrance (starting point) and an exit (goal point). The typical path that must be found to solve the maze is the one that connects the entrance to the exit. Usually, there are also many other potential paths which could possibly lead to a dead end (not the correct path to exit). A formal definition of a maze is given by Lee *et al.* (2010) as a grid consisting of cells which include the starting and ending cells, as well as walls that ultimately form the traversable paths and dead ends within the maze. Graphical examples of mazes are represented in Figure 2.2 and Figure 2.3. The first maze in Figure 2.2 is a representation of a simple maze (image drawn by hand or computer) with and entry point (start) and an exit point (finish). The second maze in Figure 2.3 is an example of a real-world garden hedge maze that is created by growing and cutting hedges in different shapes and sizes to form the walls of a maze.

Mazes have been widely used in science and technology for many years. One of the oldest recollections of a maze is in ancient Greek mythology (The Knossos Labyrinth) that dates

back over a thousand years (Castleden, 2012). Pershin and Di Ventra (2011) use mazes as a means of testing the capability of their memristor (resistors with memory) network. This network of memristors showed the capability of not only completing each maze, but also of sorting the paths according to their lengths.

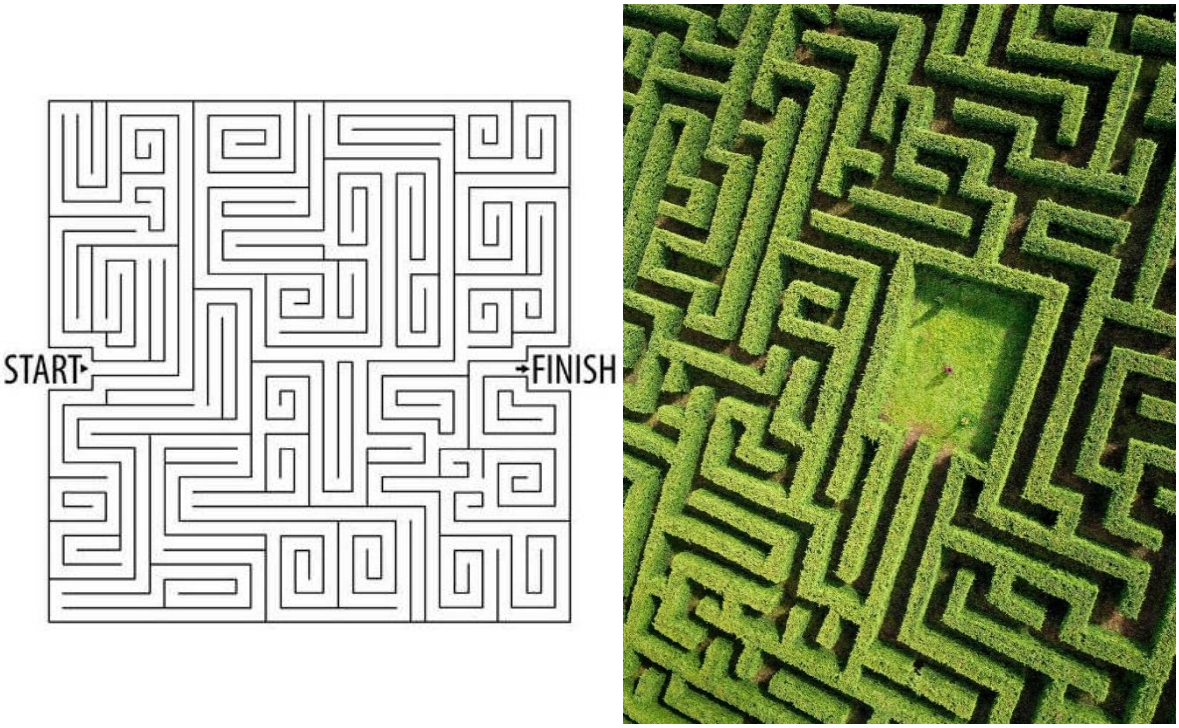


Figure 2.3 Real-world hedge maze

Real-world mazes are popular tools in the field of science for conducting studies on the behaviour of different organisms. Adamatzky (2012) discusses the use of such a maze. A Plasmodium of Physarum Polycephalum (a large cell visible to the naked eye) is placed in a maze with a source to which it is attracted. The study found that the Plasmodium cell formed a network of protoplasmic tubes that connects the cell to the source of attraction. The tubes that are formed represent the path in a maze from a start cell (Plasmodium cell) to the end cell (source of attraction).

Different maze forms can be observed in different real-world structures. Wang *et al.* (2011) noticed this occurrence in the structure of bread. Through testing different bread samples, they determined that approximately 99% of a bread's total porosity is influenced by a single, vastly interconnected, open cell found in a loaf of bread. They concluded that the sponge structures of both bread and cakes exhibit a maze-like structure where all the open cells in the structure are connected.

Diamond *et al.* (1999) discuss the extension of using a modified version of the Morris water maze to test the effects that predator exposure has on the working memory of rats. A Morris

water maze contains four or six paths that originate from a central area. At the end of one of these paths is a hidden platform (goal state) which the rat should identify in order to follow the correct path. The study was conducted by allowing several rat subjects to solve the maze and gain memory of the correct path. After a brief waiting period the rats are placed back into the maze to test if their memory is maintained and if this makes it easier for them to solve the maze.

As in the above-mentioned studies, the versatility of mazes allows researchers to shape the mazes as it would best fit their research. This advantage of mazes allows for a large number of studies to be conducted. Some of these studies are briefly mentioned in Table 2.2.

Table 2.2 Additional literature examples for a maze

Authors	Year	Title
Ashlock, D., Lee, C. and McGuinness, C.	2011	Search-based procedural generation of maze-like levels
Brown, M.F.	1992	Does a cognitive map guide choices in the radial-arm maze?
Derdikman, D., Whitlock, J.R., Tsao, A., Fyhn, M., Hafting, T., Moser, M.B. and Moser, E.I.	2009	Fragmentation of grid cell maps in a multicompartment environment
Lerch, J.P., Yiu, A.P., Martinez-Canabal, A., Pekar, T., Bohbot, V.D., Frankland, P.W., Henkelman, R.M., Josselyn, S.A. and Sled, J.G.	2011	Maze training in mice induces MRI-detectable brain shape changes specific to the type of learning.
Soukup, J.	1992	Maze router without a grid map

It is clear that mazes are not only used as a puzzle to find an optimal path, but also to provide a means of testing an individual's cognitive abilities. Many more uses for a maze exists to conduct research or assist in real-world scenarios. One such use is to incorporate a grid map within a maze to form a grid-based maze structure. Next, the uses of a grid-based maze structure will be discussed to determine how it can assist in a real-world disaster situation.

Grid-based maze structures

Mazes and grid maps both play an essential role in their separate fields of study and entertainment but by using a grid map along with maze generation techniques (to be discussed in Section 2.3), it can be used to form a grid-based maze structure. Such a structure allows for a maze to be generated, using the grid as a basis where the edges of each cell can represent a possible wall in the maze. A graphical example of a maze generated using a grid-based structure is presented in Figure 2.4. On the left side of Figure 2.4 is an empty 6x6 grid from which a maze can be generated. On the right side of Figure 2.4 is the same 6x6 grid with a maze, generated from a maze generation algorithm, fitted on

the grid. The bold black lines represent the walls of the maze that prevent movement through them.

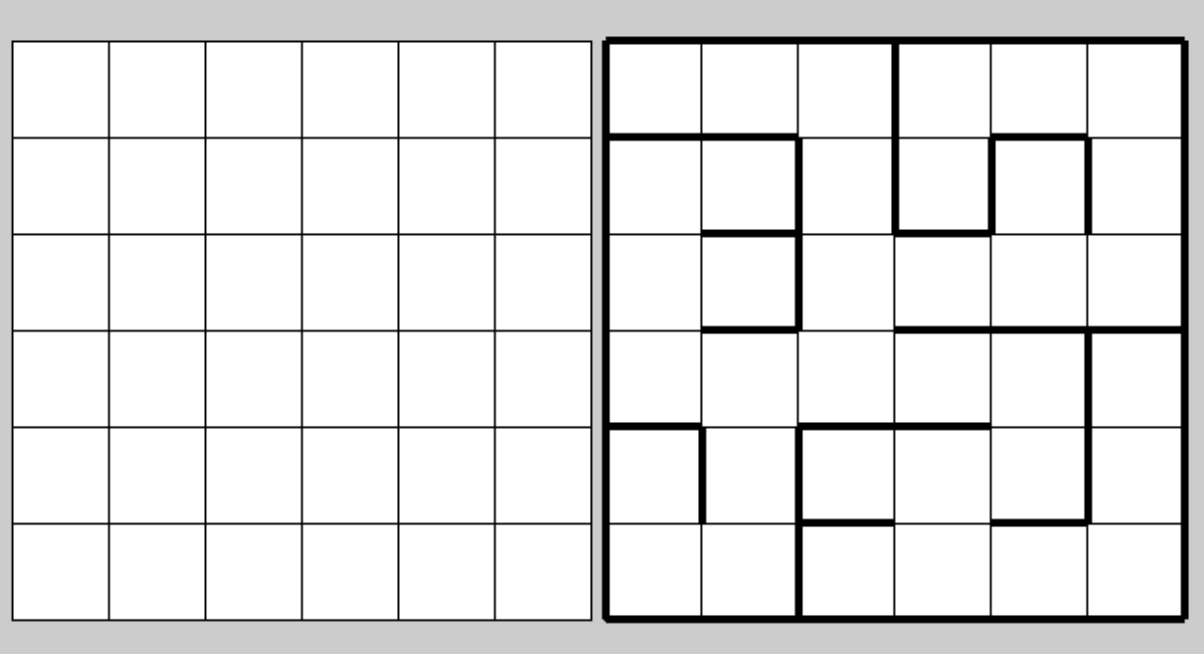


Figure 2.4 A simple grid-based maze

Kim (2019) describes a grid-based maze structure as follows: a computer-generated maze typically makes use of a two-dimensional array with each array value representing a cell (node) in the maze. The traversal between cells (nodes) can only occur in four possible ways. This includes both horizontal and vertical transitions as in Figure 2.5, but no diagonal movements as in Figure 2.6 are allowed.

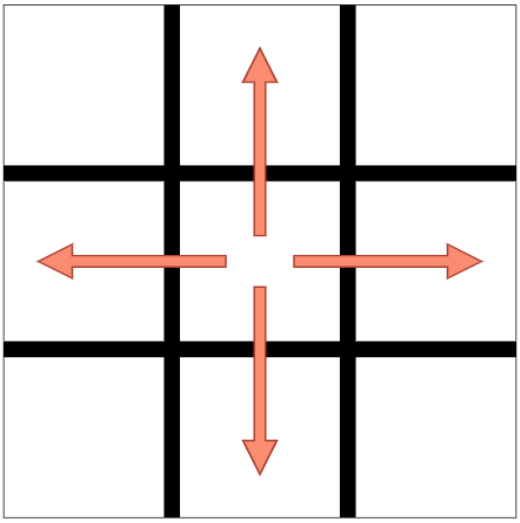


Figure 2.5 Allowed horizontal and vertical movements

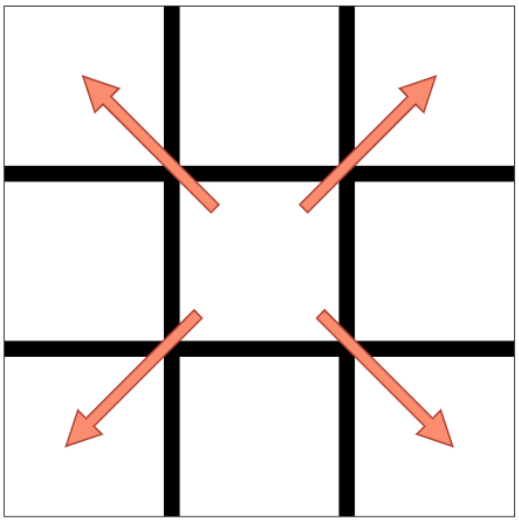


Figure 2.6 Prohibited diagonal movements

The structure of an average maze is a spanning tree of the possible paths with a cell represented as a node in the tree. The root node is represented as the starting cell and expands until a goal cell (leaf node) is reached. A graphical example of a spanning tree representing paths in a maze is illustrated in Figure 2.7.

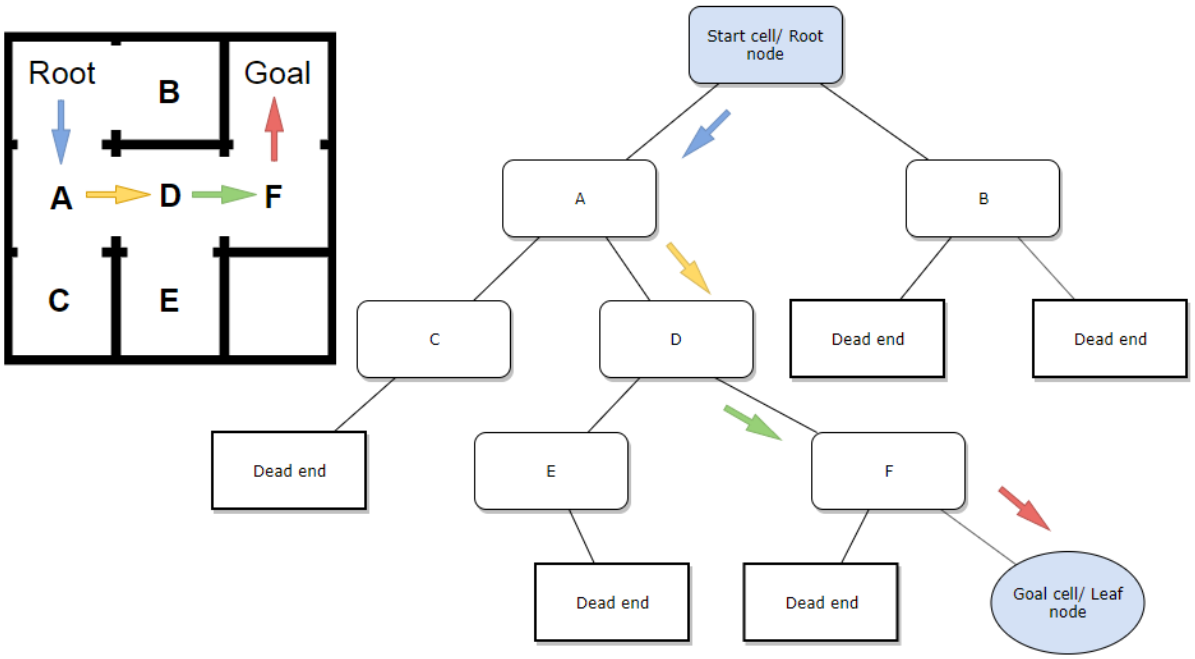


Figure 2.7 Spanning tree structure of maze

By using a grid map as a basis for a grid-based maze structure allows for real-world environments to be graphically represented in a maze. Any real-world obstacles can easily be represented as a wall in the maze, redirecting an optimal path as it would in a real-world scenario (Jubair & Hawa, 2020). This method can be especially useful in the planning for humanitarian logistics. The wake of destruction left behind by natural disasters can be used to form a grid-based maze using a grid map of the disaster-stricken area. Each wall in the maze can be a graphical representation of inaccessible paths in the real-world disaster-stricken area. By finding an optimal path in the maze, a similar optimal path is found in the real-world scenario which allows humanitarian logistics to be applied in a more efficient manner by predetermining the optimal path to traverse to certain areas.

To illustrate the practical use and advantages of a grid-based maze structure in a real-world scenario, consider the devastation of Hurricane Florence that made landfall in North Carolina in September 2018.

Most of the destruction caused by Hurricane Florence was due to flooding. During the life span of the hurricane, 913 mm of rain was recorded in Elizabethtown, North Carolina (Pender County, 2019). Even though Hurricane Florence made landfall as a category 1

hurricane, it was still able to uproot trees and cause major power outages due to its wind speeds. At least 57 deaths were recorded and property damage estimated at $24 billion was caused.

Figure 2.8 shows the damage caused in the town of Cartersville in North Carolina.

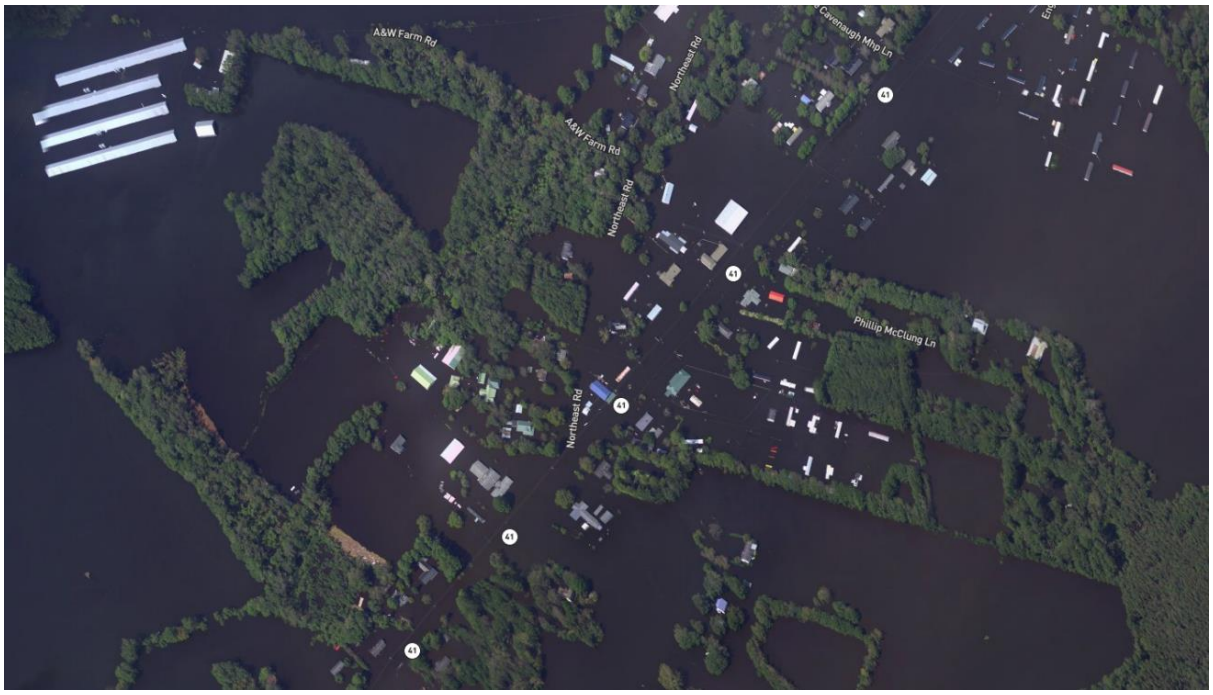


Figure 2.8 Flooding caused by Hurricane Florence in Cartersville, North Carolina

It is clear from Figure 2.8 that the scale of damage to property and the flooding of the surrounding environment, restricts the movement of civilians caught up in the tragedy, as well as that of any possible aid. Figure 2.9 illustrates how a grid-based maze can be used to navigate the terrain.

Initially, a grid is placed over the entire image. All cells that cannot be reached via any form of transportation, such as a boat or car are then blacked out, indicating inaccessible cells. By connecting all these cells, a maze structure is formed with all the inaccessible cells indicating walls in the maze and the open cells indicating possible paths through the maze. The red line in Figure 2.9 provides an example of a route through the maze by means of a boat or any waterborne vehicle to reach the affected individuals. It is assumed that air transport is not available. In the case of air transport being used, optimal routes and the location of relief facilities would not pose any problems. Different maze traversal techniques can now be implemented to generate valid paths that can be followed to solve the maze and also to generate an optimal route to reach a pre-specified destination. For illustration purposes, the maze is also placed on a Google satellite image of the same location (Cartersville, North Carolina) and is shown in Figure 2.10.

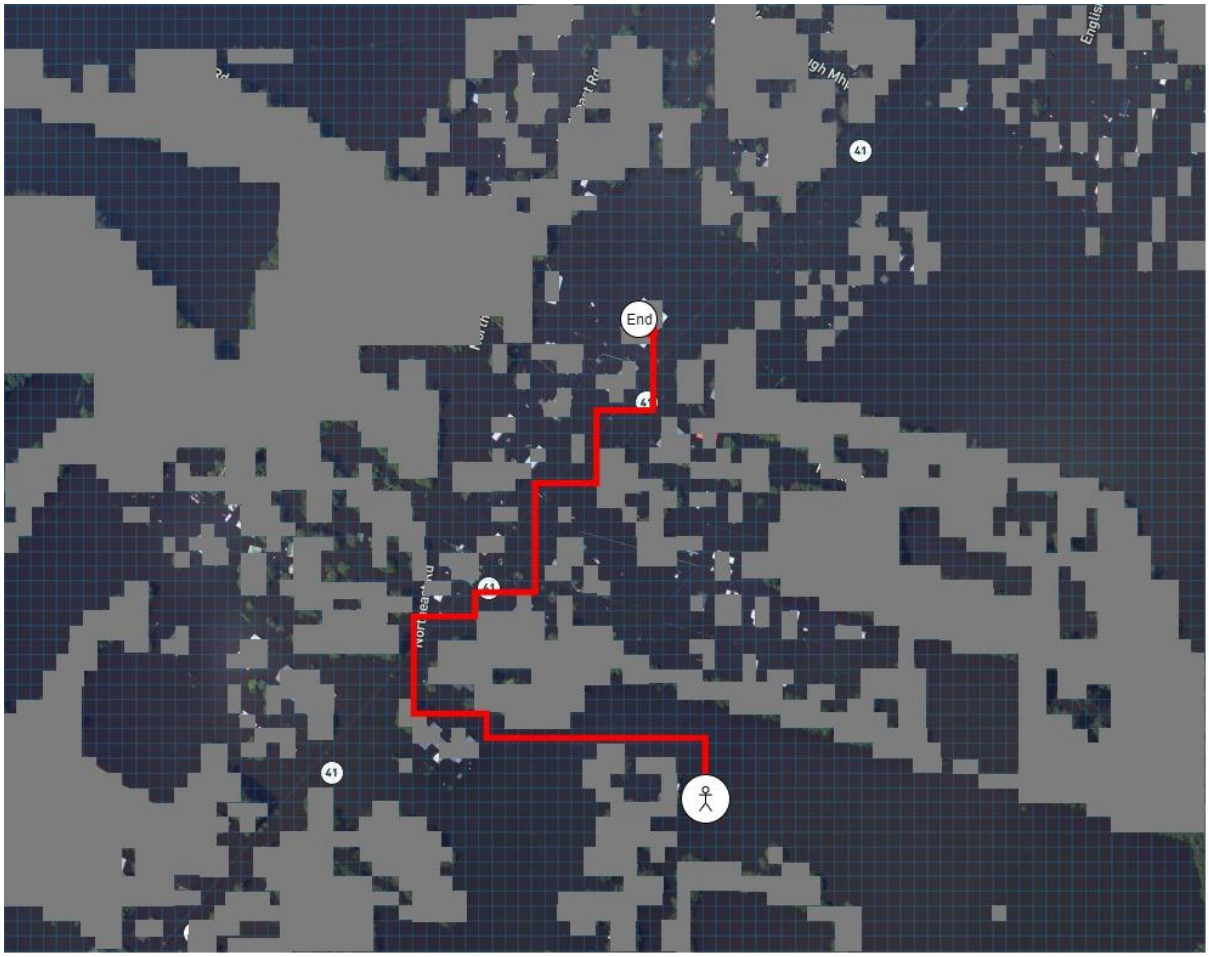


Figure 2.9 Cartersville, North Carolina with grid-based maze overlay

Figures 2.9 and 2.10 represent an important use of a grid-based maze in scientific research. Other examples of similar research will be briefly mentioned in the following paragraphs.

Jannu and Jana (2016) discuss the use of a grid-based structure in the planning and implementation of a clustering and routing algorithm to solve a hot spot problem in wireless sensor networks. Nodes situated closest to an origin are often overburdened with large loads of traffic which results in quick energy exhaustion. By designing the network in a grid-based structure, the use of different pathfinding algorithms to determine optimal paths for data to flow through the network can be implemented.

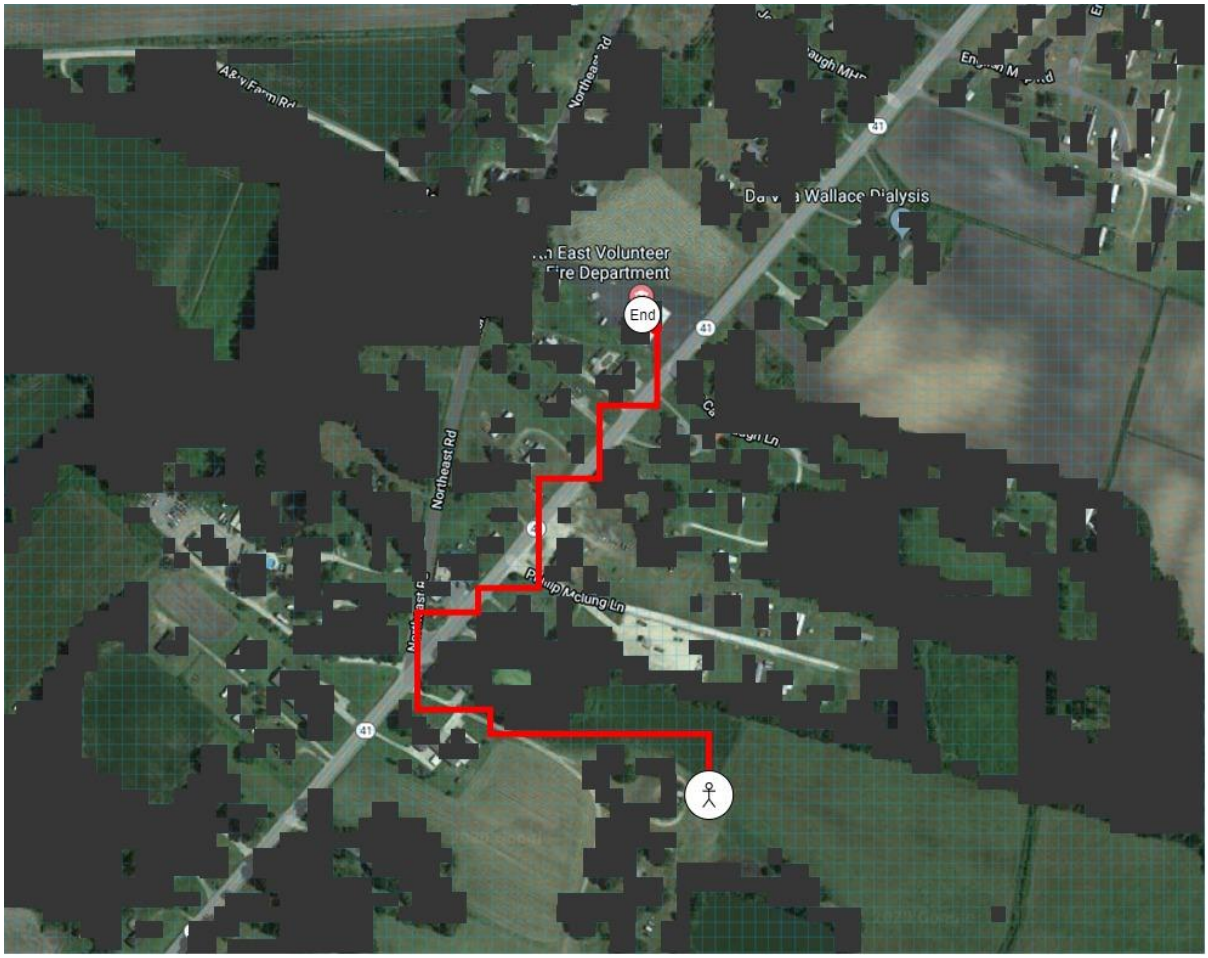


Figure 2.10 Cartersville, North Carolina with grid-based maze overlay - Google Maps satellite image

In the world of gaming, finding the shortest path from point A to B is no easy task. Barnouti *et al.* (2016) propose one way of simplifying this task. By generating a maze according to the structure of the in-game environment, maze routing algorithms can be applied to easily find the shortest path. In this scenario, the maze paths represent the possible routes that can be traversed within the game and the walls represent all the inaccessible areas (mountains, buildings, rivers, etc.). To accomplish this task, a grid is placed over a selected environment whereafter each cell that contains inaccessible areas is marked as wall cells until only the traversable cells remain. The A-star search algorithm was then applied to the generated maze to find the shortest path.

The A-star search algorithm in the field of maze routing is very popular and is viewed by many researchers as the go-to option since it produces results in fast search times. Zhang *et al.* (2016) propose a new and improved version of the A-star algorithm to be used in grid-based maps. They propose a Rectangle Expansion A-star (REA*) algorithm which explores maps in units of unblocked rectangles. The proposed algorithm uses the bounds of each rectangle (edges in grid) as the nodes to explore, instead of points within those bounds

(inside cell area). This allows for the algorithm to plot fewer points on the grid map and have a shorter open list than the normal A-star algorithm which in turn shortens the search time.

Additional examples of research studies where grid-based structures combined with mazes were used can be found in Gordon and Matley (2004), Nelson and Smith (2016) and Sturtevant (2012).

In this section, introductory information on grid-based maze structures was presented. Discussions on grid maps and mazes were introduced separately and then combined to show how grid-based maze structures can be constructed and applied. The discussion was substantiated by appropriate literature resources. In the next section, some of the most well-known maze generation algorithms that are used to design a maze will be introduced.

2.3. Maze generation algorithms

Generating a basic maze can happen in one of two ways: algorithmic or non-algorithmic. Foltin (2011) describes algorithmic maze generation algorithms as a predefined order of steps which is followed to create a maze. Non-algorithmic mazes which follow no predefined order of steps, such as a fractal maze (a maze created by combining mazes) will not be discussed further in this chapter, since they hold no value for this study. In this section, the focus will rather be on graph-based maze generation algorithms which create a maze based on a minimum spanning tree that corresponds to the maze. A minimal spanning tree is a group of edges of a connected, weighted and undirected graph that connects all the vertices together and eliminates the possibility of any cycles occurring and yielding the minimum total edge weight to explore the graph (Grygorash *et al.,* 2006). A spanning tree can be created by either continuously expanding the maze from a single randomly selected position (e.g. Prim's algorithm – see Section 2.3.1) or by joining smaller fragments of tree structures to build the maze (e.g. Kruskal's algorithm – see Section 2.3.2). Two basic methods exist to generate a maze, i.e. wall adding and passage carving. A wall adding algorithm focuses on the positioning of the walls (algorithm adds walls to an empty canvas) while passage carvers focus on cell positioning and relationships between the cells (algorithm removes cell edges from a grid).

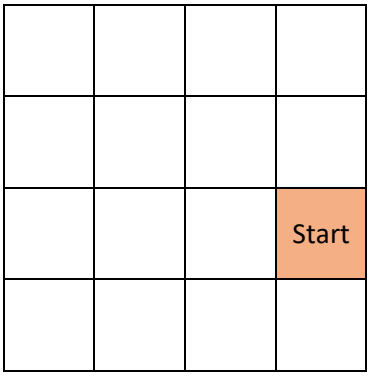
A number of different techniques and algorithms exists that can be used to generate a maze, such as algorithms based on greedy principles (Foltin, 2011), spanning-tree approaches (Kim, 2019) and graph theory (Bollobás, 2013). Four of the most well-known maze generation algorithms will be discussed to gain appropriate knowledge for the completion of this study. The choice of algorithm is based on the popularity of the algorithm and the variety

of fields to which the algorithm can be applied. The chosen algorithms are also popular for solving different problems, for example, a minimum spanning tree problem (Neumann & Witt, 2010). The four algorithms that will be discussed in this section are Prim's algorithm, Kruskal's algorithm, recursive backtracking algorithm and the hunt-and-kill algorithm. Additionally, a matrix maze (Aki & Güllü, 2016) will also be discussed in a later chapter, since it has significance for this study, but has no predefined algorithm to follow, as it is designed, based on the preferences of the user.

2.3.1. Prim's algorithm

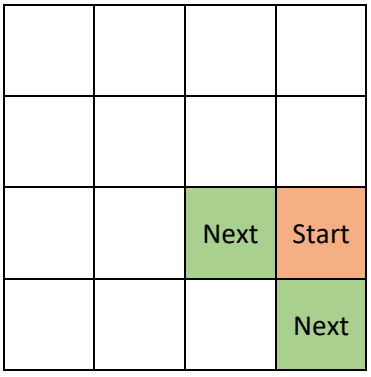
Prim's algorithm is a greedy algorithm that was initially developed to solve the minimum spanning tree problem in graph theory. A European mathematician, Vojtěch Jarník (Jarník, 1930), was the first to develop the algorithm and it is sometimes called Jarník's algorithm. It was only later that the algorithm was rediscovered and published in 1957 by computer scientist, Robert C. Prim (Prim, 1957).

The working of the algorithm is briefly illustrated with a graphical example as described by Jeong and Kim (2016). The main assumption in the algorithm is that every cell in a grid can be a wall. Based on this assumption, the following six steps illustrate the generation of a maze.



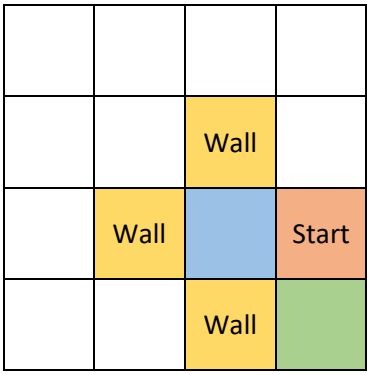
Step 1 - Select a random starting cell.

A path is systematically generated from a selected starting cell and walls are created along the path.



Step 2 – Random cell selection

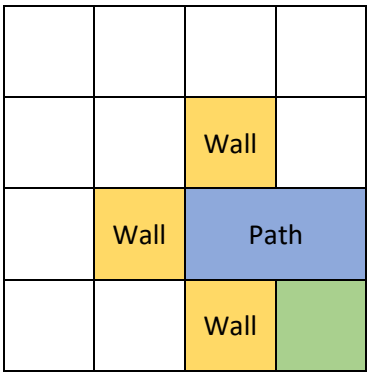
Randomly select a pair of neighbouring cells that will be explored next.



Step 3 – Wall placement

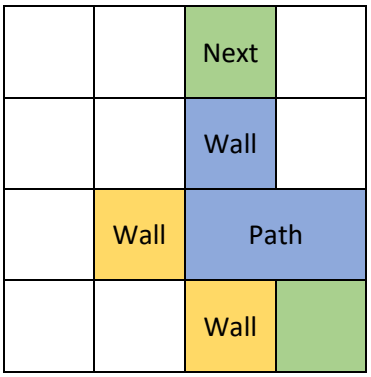
One of the randomly selected cells is selected as part of the path and four walls are placed around it.

Note that cells previously selected as part of the path or cells that would occur outside the area of the grid are not selected as possible walls. For this reason, only three walls are shown.



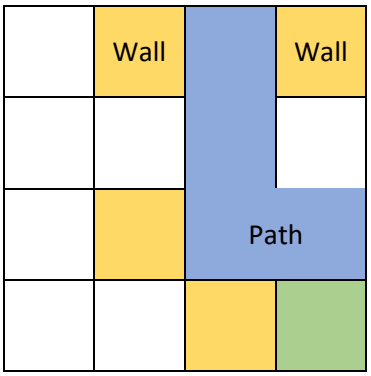
Step 4 – Path progression

The blue cells indicate the partially generated path, and the orange cells indicate cells that are currently selected as walls.



Step 5 – Penetrating wall cell

Randomly select one cell out of the previously generated walls and penetrate (pass through) it to select the next cell in a forward motion from the wall cell.



Step 6 – Iterative path development

The process continues by adding new wall cells to surround the selected cell and repeating the above steps to further generate the path.

By repeating this process, a complete maze pattern can be generated as in Figure 2.11.

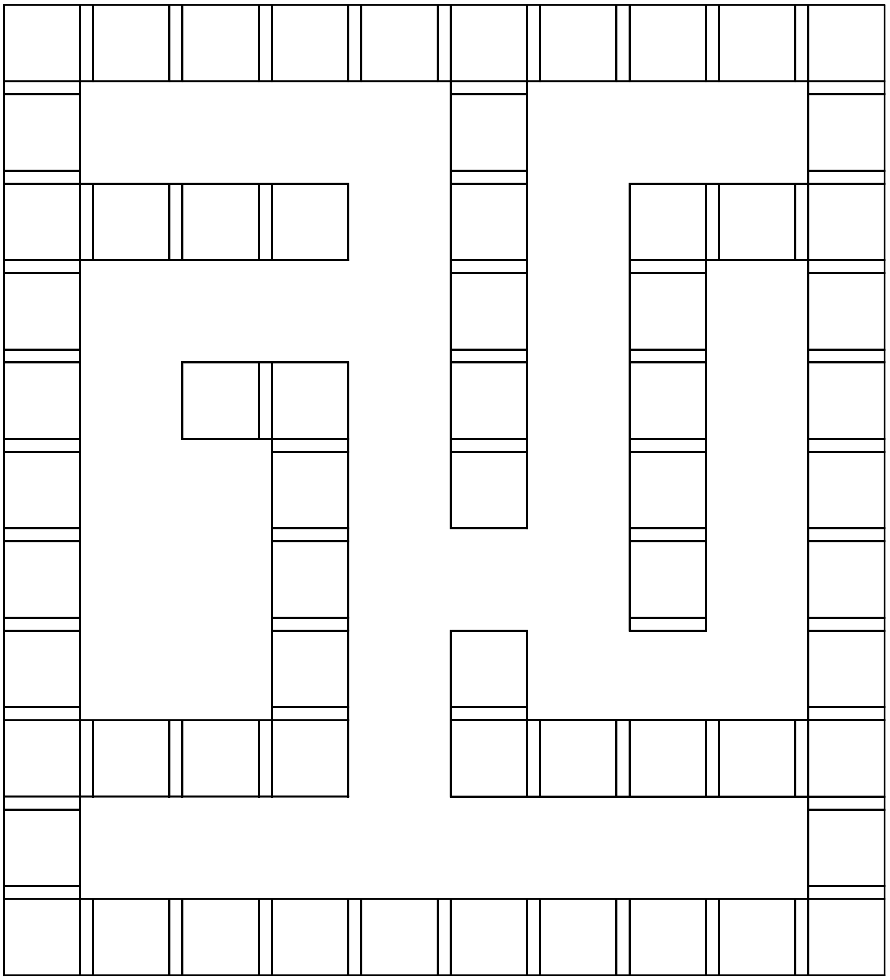


Figure 2.11 Example maze generated from Prim's algorithm

The pseudocode of Prim's algorithm is presented in Algorithm 2.1.

Algorithm 2.1: Prim's maze generation algorithm (Foltin, 2011)

Input: S - Set of cells
T - Set of edges
Output M - Maze is generated
Process: Generating a perfect maze from a set of cells by Prim's algorithm

1. declare c: cell and e: edge
2. **Select** a random cell $c \in S$
3. $M \leftarrow c$
4. **while** M is not full **do**
5. | **Select** an unmarked random cell $c \in S$
6. | **if** c is adjacent to one of the cells in M **then**
7. | | $M \leftarrow M \cup c$
8. | | $e = \{c, c\,(adjacent)\}$
9. | | remove e from T
10. | **endif**
11. | mark c so it will not be selected again in S
12. **end while**

Using Prim's algorithm to generate a maze is very popular, since it can generate a perfect maze (a maze that will always yield a solution), based on a minimum spanning tree of all the cells. This allows for multiple implementations in various fields of science. The following paragraphs are brief discussions of different case studies in which the algorithm is implemented.

Manen *et al.* (2013) use a randomised version of Prim's algorithm in generic object detection software. The aim of this software is to give computers vision, enabling them to detect objects within an image. The algorithm is used to generate a minimum spanning tree, using the pixels of a picture as the collection of vertices and the connections between the super-pixels (grouping of pixels with similar colours or grey levels) as its edges.

In cases of power outages, where the power supply is isolated from a main network, it is important to maintain a minimum shortage of power flow within the affected area. Sudhakar and Srinivas (2011) determine the appropriate switching of power lines within a distribution network. Prim`s algorithm is selected to perform this task since it gives an optimal path that visits each of the vertices (power supplies) through which power will flow.

Wang and Hsieh (2018) identify areas that could possibly become isolated in the case of a natural disaster. Areas that are most likely to be affected by a natural disaster are targeted and then used to construct a tree according to Prim's algorithm by using roads and bridges as the vertices. From this tree, it can be determined if resources would be required at certain areas, given the possibility of restricted movement when certain vertices (roads and bridges) are demolished by a natural disaster. The system further determines possible disaster relief areas through the analysis of the decision tree model and relevant patterns that will indicate if the area might become isolated or not.

Iqbal *et al.* (2017) discuss the shortest route for planting fibre optic cables in a city or town, using the principles of Prim's algorithm. By identifying the best route before any cables are installed, it is possible to save costs through the elimination of any excess cables that would have been used.

Due to the versatility of Prim's algorithm, many different studies use the algorithm to accomplish specific research goals. Examples of such studies are briefly mentioned in table 2.3.

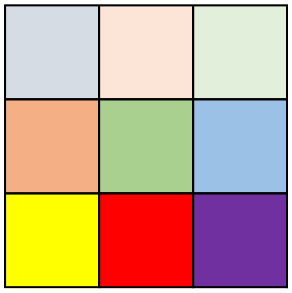
Table 2.3 Additional literature examples for Prim`s algorithm

Authors	Year	Title
Mariano, A., Lee, D., Gerstlauer, A. and Chiou, D.	2013	Hardware and software implementations of Prim's algorithm for efficient minimum spanning tree computation
Patil, A. and Nipanikar, S.	2017	Implementation of MATLAB-based self-healing grid using Prim's algorithm
Sudhakar, T.D. and Srinivas, K.N.	2010	Prim's Algorithm for loss minimisation and service restoration in distribution networks
Sudhakar, T.D. and Srinivas, K.N.	2011	Power system reconfiguration, based on Prim's algorithm
Wang, W., Huang, Y. and Guo, S.	2011	Design and implementation of GPU-based Prim's algorithm

2.3.2. Kruskal's algorithm

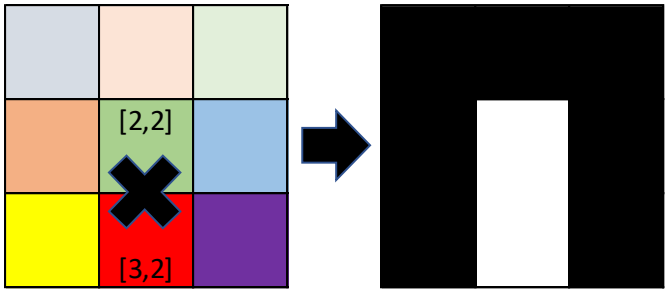
Joseph Bernard Kruskal was born to Jewish parents in New York City in 1928. Kruskal attended the University of Chicago where he completed a bachelor's degree and a master's degree in mathematics. He later continued to complete a PhD at Princeton University in 1954. In the field of computer science, Kruskal is most famous for his work that led to Kruskal's algorithm (Kruskal, 1956).

This algorithm is a greedy algorithm that aims to solve the minimum spanning tree problem but can be implemented within a connected or disconnected graph. Foltin (2011) describes the steps of the algorithm as follows:



Step 1 – Initial grid

Start by creating a set for each cell and a list of all the walls in a grid. Each differently coloured cell represents a different set.

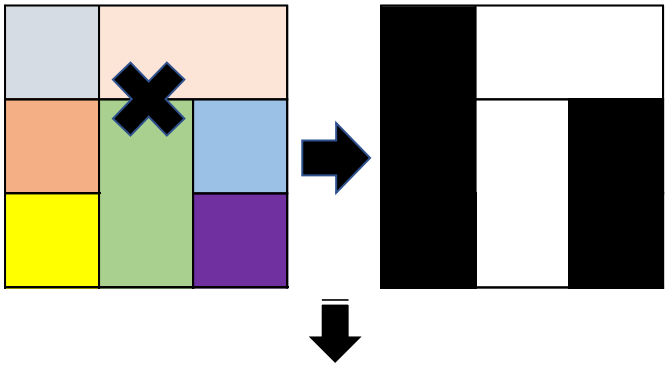


Step 2 – Random wall selection

Randomly select a wall from the list of walls.

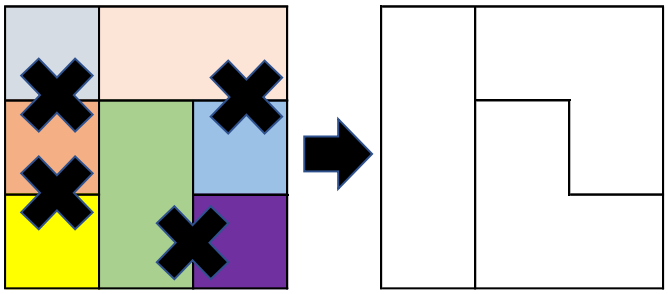
If the two cells connected by the wall are from different sets, the two sets will be linked, and the wall will be removed between cells [2,2] and [3,2] to create a new set containing two

cells. However, if the two cells connected by a wall are from the same set, the wall will be removed from the list of walls but will not be removed from the maze.

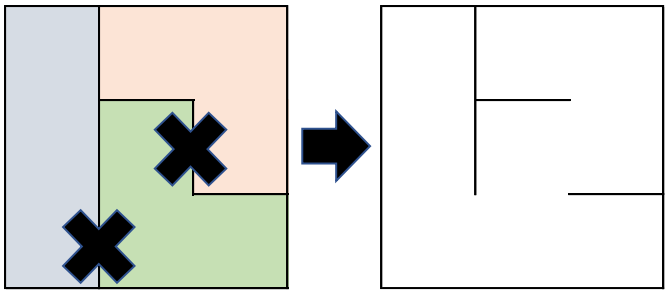


Step 3 – Select next random wall

The process is repeated by selecting a new random wall and joining the sets if they are from different sets.



After a few iterations of the process (until each cell is part of a larger set), a maze-like structure is



Step 4 – Combine all sets

The process will continue to remove walls until all sets are combined into one set or there are no walls available in the list of walls from which to select.

When all the sets are joined to form one large set, a maze structure is formed that represents the final maze generated by the algorithm.

The pseudocode of this maze generation algorithm is presented next in Algorithm 2.2.

Algorithm 2.2: Kruskal's maze generation algorithm (Foltin, 2011)

Input: S - Set of a collection of sets containing cells to explore
output: M - Maze
Process: Generating a perfect maze from a set of cells by Kruskal's algorithm

```
1. declare e: edge and c1, c2: cells
2. Select a random edge e = (c1, c2) ∈ S
3. M ← {(c1, c2)}
4. while number of sets in S >1 do //all cells do not belong to the same set
5.     Select a random edge e = (c1, c2) ∈ S with c1 and c2 in different sets
6.     M ← M ∪ {(c1, c2)}
7.     unify c1 and c2 in S into a single set
8. end while
```

According to Foltin (2011), one of the advantages of using Kruskal's algorithm is that fewer dead ends are generated which results in lower execution time. It also generates a perfect maze which causes an increase in the algorithm's popularity. A number of studies where Kruskal's algorithm has been employed is reported on in the literature. A few examples of these studies are briefly introduced in the following discussion.

Han and Lim (2010) discuss the use of this algorithm in an energy management system for a smart home. A smart home is an establishment that contains appliances, lights, doors, etc. that can communicate with each other over a network. This enables the user (homeowner) to control the components remotely or by a predefined time schedule. Kruskal's algorithm is used to minimise the total amount of power consumption within this network of connected appliances by seeking a path that would yield the lowest power consumption.

Dense traffic is a problem that arises in many densely populated cities. Lindorfer *et al.* (2013) describe the use of this algorithm in a simulation of street networks to improve the network structure and reduce the possibility of dense traffic. The algorithm is used to create a network of the highways that could form a bottleneck in the traffic.

When delivering packages, the length of the road travelled comes second to the time it takes to reach the destination. Ribeiro and Laporte (2012) use a modified version of Kruskal's algorithm to help solve the cumulative capacitated vehicle routing problem. The modified algorithm does not iterate until all the vertices are visited, but rather stops as soon as two connected components are found.

Table 2.4 presents more examples of studies where the algorithm was successfully applied.

Table 2.4 Additional literature examples for Kruskal's algorithm

Authors	Year	Title
Katsigiannis, A., Anastopoulos, N., Nikas, K. and Koziris, N.	2012	An approach to parallelise Kruskal's algorithm using helper threads
Montemanni, R. and Gambardella, L.M.	2005	A branch and bound algorithm for the robust spanning tree problem with interval data
Moret, B.M. and Shapiro, H.D.	1991	An empirical analysis of algorithms for constructing a minimum spanning tree
Quirin, A., Cordón, O., Guerrero-Bote, V.P., Vargas-Quesada, B. and Moya-Anegón, F.	2008	A quick MST-based algorithm to obtain Pathfinder networks
Sudhakar, T.D. and Srinivas, K.N.	2011	Power system restoration based on Kruskal's algorithm

The principle of randomised Prim's and Kruskal's algorithms is essentially the same, which is to create a perfect maze, based on a minimum spanning tree algorithm. The difference is that the former algorithm functions by adding nodes or cells while the latter algorithm

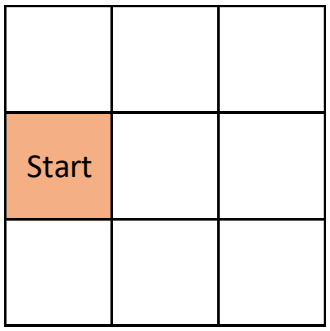
focuses on adding and removing edges or walls. Since both the algorithms are two of the most popular algorithms used in maze generation, they will be considered in the generation of a maze for this study. However, to gain a better understanding of how different mazes are generated, two additional algorithms will be discussed briefly. They are the recursive backtracking and hunt-and-kill algorithms.

2.3.3. Recursive backtracking algorithm

The recursive backtracking algorithm used to generate a maze is based on a depth-first search algorithm. The depth-first search algorithm searches through a grid by selecting (if possible) the child node of a previously visited node and continues until there are no available child nodes left (Foltin, 2011). At this point, the algorithm backtracks to the most recently visited node and repeats the process by selecting a new child node or backtracking even further. The depth-first search algorithm stops when all the nodes are visited.

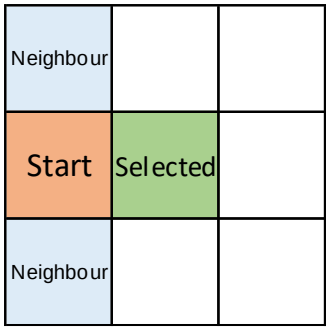
Randomisation is introduced to influence how the next child node is chosen. The first randomization in the recursive backtracking algorithm occurs when the starting cell is chosen. From the entire grid, a random cell will be selected and set as the starting cell whereafter the second randomisation occurs. A random neighbouring cell from the current cell is then selected to be explored next. As soon as there are no more available neighbouring cells that were not yet visited, the algorithm backtracks to the previously explored cell and continues the process of selecting a random cell. The algorithm will stop as soon as the starting cell is reached again through the backtracking process.

The recursive backtracking algorithm accomplishes its task by means of three main processes. It randomly selects a starting cell, randomly selects a neighbouring cell and backtracks if no neighbours are left. This will be graphically illustrated in the following six steps.



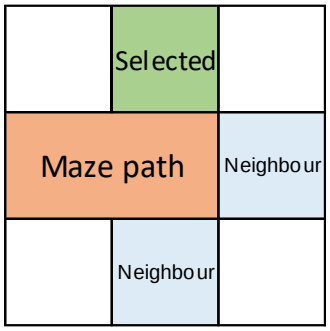
Step 1 – Select starting cell

Start by randomly selecting a cell from the entire grid and set it as the starting cell.



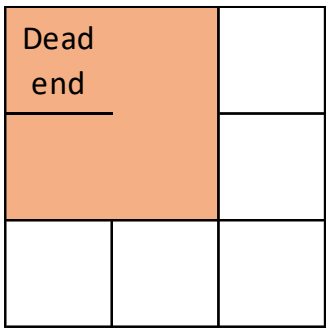
Step 2 – Select neighbouring cell

Randomly select one of the neighbouring cells that will be explored next.



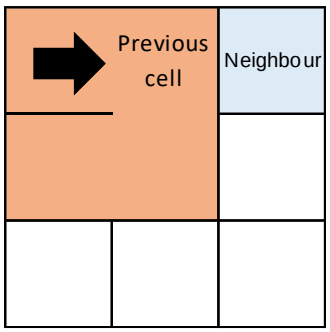
Step 3 – Maze path progression

The wall separating the previous cell and the newly selected neighbouring cell is removed, connecting the cells and forming part of the maze. Next, step 2 is repeated to select a new neighbouring cell.



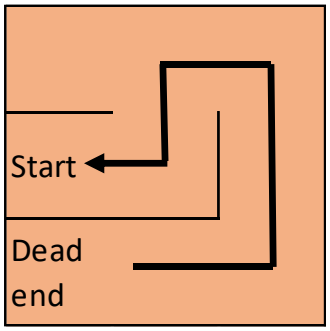
Step 4 – Dead end reached

Steps 2 and 3 are repeated until the currently selected cell has no neighbouring cells available from which to select.



Step 5 – Backtracking

Backtrack to the previously selected cell and repeat Steps 2 and 3.



Step 6 – Algorithm completion

Steps 2 to 5 are repeated until all the cells are visited and the backtracking process reaches the initial starting cell again whereafter the recursive backtracking algorithm completes.

Since the structure of the recursive backtracking algorithm is based on the depth-first search algorithm, it will deliver a perfect maze when completed. The following pseudocode demonstrates the recursive backtracking algorithm.

Algorithm 2.3: Recursive backtracking algorithm (Foltin, 2011)

Input: *S* - Set of cells
E - Set of edges
Output: *M* – Maze is generated
Process: Generating a perfect maze from a set of cells by the recursive backtracking algorithm

```
1.  declare c1, c2: cell and e: edge
2.  Select a randomly chosen cell c1 ∈ S.
3.  while c1 != starting c //executes until algorithm backtracks to starting position
4.      If current cell c1 has at least one unvisited neighbour
5.          choose neighbouring cell c2 at random
6.          e = {c1, c2}
7.          remove e from E
8.          c1 = c2
9.      else If c has no unvisited neighbours
10.         c1 = parent cell of c1 //Backtrack to the previously used cell
11. end while
```

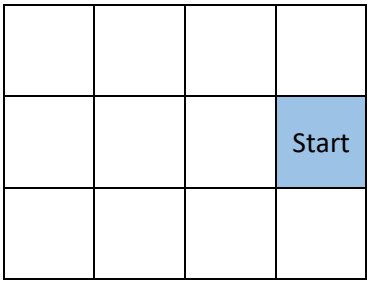
The recursive backtracking algorithm is often associated with search algorithms that are widely implemented in many fields where it is necessary to search through large volumes of data. It is also frequently used in maze generation studies and examples of such studies include Sukumar and Santha (2012) and Alsegård (2017). The former applied the algorithm in the field of steganography while the latter study employed the algorithm in an artificial intelligence application.

2.3.4. Hunt-and-kill algorithm

The hunt-and-kill algorithm may sound inappropriate when associating it with a task like maze generation, but in reality, the algorithm is quite adequate. The main idea behind the hunt-and-kill algorithm is that it generates a maze by going on a random walk through a grid, “killing” the walls it passes through, creating the maze path along its journey. When a dead end appears along the walk, the “hunt” part begins by scanning the grid for any cells not yet visited and continuing the journey (Lee *et al.*, 2010). By its nature, the algorithm forms a spanning tree of the different possible paths followed during the walk. This means that the algorithm will always produce a perfect maze, since any two nodes in a tree have one and only one path which matches the property of a perfect maze.

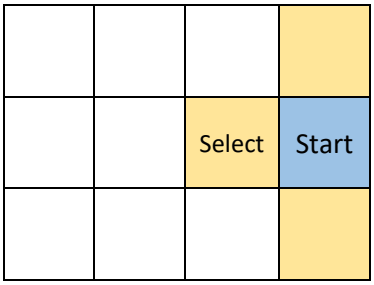
The hunt-and-kill algorithm is highly focused on randomisation to accomplish its tasks of walking through the maze and selecting a new position from which to continue when a dead

end is encountered. The process of the hunt-and-kill algorithm is graphically illustrated in the following six steps as described by Lee *et al.* (2010).



Step 1 – Randomly select a starting cell

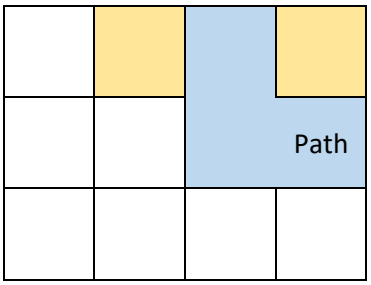
A starting cell is selected at random.

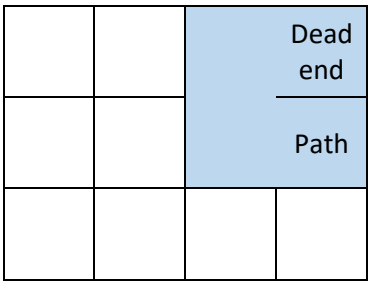


Step 2 – Randomly select a neighbouring cell

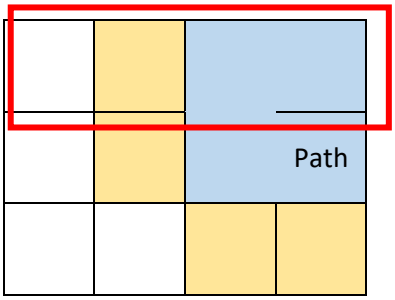
Randomly select a neighbouring cell (orange cell) that was not previously visited yet and "kill" (remove) the wall to add the new cell as part of the maze path.

Step 3 – Algorithm progression





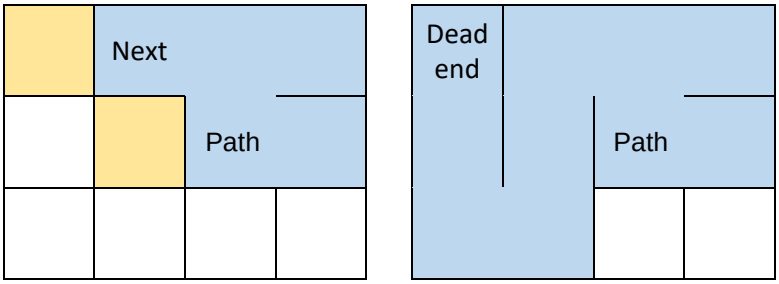
Next, the algorithm continues to walk through the grid by repeating Step 2 until a dead end occurs or all the cells are visited.



Step 4 – Select new neighbouring cell

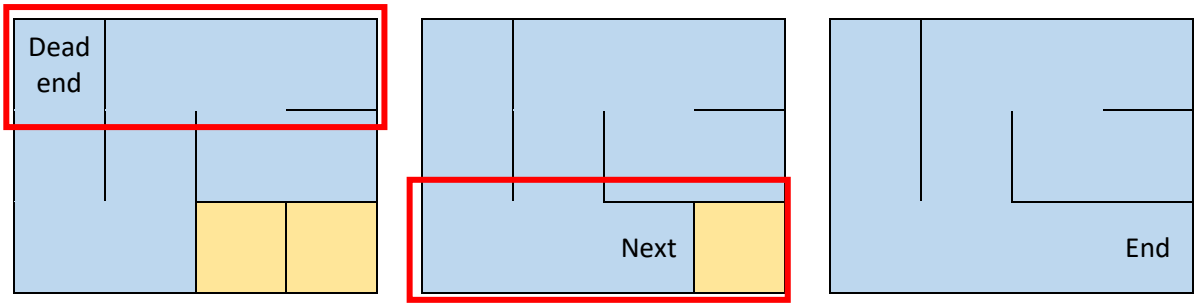
When a dead end is reached, the "hunt" (search) process begins. The algorithm starts to search for the first available cell (row by row) that is a neighbour of a previously visited cell.

Step 5 – Algorithm progression



Again Step 2 is repeated until a dead end occurs or all the cells are visited. In the case of a dead end, Step 4 is repeated.

Step 6 – Final progression of algorithm



Steps 4 and 5 are repeated until all the cells in the grid are visited and the maze is complete. These steps are explained using pseudocode of the algorithm as shown in Algorithm 2.4.

Algorithm 2.4: Hunt-and-kill algorithm (Foltin, 2011)

Input: *S1* - Set of unvisited cells
S2 - Set of visited cells
E - Set of edges
Output: *M:* Maze is generated
Process: Generating a perfect maze from a set of cells by the hunt-and-kill algorithm

```
1.  declare c1, c2: cell and e: edge
2.  Select a randomly chosen cell c1 ∈ S1
3.  S2 ← S2 ∪ c1
4.  while S2 does not contain all of S1
5.      If the current cell c1 has at least one unvisited neighbour
6.          choose neighbouring cell c2 at random
7.          e = {c1, c2}
8.          remove e from E
9.          c1 = c2
10.         S2 ← S2 ∪ c1
11.     else randomly select cell from S2 with available neighbours
12. end while
```

The hunt-and-kill maze generation algorithm is less popular than the other algorithms discussed. However, its usability and relatively low processing time make it suitable in many

studies. An example of the application of the hunt-and-kill algorithm can be found in Zehetmeier *et al*. (2019).

2.4. Summary

In Chapter 2, comprehensive background knowledge of maze generation algorithms was provided. Four well-known maze generation algorithms were reviewed and explained. The concepts of a maze and grid-based structures were also defined and examples from the literature were quoted where the algorithms were applied and implemented in other studies. While the focus of this chapter was on the generation of a maze, in the next chapter, the focus will be on maze-solving strategies.

Chapter 3 Solution strategies for grid-based mazes

3.1. Introduction

In the previous chapter, some background knowledge was gathered on various maze generation algorithms to gain a proper understanding of how mazes function and how they are generated. In order to assist the relief efforts of humanitarian logistics, a path must be found that can navigate response teams from a start point to a point of interest in the fastest way available. When viewing the disaster-stricken area as a maze (as discussed in Chapter 2), it is important to be able to solve this maze in an efficient way that will ensure each point of interest is reached within an acceptable time. Since a wide variety of maze-solving algorithms exists, it is important to gain proper knowledge of some of the most influential maze-solving algorithms.

The goal of this chapter is to present background knowledge of maze-solving algorithms. Four well-known maze-solving algorithms were chosen for this review. These algorithms are the Lee algorithm (Lee, 1961), the A-star algorithm (Hart *et al.*, 1968), the flood-fill algorithm (Law, 2013) and finally, the recursive backtracking algorithm (Walker, 1960). The choice of algorithms was based mainly on the popularity and versatility of the maze-solving technique. In addition, the selection of algorithms also attempts to show the wide variety of applications in different fields of study and emphasises the range of possibilities offered by different maze-solving strategies. Firstly, humanitarian logistics will be discussed whereafter the different maze-solving algorithms will be explained. Brief literature reviews of how the algorithms were applied and implemented in different studies will also be provided.

3.2. Humanitarian logistics

The term logistics is widely used in many fields involving the procurement and distribution of products or services. Christopher (2016) formally defines logistics as a strategic process of managing the procurement, transportation and storage of materials, parts, and finished products through an organisation and all its marketing channels. This has to be done in an efficient way to ensure that the current and future profitability are maximised in a cost-effective manner that will guarantee the completion of orders.

Humanitarian logistics entails all the steps of the logistics process, but implemented in such a way as to promote the welfare of all humans. Apte (2010) defines humanitarian logistics as a special branch of logistics that manages the response supply chain of critical supplies and services within a challenging environment (particularly in disaster situations). These

environments may include areas with large demand surges, areas requiring uncertain supplies, areas with a critical time window given infrastructure vulnerabilities and operations of a vast scope and size.

Duran *et al.* (2013) classify humanitarian logistics as a bridge between disaster preparedness and response. It involves being prepared for the procurement of supplies and their distribution between headquarters and the field when the situation arises. Humanitarian logistics is responsible for ensuring that all the correct relief supplies are gathered and delivered to the correct locations. Furthermore, efficient delivery of all these related relief items must be maintained during operations. The possible relief items (Figure 3.1) are categorised as follows.

- *Consumable items* – those items that need to be provided to affected individuals on a continuous basis;
- *Essential items* - this forms part of the consumable items and include items, such as water, food, hygiene and sanitation;
- *Non-consumable items* – any items that only need to be delivered to the affected environment once, after which the items are classified as operational or non-operational;
- *Operational items* – includes all the equipment required by relief personnel to accomplish their tasks and communicate efficiently; and
- *Non-operational items* - all the different supplies delivered to affected people that when received, can be used until the disaster has been cleared. These supplies include tents, blankets, utensils and other supplies that can ease their living situation.

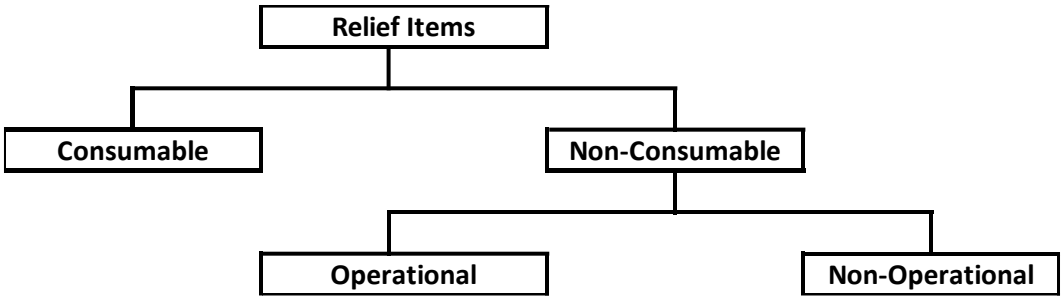


Figure 3.1 Relief item categorisation

One of the major tasks in humanitarian logistics, especially in disaster situations, is to determine appropriate locations where various facilities can be established to ensure that all resources are accessible. The placement of suitable facilities will not only provide safe and secure storage, but may give relief teams a strategic advantage in the timeous distribution of needed resources and relief items. Boonmee *et al.* (2017) discuss a possible solution to this problem by introducing the use of a facility location optimisation model. Using this model

alongside data gathered about pre- and post-disaster situations, appropriate locations can be determined for various facilities, including distribution centres, warehouses, shelters, debris removal sites and medical centres. Facility location models will also be used in this study and these models will be elaborated on further in Chapter 4.

Humanitarian logistics plays a significant and important role in the welfare of humans and is of particular importance in disaster and relief operations. Many studies have been conducted on this subject and a few examples from the literature are listed in the following paragraphs.

Griffith *et al.* (2017) proposed a decision support tool to address the problem of critical time management, i.e. the time taken to gather information before humanitarian logistics activities can be activated. The proposed tool is based on the use of analytical techniques, embedded within an open source platform, and allows decision makers to employ more complete techniques in analysing a disaster situation while maintaining efficient time constraints.

In a study by Shao *et al.* (2020), the deprivation cost (qualification of human suffering) is reviewed. Key issues and the development of deprivation cost are highlighted while the usefulness and contribution of using this cost in humanitarian logistics are pointed out.

The importance of simulation capabilities of games in the context of humanitarian logistics was emphasised in a study performed by Lukosch and Comes (2019). In this study, a framework was developed that can assist with the design of games to simulate specific environments and conditions in humanitarian logistics.

Sigala and Wakolbinger (2019) explained the potential of outsourcing in humanitarian logistics. They have shown that outsourcing may be a viable option and presented a set of criteria that can be used in the selection of appropriate outsourcing partners.

Humanitarian logistics is a dynamic field of study and many research studies on different topics related to the subject can be found in the literature. Further examples of such studies (which will not be elaborated on here) include Jahre *et al.* (2007), Rodríguez-Espíndola *et al.* (2018), and Widera *et al.* (2017).

3.3. Maze-solving algorithms

Solving a grid-based maze is of particular importance in this study as a maze will be used to represent a disaster-stricken area. An optimal route through the maze that can be used for disaster relief activities will be determined. There are many strategies, heuristics and algorithms to find a solution to a maze, however, some of these do not guarantee a solution

while others may find only sub-optimal solutions. One example of such a basic technique to solve a maze is called the *wall follower* method, also known as either the left-hand rule or right-hand rule. By simply keeping the left-hand side in contact with the wall of the maze, one is guaranteed to find an exit if it exists (Hendrawan, 2020).

In this section, four selected algorithms to solve a maze will be discussed. The algorithms are the Lee algorithm (Lee, 1961), the A-star algorithm (Hart *et al.*, 1968), the flood-fill algorithm (Law, 2013) and the recursive backtracking algorithm (Walker, 1960). The choice of algorithms was based on their versatility and popularity and they should provide an adequate overview of existing algorithms. Furthermore, the Lee and A-star algorithms will be applied in subsequent chapters to a real-life disaster scenario to illustrate the practical usefulness of maze-solving algorithms.

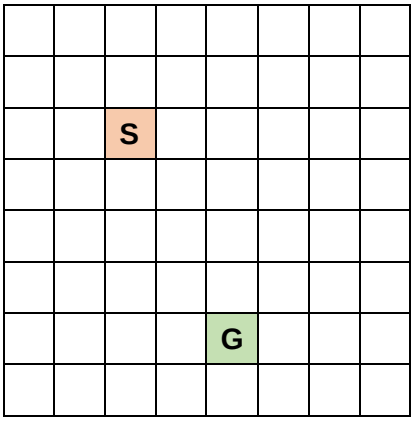
Each of the discussions will conclude with a few example studies from the literature.

3.3.1. Lee algorithm

The Lee algorithm was first formulated in 1961 as a way to allow computers to efficiently solve path-connection and optimal route-finding problems. The proposed method can solve several problems, including finding a path between two points while crossing the least number of paths, finding a path between two points while avoiding as many obstacles (walls) as possible, and finding a path between two points so that the path is optimal (e.g. shortest path) (Lee, 1961).

Gupta and Sehgal (2014) describe the Lee algorithm as a two-stage technique, i.e. a *filling* stage in which each cell is visited until the goal is reached, and a *retrace* stage where the path followed to the goal is traced back to the starting cell to construct the final path.

The algorithm works as follows:



Step 1 – Random grid

The algorithm starts with an empty grid with the starting (S) and goal (G) cells selected.

		2					
	2	1	2				
		2					
				G			

Step 2 – Add adjacent cells

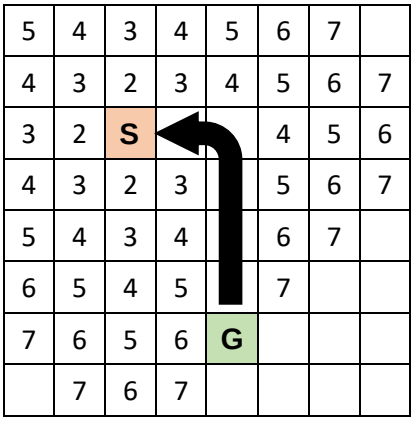
The starting cell (S) is filled with a 1 and the adjacent cells each with a 2. These values represent the distance from the starting cell.

Step 3 – Continue adding cells

Step 2 is repeated by continuously adding an increased value to neighbouring cells until the goal cell (G) is reached. In this scenario, the goal cell is reached with a value of 7.

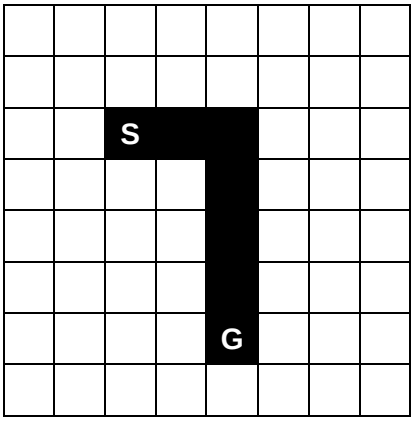
		3					
	3	2	3				
3	2	1	2	3			
	3	2	3				
		3					
				G			

5	4	3	4	5	6	7	
4	3	2	3	4	5	6	7
3	2	1	2	3	4	5	6
4	3	2	3	4	5	6	7
5	4	3	4	5	6	7	
6	5	4	5	6	7		
7	6	5	6	7			
	7	6	7				



Step 4 – Backtracking

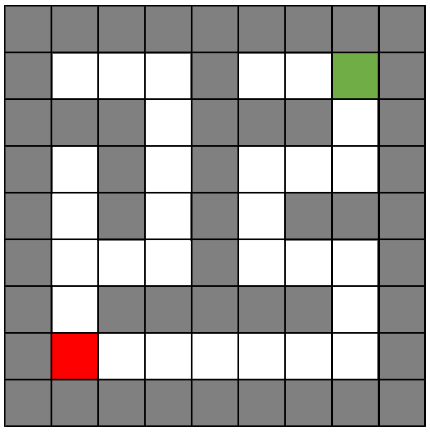
When the goal cell (G) is reached, the retrace stage starts by backtracking from (G) back to the starting cell (S). The backtracking is accomplished by selecting an adjacent cell to the currently selected cell (G) with a value of *i*-1, where *i* is the value of the currently selected cell.



Step 5 – Final path

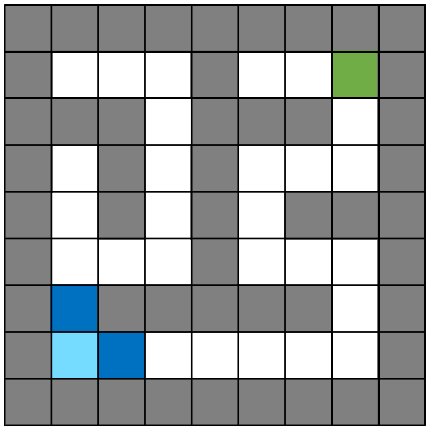
The final path is now generated by the algorithm, giving the path with the shortest distance from the starting cell (S) to the goal cell (G).

The above five steps illustrate the Lee algorithm in a grid. However, a grid is normally only the underlying element of a maze with limitations of edges (walls) that restrict movement. The following five steps illustrate graphically how a maze is solved using the Lee algorithm (the process is based on the steps of the above grid illustration).



Step 1 – Initial maze

Given a maze, the initial state is set by selecting a valid start (red) and goal (green) cell. Valid cells will include all cells that are not set as an edge (wall) cell and have at least one neighbouring cell that can be explored.

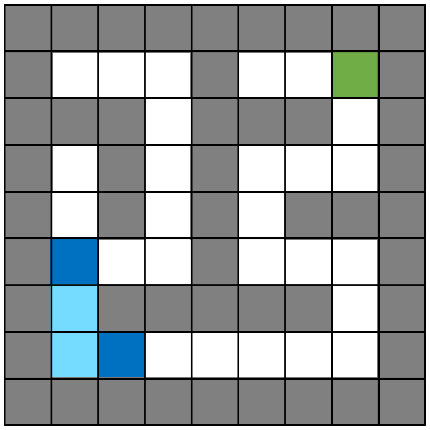


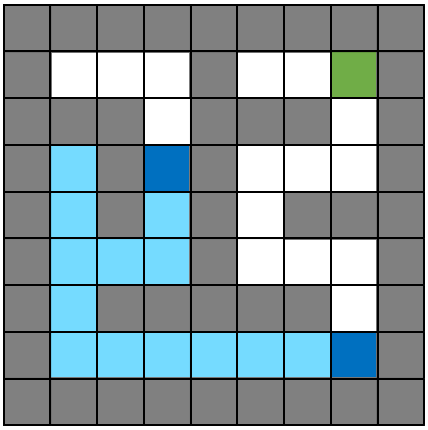
Step 2 – Cell expansion

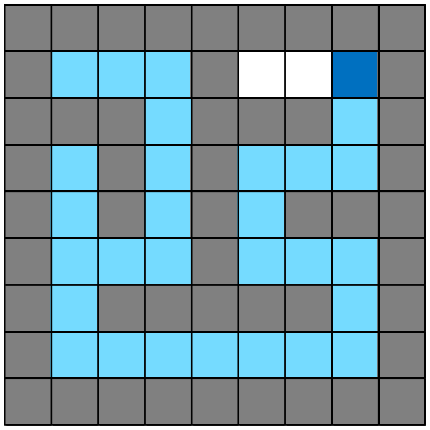
Using the starting cell to initiate the algorithm, direct neighbouring cells are selected to be explored next.

Step 3 – Explore neighbouring cells

With each iteration, only one of the neighbouring cells is expanded. Neighbouring cells are explored in a manner of first-in-first-out, which means that the oldest expanded cells will be explored first before moving on to each of their respective neighbouring cells.

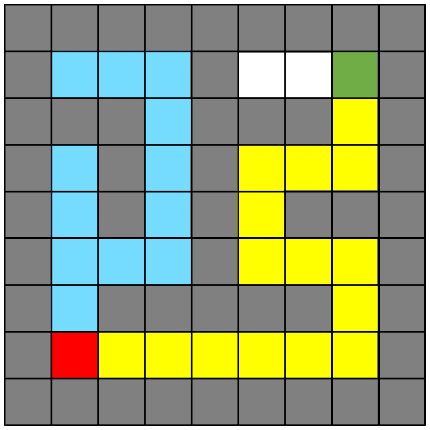






Step 4 – Reaching goal cell

The algorithm continues to iteratively expand each cell until the goal cell is reached; whereafter the algorithm stops and starts a backtracking phase to determine the generated route.



Step 5 – Backtracking phase

Backtracking is done by using the goal cell as the current cell and adding its parent cell to the path. The parent cell is then set as the new current cell and the step is repeated until the initial starting cell is reached. The path generated with this backtracking phase will be the path from the start to goal with the least number of expanded cells.

The pseudocode of the Lee algorithm is presented in Algorithm 3.1.

Even though Algorithm 3.1 does not always yield the fastest execution times or the lowest memory usage, it remains one of the most used algorithms due to its low complexity. Furthermore, the algorithm also guarantees a solution if one exists. A brief discussion of a few example case studies that were conducted using the Lee algorithm will be given in the following paragraphs.

Algorithm 3.1: The Lee Algorithm

Input: *start_node* - the source node to start from
goal_node - the goal node to reach
Output: *PATH* - list from *start_node* to *goal_node*
lists: *OPEN, CLOSE* and *PATH*

```
1.  push start_node to OPEN
2.  while OPEN is not empty
3.      current_node ← first element of OPEN
4.      If current_node = goal_node
5.          break to line 15
6.      end if
7.      for each neighbour of current_node
8.          If neighbour is not a wall
9.              push neighbour to OPEN
10.         end if
11.     end for
12.     push current_node to CLOSED
13.     pop current_node from OPEN
14. end while
15. while current_node != start_node //backtracking phase
16.     push current_node to PATH
17.     current_node ← parent of current_node
18. end while
19. return PATH
```

Gupta and Sehgal (2014) discuss the use of different techniques and algorithms, including the Lee algorithm, used by autonomous robots to solve mazes. They concluded that despite certain shortcomings of the Lee algorithm (i.e. being slow with high memory requirements), the algorithm remains one of the best options, as it guarantees to deliver an optimal, shortest path from a starting position to a goal position.

Polanczyk *et al.* (2012), propose the use of the Lee algorithm instead of the heuristic A-star search algorithm in situations where dynamic environments exist (e.g. changes to the locations of obstacles). They recommend that a custom version of the Lee algorithm be implemented to take changes to obstacles into consideration and dynamically reassign path values to accommodate these types of change. They also concluded that their proposed version of the Lee algorithm proved to be considerably faster than the original Lee algorithm, as well as the A-star algorithm.

The Lee algorithm can also be implemented in industrial work environments to ensure that robots move optimally, with precision and in a safe manner (Daneshjo *et al*., 2019). Based on the assumption that every obstacle within the work environment should be programmed to be avoided by the robots, their study concluded that the Lee algorithm provides a simple solution for ensuring that robots can move freely within the environment, avoiding any collisions with obstacles.

Due to the efficiency and low complexity of the Lee algorithm, the algorithm is often implemented and regularly forms the topic of many studies. Table 3.1 lists more examples of such studies.

Table 3.1 Additional literature examples for the Lee algorithm

Authors	Year	Title
Bejarano, N.C., Ambrosini, E., Pedrocchi, A., Ferrigno, G., Monticone, M. and Ferrante, S.	2014	A novel adaptive, real-time algorithm to detect gait events from wearable sensors
Chi, H.Y., Tseng, H.Y., Liu, C.N.J. and Chen, H.M.	2018	Performance-preserved analogue routing methodology via wire load reduction
Adamatzky, A., Chiolerio, A. and Szaciłowski, K.	2020	Liquid metal droplet solves maze
Jiang, Y. and Yang, M.	2017	Hardware design of parallel switch setting algorithm for Benes networks
Lee, M.C., Jan, G.E. and Luo, C.C.	2020	An efficient rectilinear and octilinear steiner minimal tree algorithm for multidimensional environments
Nestor, J.A.	2002	A new look at hardware maze routing
Quislant, R., Gutierrez, E., Zapata, E.L. and Plata, O.	2018	Privatising transactions for Lee's algorithm in commercial hardware transactional memory
Reddy, A.V., Vinoth, G. and Chiranjeevi, G.N.	2018	Implementation of Lee's algorithm for different routing constraints
Wu, Y.R., Tsai, M.C. and Wang, T.C.	2005	Maze routing with OPC consideration
Zhang, Y.	2018	Deep reinforcement learning on 1-layer circuit routing problem

3.3.2. A-star algorithm

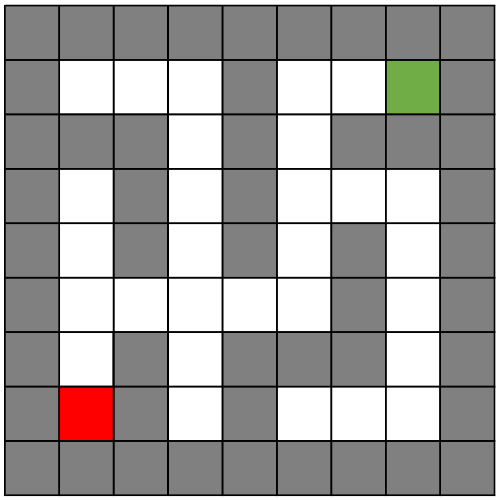
The A-star algorithm was first published in 1968 by Hart *et al.* (1968) and was initially created as part of a project that aimed at building a mobile robot that could plan its own actions. Although this can be perceived as an extension of Dijkstra's algorithm, it manages to achieve better performance by adding heuristics to direct the search for a path. This algorithm uses combinations of heuristic search and shortest path searching techniques to determine an optimal path (Duchoň *et al.*, 2014) and is defined as a best-first algorithm, since each cell in a grid-based environment gets evaluated with the following function:

$$f(v) = h(v) + g(v) \tag{3.1}$$

where $f(v)$ denotes the value assigned to each adjacent cell (used to determine the next expansion of the path), $h(v)$ is the heuristic distance from the initial state to the goal state, and $g(v)$ is the path length from the initial state to the goal state.

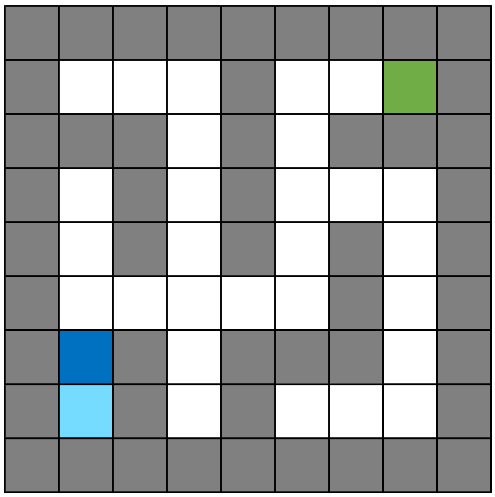
The function in (3.1) is applied to each adjacent cell of a currently expanded cell. The neighbouring cell with the lowest value for $f(v)$ is selected as the next cell to expand in the sequence. The selected cell is expanded and a $f(v)$ value is calculated and assigned to each of the neighbouring cells. This iterative process continues until the goal state is reached, whereafter a backtracking algorithm returns the optimal path. A significant

advantage of the algorithm is that the distance that is used as a criterion for the function can be adapted, changed or another distance measure (like time or energy consumption) can be added which extends the range at which the algorithm can be modified to fit specific needs. Maze traversal with the A-star algorithm happens in the same manner as described above but with limited expandable cells at each iteration. The following four steps, with graphical examples, illustrate how the A-star algorithm can be used to solve a maze.



Step 1 – Initial maze

Given a maze, the initial state is set by selecting a valid start (red) and goal (green) cell. A valid cell is any cell that is not set as an edge (wall) and which has at least one neighbouring cell that can be explored.

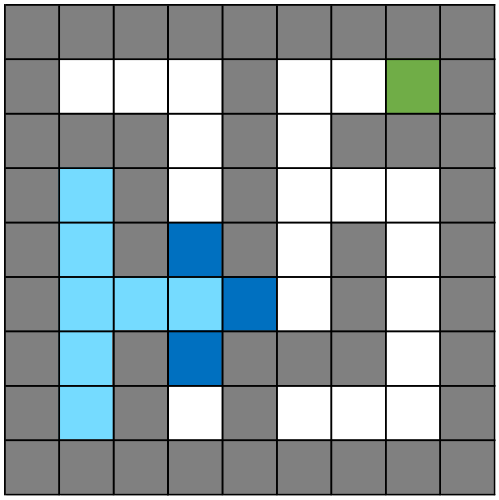


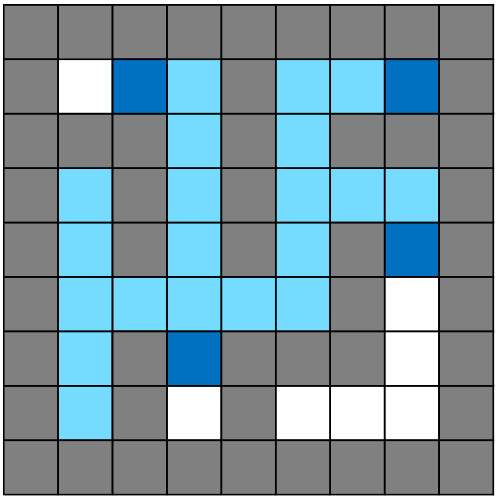
Step 2 – Initial cell expansion

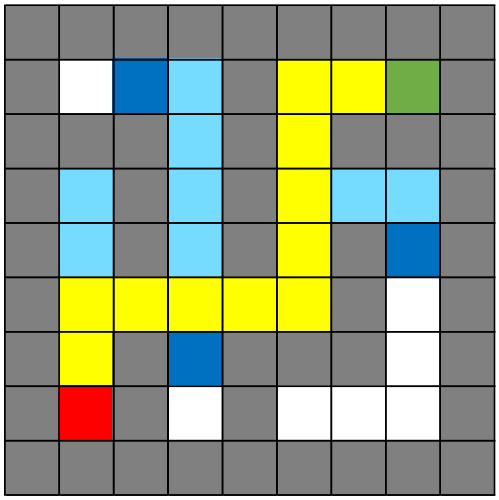
The starting cell is set as the current cell. The current cell is then expanded by adding its neighbouring cells and calculating a $f(v)$ value for each of them. The maze used in this example depicts limited movement as the wall cells only allow one neighbouring cell to be expanded.

Step 3 – Continues cell expansion

Step 2 is repeated iteratively by evaluating and expanding each valid (not separated by a wall) cell until the goal cell is reached. The algorithm will then stop the search process and start to apply a backtracking algorithm to determine the optimal path.







Step 4 – Backtracking phase

Backtracking is done by setting the goal as the current cell and adding its parent cell to the path. The parent cell is then set as the new current cell and the step is repeated until the initial starting cell is reached.

The pseudocode of the A-star algorithm is presented in Algorithm 3.2.

The A-star algorithm is generally more complex to implement and also uses more memory to execute than, for example, the Lee algorithm. However, the faster execution time is a major advantage and entices a significant number of researchers to use the A-star algorithm in their research. Some examples of such studies are presented in the following paragraphs.

Frey *et al.* (2011) discuss the use of a modified version of the A-star algorithm to aid in the design of electrical harness wiring and pipe layouts within aircrafts. An additional value is added to the original function, which is then used to modify the total cost, based on their proximity to given obstacle types. This allows designers to avoid certain obstacles during route planning while simultaneously permitting the path to favour certain obstacles and manoeuvre closer to them. This means that the harness can be routed closer to certain structures while avoiding areas that might damage the harness, i.e. areas with high heat.

Algorithm 3.2: A-star Algorithm

Input: *start_node* - the source node to start from
goal_node - the goal node to reach
Output: *PATH* list from *start_node* to *goal_node*
lists: *OPEN, CLOSE* and *PATH*

```
1.  push start_node to OPEN with f(start_node) = h(start_node) + g(start_node)
2.  while OPEN is not empty
3.      current_node ← pop from OPEN with the lowest f(current_node) value
4.      if current_node = node_goal
5.          break to line 27
6.      end if
7.      for each neighbour of current_node
8.          neighbour_current_cost ← g(current_node) + w(current_node, neighbour)
9.          if neighbour is in OPEN
10.             if g(neighbour) ≤ neighbour_current_cost
11.                 continue to line 7
12.             end if
13.         else if neighbour is in CLOSED
14.             if g(neighbour) ≤ neighbour_current_cost
15.                 continue to line 7
16.             end if
17.             move neighbour from CLOSED to OPEN
18.         else
19.             push neighbour to OPEN
20.         end
21.         h(neighbour) ← heuristic distance to goal_node
22.         g(neighbour) ← neighbour_current_cost
23.         parent of neighbour ← current_node
24.     end for
25.     push current_node to CLOSED
26. end while
27. while current_node != start_node //backtracking phase
28.     push current_node to PATH
29.     current_node ← parent of current_node
30. end while
31. return PATH
```

In a study by Barnouti *et al.* (2016), the A-star algorithm is used to improve the core movement of artificial intelligence agents in computer games. Using an image that represents a map or a maze, the algorithm seeks to find the shortest path between an identified source and identified destination, so that certain obstacles on the map (i.e. mountains or bodies of water) are avoided.

A method to determine travel time to and from campus for students was proposed by Siregar *et al*. (2018). In this study the A-star algorithm was employed to determine optimal (time) routes from various boarding home locations to the learning facility thereby enabling students to make informed decisions when confronted with different housing options.

The popularity of the A-star algorithm has increased greatly over the past years due to its faster execution time and adaptability to specific needs. Table 3.2 lists additional examples of studies where the A-star algorithm was implemented.

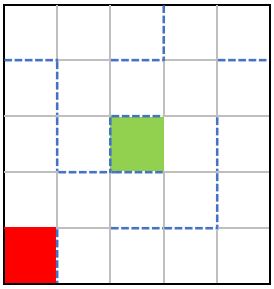
Table 3.2 Additional literature examples for the A-star algorithm

Authors	Year	Title
Cai, Y. and Ji, X.	2018	ASA-routing: A-Star adaptive routing algorithm for network-on-chips
Elgaroui, L., Chamberland, S. and Pierre, S.	2019	A new routing metric for real-time applications in smart cities
Febliama, A.B., Fitria, N.D. and Handayani, A.N.	2019	The application of A-star (A*) algorithm on the Android-based Pacman adaptation educational game as a learning media for SMK
Kurosawa, K., Uchiyama, Y. and Kosako, T.	2020	Development of a numerical marine weather routing system for coastal and marginal seas using regional oceanic and atmospheric simulations
Lu, D.N., Nguyen, T.H., Nguyen, D.N. and Nguyen, H.N.	2017	A novel traffic routing method using hybrid ant colony system, based on genetic algorithm
Nanda, A. and Rath, A.K.	2018	Fuzzy A-star, based cost effective routing (FACER) in WSNs
Reeves, M.C.	2019	An analysis of path planning algorithms focusing on A* and D*
Sun, D. and Li, M.	2016	Evaluation function optimisation of A-star algorithm in optimal path selection
Wu, C.M., Liaw, D.C. and Lee, H.T.	2018	A method for finding the routes of mazes
Zheng, C., Liu, H., Ge, M. and Liu, Y.	2019	A novel maze representation approach for finding filled path of a mobile robot

3.3.3. Flood-fill algorithm

Micro-mouse competitions (an event where small robot mice solve a 16×16 maze) have been held for decades and can be traced back to the year 1972. According to Law (2013), the flood-fill algorithm became one of the most commonly used algorithms in these competitions since contestants would use a modified version of flood-fill to reduce the number of cells visited before the maze is completed. In modern times, the flood-fill algorithm is one of the more popular algorithms used to solve a maze in robot maze problems because of its balance between finding the shortest path and discovering walls along the path. Each cell in a grid is assigned a value that indicates its distance from a target cell. The algorithm can then traverse the maze by moving to a neighbouring cell that is closer to the target than the current cell. This is indicated by the cell value which will be one less than the current cell. With each movement, the algorithm will check if any walls are present at the current cell and recalculate each cell's value (flooding), based on the currently discovered walls. This will prevent any movements to cells that are separated by a wall (Jabbar, 2016).

To illustrate the flood-fill algorithm, the following graphical explanation is given (Tjiharjadi & Setiawan, 2016).

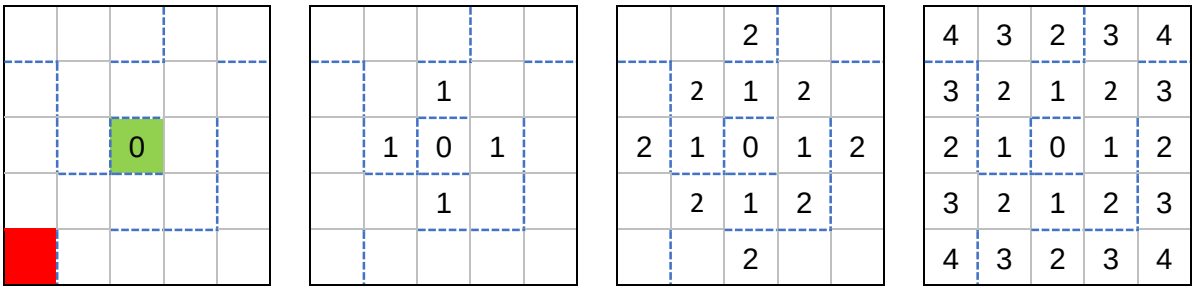


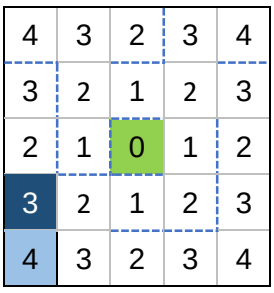
Step 1 – Initial maze

The algorithm starts with an initial maze and a predetermined starting cell (red) and goal cell (green). Notice the walls of the maze are faded out. This is because their locations are initially unknown to the search algorithm.

Step 2 – Cell value assignment

The goal cell is set as the source cell and the algorithm assigns a value to each cell that represents the total distance from this new source cell. This cell is assigned a value of 0 and its neighbouring cell a value just one higher (a 1 in this scenario). In an iterative manner, the algorithm continues to assign each neighbouring cell a higher cell value than the previous cell until the entire grid is populated.



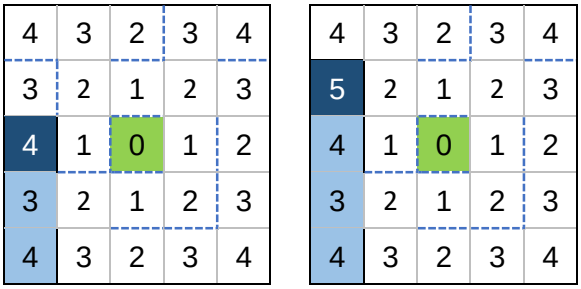


Step 3 – Initial movement

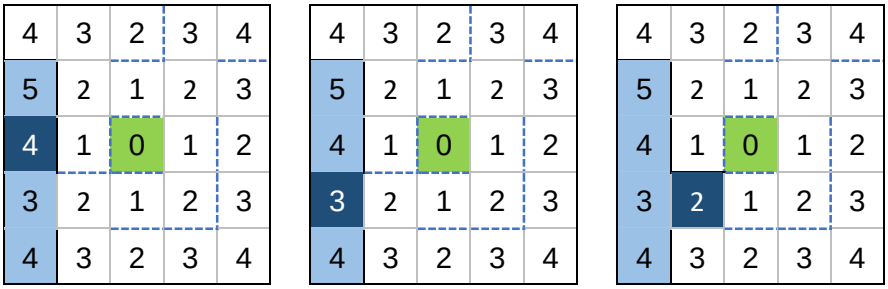
When the algorithm starts to explore the maze, it checks to see if any walls are prohibiting its movements. In this scenario, the starting cell is blocked in three directions (the right wall, as well as the left and lower bounds of the maze), allowing the first movement only to be made upwards. For each movement (dark blue square), a cell can only be chosen if its assigned value is smaller than that of the current cell.

Step 4 – Maze traversal

Step 3 is repeated until a new wall is identified. When this new wall appears, the algorithm assigns new values to each cell (flooding the grid) while taking the new walls into account. The cell separated by this wall will not be selected as a neighbouring cell when assigning values.

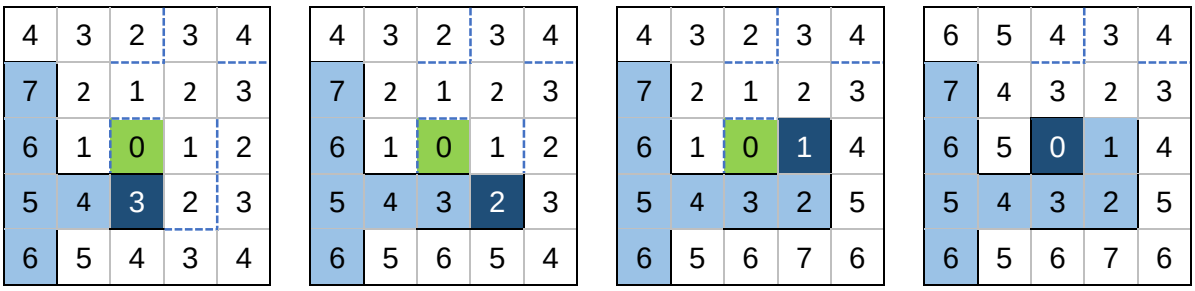


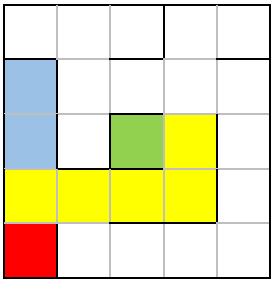
In this scenario, a dead end is reached. When this occurs, previously visited cells can be visited again, since the newly flooded values are less than that of the current cell (value of 5). In this example, a dead end is reached at a value of 5. Selecting a neighbour cell with a smaller value will be the cell with value 4 and then 3 until an unvisited neighbour is reached with a smaller value of 2.



Step 5 – Path completion

Steps 3 and 4 are repeated until the goal cell is reached, indicated by a cell value of 0.





Step 6– Maze path

The algorithm completes by backtracking from the goal cell to the starting cell. Backtracking is done by setting the goal as the current cell and adding its parent cell to the path. The parent cell is then set as the new current cell and the step is repeated until the initial starting cell is reached.

The pseudocode of the flood-fill algorithm is presented in Algorithm 3.3.

Algorithm 3.3: Flood-fill algorithm

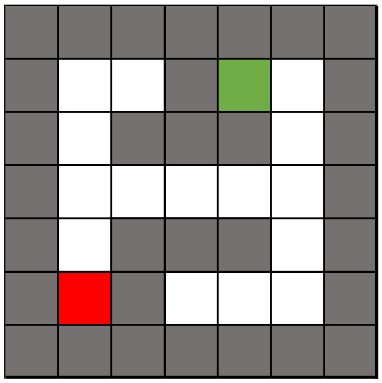
Input: *start_node* - the source node to start from
goal_node - the goal node to reach
Output: *PATH* - list from *start_node* to *goal_node*
lists: *OPEN*, *FLOOD*, *WALLS* and *PATH*
Process: *Generate a maze using the Flood-fill algorithm*

```
1.  set WALLS as empty
2.  current_node ← start_node
3.  while PATH is empty
4.      i ← 1 // counter to increment cell values
5.      while current_node != goal_node //flooding phase
6.          for each neighbour of current_node
7.              If neighbour is not in WALLS
8.                  push neighbour to OPEN with cell(i)
9.              end if
10.         end for
11.         i++ //increase counter
12.         push current_node to FLOOD with cell(i-1)
13.         current_node ← pop from OPEN
14.     end while
15.     current_node ← pop goal_node from FLOOD
16.     while current_node != start_node
17.         for each neighbour of current_node
18.             if neighbour is a wall
19.                 push neighbour to WALLS
20.                 continue to line 3
21.             else if neighbour(cell) < current_node(cell) //cell value
22.                 current_node ← neighbour
23.             else
24.                 continue to line 3 //dead-end reached
25.             end
26.         end for
27.     end while
28.     while current_node != goal_node //backtracking phase
29.         push current_node to PATH
30.         current_node ← parent of current_node
31.     end while
32. end while
33. return PATH
```

The ability of the flood-fill algorithm to discover walls in a maze and to adapt the path as required makes this algorithm a popular choice for many researchers. The algorithm is of particular importance to solve robot maze problems in which the objective is to enable a robot to traverse a maze. Examples of how the algorithm is used can be found in studies performed by Benavides *et al.* (2018) and Ranade and Manicannan (2019). The former study entails robotic path planning in a maze and the latter uses the algorithm to avoid obstacles in an unmanned air vehicle.

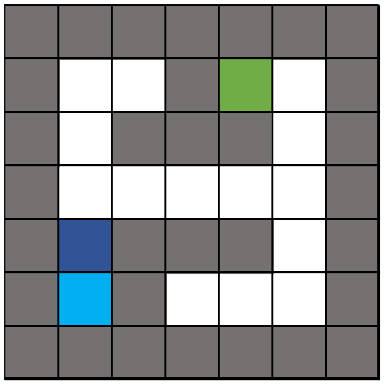
3.3.4. Recursive backtracking algorithm

As the name implies, the recursive backtracking algorithm makes use of recursion (repeated application of a procedure) to traverse a maze until a certain goal is reached. The process starts by selecting a neighbouring cell of the starting cell that is not separated by a wall. Any direction can be set to be expanded first and is usually set in the direction of the goal (if known). The technique continues to expand the neighbouring cell of each selected cell at the current iteration until there is no valid neighbouring cell left for the currently explored cell. When this occurs, the algorithm will traverse back the way it came until a cell is reached that has unvisited neighbours. It will then continue to adapt its path whenever dead ends occur until the goal cell is reached. Reaching the goal cell will start a backtracking process to determine the path used to reach the goal (Niemczyk and Zawislak, 2020). This recursive backtracking method produces the first path that leads to the goal and does not guarantee a shortest path. A graphical explanation of the algorithm is presented below.



Step 1 – Initialisation

The recursive backtracking algorithm starts with an initial maze and a predetermined starting cell (red) and goal cell (green). Since the goal cell is known in this example, the upward movement will be used, as it is also known that the goal cell is at the top of the maze.

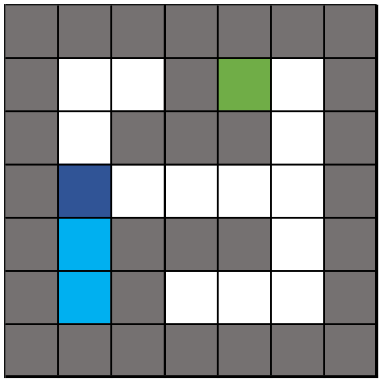


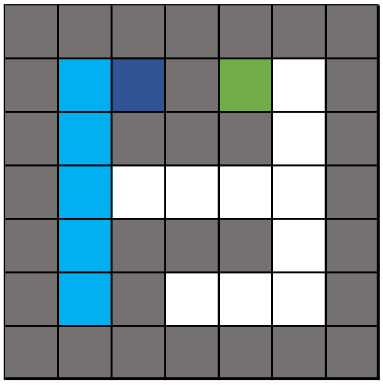
Step 2 – First traversal

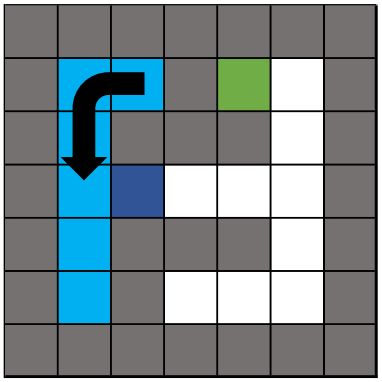
Continue to determine all available neighbouring cells that were not already visited, and which are not separated by a wall. A neighbouring cell will be chosen and set as the new current cell which will be explored next.

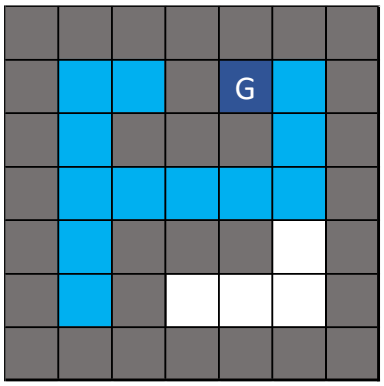
Step 3 – Traversal/Dead end

Step 2 is repeated until the goal is reached or until a dead-end occurs along the path. In the case of a dead end (as in this scenario), the algorithm will backtrack until a cell is reached that has a valid neighbouring cell that was not previously visited. This newly selected neighbouring cell will then be further explored.



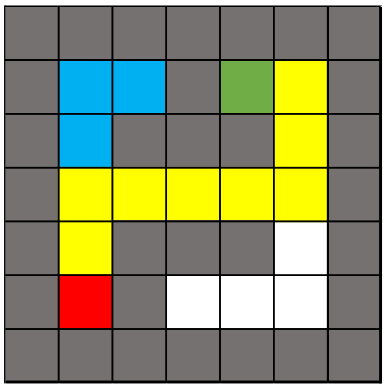






Step 4 – Complete traversal

Steps 2 and 3 are repeated until the goal cell is reached. When this occurs, a backtracking process is initiated to determine the path.



Step 5 – Backtracking path

The algorithm completes by backtracking from the goal cell to the starting cell. Backtracking is done by setting the goal as the current cell and adding its parent cell to the path. The parent cell is then set as the new current cell and the step is repeated until the initial starting cell is reached.

There is a technical drawback attached to the use of the recursive backtracking method. To avoid stack overflow problems and excessive memory usage, most programming environments limit the recursion depth which may impede the efficient use of the algorithm. However, the recursive backtracking approach remains a useful technique and may be found in many research studies. Some examples of the use of the algorithm include Gupta *et al.* (2019) who apply the recursive backtracking method to fix possible distortions in QR codes, Karlsson (2018) who uses the technique in a comparative study of different maze-solving algorithms, and Silva *et al.* (2017) who employ the algorithm to program a maze-solving robot.

The pseudocode of the recursive backtracking algorithm is presented in Algorithm 3.4.

Algorithm 3.4: Recursive backtracking algorithm

Input: *start_node* - the source node to start from
goal_node - the goal node to reach
Output: *PATH* - list from *start_node* to *goal_node*
lists: *OPEN* and *CLOSE*
Process: Generate a maze using the Recursive backtracking algorithm

```
1.  push start_node to OPEN;
2.  while OPEN is not empty
3.      current_node ← pop from OPEN;
4.      if current_node = goal_node
5.          break to line 22;
6.      end if
7.      if neighbour_up != wall & not in CLOSED // tests each neighbour, starting with upwards
8.          push neighbour_up to OPEN;
9.      else if neighbour_right != wall & not in CLOSED
10.         push neighbour_right to OPEN;
11.     else if neighbour_down != wall & not in CLOSED
12.         push neighbour_down to OPEN;
13.     else if neighbour_left != wall & not in CLOSED
14.         push neighbour_left to OPEN;
15.     else
16.         current_node ← pop last value of CLOSED; // backtracks to previous value
17.         push current_node to OPEN;
18.         continue to line 2;
19.     end
20.     push current_node to CLOSED;
21. end while
22. while current_node != start_node //backtracking phase
23.     push current_node to PATH;
24.     Current_node ← parent of current_node;
25. end while
26. return PATH;
```

3.4. Summary

In Chapter 3, the focus was on solution strategies for grid-based mazes which play an important role in humanitarian logistics. The chapter started with a brief introduction to humanitarian logistics which was followed by a discussion of four grid-based maze-solving algorithms. These algorithms were selected, based on their popularity and versatility and were graphically explained. The discussions were further elaborated on by including the pseudocode for each algorithm, as well as presenting examples from the literature of relevant applications.

Chapter 4 Discrete facility location models

4.1. Introduction

Facility location decisions are critical in the design of adequate systems to provide healthcare and relief supplies in disaster-stricken areas. Wrong decisions may have a serious impact on disaster victims and hard-to-access medical and food supplies are likely to increase mortality rates during a disaster. While the focus in previous chapters concentrated on the development of grid-based mazes that can be used to represent traversable routes in a disaster-stricken area, in this chapter, the focus will be on facility location problems. The aim in the chapter is to provide a background to facility location models and the formulation of mathematical models that will assist with decision making and which will help to maximise the relief effort at the lowest cost possible. A set covering model, as well as a maximal covering model will be developed in this study to determine the optimal placement of rescue facilities and the brief introduction of these types of model in this chapter will support the objective of formulating such models. Firstly, brief comments on the importance of healthcare and relief supply facilities (Section 4.2) are made. The core of the chapter is then presented in Section 4.3 in which various models for discrete location problems are described in detail. Aspects, such as a taxonomy and classification of the different models, as well as model descriptions and formulations are presented. Discussions are also backed up by relevant literature resources.

4.2. Facilities in disaster-stricken areas

Although facility location models will be introduced in section 4.3, it is important to first present a brief additional background section on the importance of healthcare and relief supply facilities in the event of a major disaster. Due to the serious and life-threatening situations that often occur during a disaster, logistics play a central role in any response activities. A humanitarian relief chain is normally established during disaster events with the objective to provide relief in the form of medicine, shelter, emergency food and water and other supplies to affected areas. To be able to comply with the requirements of a humanitarian relief chain, it is important to determine optimal locations for healthcare facilities, as well as supply facilities. In this section, a brief motivation from the literature on the importance of healthcare and supply facilities is presented.

4.2.1. Healthcare facilities in disaster-stricken areas

During a natural disaster, the scale of the damage caused to the surrounding environment, as well as to the well-being of people caught in its path is a great unknown. This means that a rapid response is required to ensure that no lives are unnecessarily lost. Organisations that provide health care support during disaster situations include military health care services, national rescue bodies, international organisations (i.e. the United Nations), coordination centres, international government organisations, non-governmental organisations, and any private groups that may be able to provide medical assistance and support (Bitterman and Zimmer, 2018). The support provided may vary from the acquisition and distribution of medical supplies to the establishment of medical and healthcare facilities.

The types of healthcare facility (i.e. portable, temporary or permanent) are dictated by the disaster situation. Permanent portable healthcare facilities can function continuously and can be transported to different areas as the need arises. These types of facility contain complete medical equipment and are ready to use as soon as they arrive, since they require no construction. The facilities are classified according to land, air and water facilities and are depicted in Figure 4.1

Figure 4.1 Examples of an earth-bound mobile hospital, flying hospital and floating hospital ship

Temporary healthcare facilities, as opposed to permanent facilities, comprise different non-functional components that can be transported to a desired area in various configurations. Examples of equipment may include foldable or collapsible flat components that are packaged together, and which are immediately ready for assembly. These components may consist of rigid type construction types, inflatable tent types, and regular framed tents. See Figure 4.2 for examples of such assembled temporary facilities.

Figure 4.2 Examples of temporary facilities: Rigid-type building, inflatable tent-type and framed tent

It is obvious that each of the different types of facility would play a crucial role during relief efforts in a natural disaster and choosing the most appropriate facility is therefore vital. Of equal importance is the number of facilities required and the identification of acceptable locations where they can be established.

A large number of research studies has been performed on facility location models in general and in healthcare facility location in particular (Ahmadi-Javid *et al.,* 2017). The importance of healthcare facility location models is evidenced by the large range of studies covering this and related problems. Studies range from the location of temporary blood banks (Sharma *et al.*, 2019) to facilities located to cater for the humanitarian need of refugees (Günay *et al.*, 2019) to the placement of field hospitals (Naor and Bernardes, 2016). Other examples of studies related to the location of healthcare facilities can be found in Gümüş and Celik (2017), Dascioglu *et al.* (2019) and Güneş *et al.* (2019). There is also a large number of studies that pertain to the mathematical and computational aspects of healthcare facility modelling and examples of such studies include Boonmee *et al.* (2017), Zokaee *et al.* (2016) and Babaei and Shahanaghi (2018), Finally, a large body of knowledge on this topic also exists in earlier work of researchers, such as Balcik and Beamon (2008), Jahre *et al.* (2007) and Ittmann (2005).

4.2.2. Relief supply facilities in disaster-stricken areas

Relief supply facilities and healthcare facilities are, to a certain extent, almost the same types of facility. However, natural disasters do not only put lives at risk in a way that can be solved with medical supplies, but also impact negatively on all survivors. Prior to a disaster, people have shelter and food security, normally for the foreseeable future. Following a disaster, people suddenly find themselves without a home, food to eat and clothes to wear. To assist these victims and to improve their survivability, humanitarian logistics is tasked with providing disaster victims with relief supplies to sustain them. Figures 4.3 and 4.4 provide some insight into what relief supplies are and how they are utilised.

Figure 4.3 Example of a typical disaster relief supply kit

Figure 4.4 Modular cardboard beds, use of drones and temporary shelters

There are many studies and strategies for the management and distribution of relief supplies during a disaster situation which also include pre- and post-disaster activities. One such strategy is to procure relief supplies ahead of time and pre-position them at facilities that are close to disaster-prone areas (Richter, 2016). However, pre-positioning relief supplies may not always be an adequate solution, as disasters can occur at any location at any given time. To overcome this problem, Richter (2016) proposes the use of large supply-holding ships, accompanied by smaller ships and/or helicopters to distribute the relief supplies to the affected area. Continuous relocation of supplies to facilities at different locations may also mitigate the problem, i.e. areas that are known to suffer from hurricanes may be prioritised for storing relief supplies.

Other examples of relief supply research include studies on the number and allocation of facilities during a natural disaster (Cavdur *et al.*, 2016). The study by Cavdur *et al.* (2016) is also an example of the use of mathematical modelling techniques to determine supply facilities, based on factors, such as walking distances, demand satisfaction and the utilisation of facilities, studies pertaining to specific needs, i.e. See *et al.* (2017) performed a study to highlight challenges specific to fresh water supply that are linked to problems, such

as portable (bottled) water, the treating (cleaning) of water and the restoration of water supplies, studies on the sustainability of relief efforts (Sebatli *et al.*, 2017) that involve the determination of the demand for relief supplies, and finally, studies on the pre- and post-management of disaster relief supplies, for example, the use of drones to determine appropriate sites for relief and recharge facilities, based on travel distances and coverage of large areas (Shavarani, 2019).

In Section 4.2, the aim was to provide a brief introduction to the concepts and importance of healthcare and relief supply facilities. In the next section, the mathematical and computational techniques that are used to perform discrete facility location modelling will be presented.

4.3. Models for discrete facility location problems

A typical facility location problem deals with selecting the best placement of a facility in order to meet, in an optimal manner, the demands at several service points. The problem often lies in selecting a site that minimises total weighted distances from facilities and customers. Facility location problems form part of an important field of study and are applied in a wide variety of applications areas. Applications range from supply chain problems (e.g. location of plants, warehouses and retail outlets), government agencies (e.g. location of schools, police stations, fire stations and ambulance or healthcare facilities), telecommunications (e.g. placement of cell towers in a network), etc.

The optimisation of the placement of facilities is a well-known and widely studied subject and numerous examples of research conducted in this area exist. Due to this large body of knowledge it would be appropriate to only list a few examples, as opposed to describing the studies in detail. Table 4.1 lists some of the recent studies on facility location problems found in the literature.

Table 4.1 Recent literature examples of facility location problems

Authors	Year	Title
An, H.C., Singh, M. & Svensson, O.	2017	LP-based algorithms for capacitated facility location
Amin, S.H. & Baki, F.	2017	A facility location model for global closed-loop supply chain network design
Correia, I. & Saldanha-da-Gama, F.	2019	Facility location under uncertainty
Fallah Nezhad, M.S., Zarrinpoor, N. & Pishvaee, M.S.	2017	Design of a reliable facility location model for health service networks
Farahani, R.Z., Fallah, S., Ruiz, R., Hosseini, S. & Asgari, N.	2019	OR models in urban service facility location: a critical review of applications and future developments
Fong, C.K.K., Li, M., Lu, P., Todo, T. & Yokoo, M.	2018	Facility location games with fractional preferences
Gendron, B., Khuong, P.V. & Semet, F.	2017	Comparison of formulations for the two-level uncapacitated facility location problem with single assignment constraints

Table 4.1 Continued

Authors	Year	Title
Karatas, M. & Yakıcı, E.	2018	An iterative solution approach to a multi-objective facility location problem
Lynskey, J., Thar, K., Oo, T.Z. & Hong, C.S.	2019	Facility location problem approach for distributed drones
Ortiz-Astorquiza, C., Contreras, I. & Laporte, G.	2018	Multi-level facility location problems
Raghavan, S., Sahin, M. & Salman, F.S.	2019	The capacitated mobile facility location problem
Seker, S. & Aydin, N.	2020	Hydrogen production facility location selection for Black Sea using entropy based TOPSIS under IVPF environment
Shavarani, S.M., Nejad, M.G., Rismanchian, F. & Izbirak, G.	2018	Application of hierarchical facility location problem for optimization of a drone delivery system: a case study of Amazon prime air in the city of San Francisco
Turkoglu, D.C. & Genevois, M.E.	2019	A comparative survey of service facility location problems
Wichapa, N. & Khokhajaikiat, P.	2017	Solving multi-objective facility location problems using the fuzzy analytical hierarchy process and goal programming: a case study on infectious waste disposal centers
Yadav, V., Bhurjee, A.K., Karmakar, S. & Dikshit, A.K.	2017	A facility location model for municipal solid waste management system under uncertain environment
Yu, G., Haskell, W.B. & Liu, Y.	2017	Resilient facility location against the risk of disruptions

The focus of the facility location problem in this study is on healthcare and relief supply facilities in a disaster-stricken area. Sections 4.2.1 and 4.2.2 described examples of relevant literature resources and to conclude the background discussion on existing literature, Table 4.2 presents some additional resources.

Table 4.2 Additional examples of facility location studies related to disaster-stricken areas

Authors	Year	Title
Acar, M. & Kaya, O.	2019	A healthcare network design model with mobile hospitals for disaster preparedness: a case study for Istanbul earthquake
Ahmadi-Javid, A., Seyedi, P. & Syam, S.S.	2017	A survey of healthcare facility location
Boonmee, C., Arimura, M. & Asada, T.	2017	Facility location optimization model for emergency humanitarian logistics
Golabi, M., Shavarani, S.M. & Izbirak, G.	2017	An edge-based stochastic facility location problem in UAV-supported humanitarian relief logistics: a case study of Tehran earthquake
Güneş, E.D., Melo, T. & Nickel, S.	2019	Location problems in health care
Hashim, N.M., Shariff, S. & Deni, S.M.	2017	Capacitated maximal covering location allocation problem during flood disaster
Liu, Y., Cui, N. & Zhang, J.	2019	Integrated temporary facility location and casualty allocation planning for post-disaster humanitarian medical service
Shavarani, S.M.	2019	Multi-level facility location-allocation problem for post-disaster humanitarian relief distribution
Trivedi, A. & Singh, A.	2018	Facility location in humanitarian relief: A review

4.3.1. A taxonomy of facility location problems

Daskin (2011) presents a taxonomy of facility location models which is largely based on assumptions pertaining to the service of demand points and the facilities providing the service. Four main categories are identified.

- *Analytic location models*. In this type of model, facilities can be located anywhere in a predefined service region and demands are distributed in some manner or another (e.g. uniformly) over the region.
- *Continuous location models*. In continuous location models, demands are known *a priori* and occur at discrete sites while candidate facilities can be located anywhere in the region.
- *Network location models.* When demand points and facilities are located on the nodes and links (as opposed to anywhere in a region) of a network, the model is referred to as a network location model.
- *Discrete location models.* In discrete location models, no particular assumptions are made about the demands and facilities and modelling is based on the locations and some arbitrary distance metric. These types of model are mostly formulated as integer programming models.

Figure 4.5 shows a graphical representation of Daskin's taxonomy.

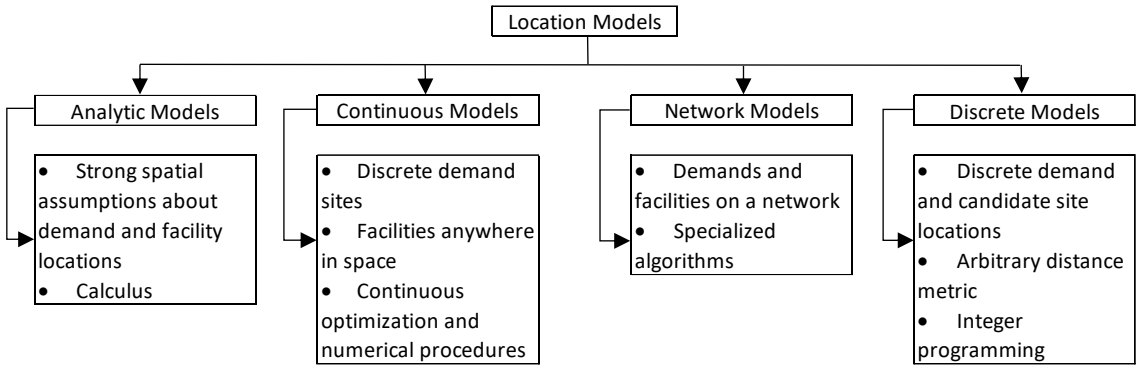


Figure 4.5 Taxonomy of facility location models as proposed by Daskin (2011)

Facility location problems are also characterised by capacitated versus uncapacitated problems. If the capacity of each facility is known (e.g. the capacity of a factory to manufacture a certain number of commodities), the problem is referred to as a capacitated problem – this means that a facility may or may not be able to fully serve the demand at a specific node. If, however, a facility can produce unlimited quantities of a commodity then the problem is known as an uncapacitated problem. In this study, it will be assumed that the healthcare and relief facilities can be supplied from another source and would therefore be treated as uncapacitated.

In addition to the capacity property of facilities, location problems are also generally classified as either discrete (i.e. facilities can only be located at specific candidate sites) or continuous (i.e. facilities can be established anywhere in a region). In this study, the focus will be on discrete facility locations.

Discrete facility location problems are further classified into three broad categories, namely covering-based problems, median-based problems and other problems (Ahmadi-Javid *et al.*, 2017). The model developed in this study is a covering-based model and more specifically, a set covering location model. Figure 4.6 shows the three broad classes of discrete location problems.

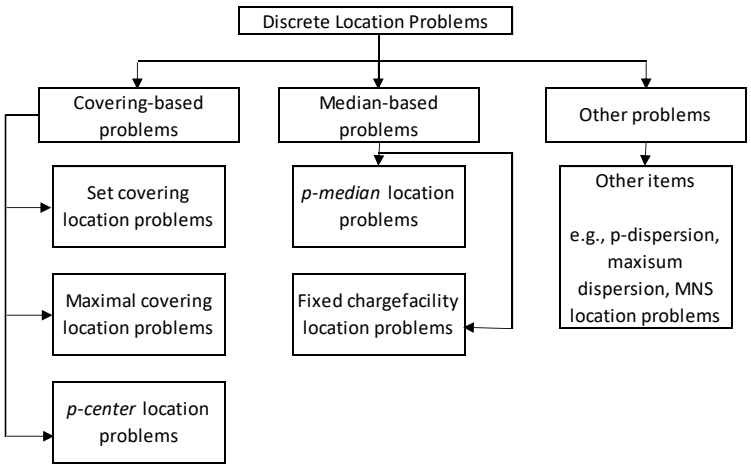


Figure 4.6 Classification of discrete facility location models (Ahmadi-Javid *et al.,* 2017)

The formulations of the five types of covering-based and median-based models depicted in Figure 4.6 will be given in the next section.

4.3.2. Model formulations

Facility locations models, especially discrete models, are mainly formulated as integer linear programming models. These formulations will be provided in the subsequent subsections and will be based on the work and models presented in Ahmadi-Javid *et al*. (2017).

4.3.2.1. Covering-based problems

To ensure that demand points are serviced by facilities (or supply points), the demand locations should be within a specific distance or time from the facilities, i.e. they need to be within a certain coverage distance. According to the classification scheme presented in Figure 4.6, there are three basic types of set covering problem. To formulate these three covering-based models, let $\mathcal{I}$ represent a set of demand points and $\mathcal{J}$ a set of candidate locations. $\mathcal{N}_i$ is the set of all candidate locations which can cover demand point $i \in \mathcal{I}$ and $\mathcal{N}_i = \{j \in \mathcal{J}: d_{ij} \leq D_i\}$ with d_{ij} as the travel distance (or time) from demand point $i \in \mathcal{I}$ to candidate location $j \in \mathcal{J}$ and D_i is the maximum acceptable travel distance (or time) from demand point $i \in \mathcal{I}$ (this is the covering distance or time).

(a) Set covering location problems

The aim of a set covering location problem is to minimise the number of facilities (i.e. minimise the total cost of establishing facilities), while satisfying a pre-specified level of demand coverage.

The input parameter f_j is defined as the fixed cost to establish a facility at candidate location $j \in \mathcal{J}$.

Define the binary decision variable x_j as follows:

$$x_j = \begin{cases} 1 & \text{if a facility is located at candidate site } j \in \mathcal{J} \\ 0 & \text{otherwise} \end{cases}$$

The model formulation is then given by

$$\text{Minimise} \quad \sum_{j \in \mathcal{J}} f_j x_j \tag{4.1}$$

$$\text{subject to} \quad \sum_{j \in N_i} x_j \geq 1 \qquad i \in \mathcal{I}, \tag{4.2}$$

$$x_j \in \{0,1\} \qquad j \in \mathcal{J}. \tag{4.3}$$

The objective function (4.1) minimises the cost of facilities established. Constraint set (4.2) guarantees that each demand point is covered while constraint set (4.3) ensures the binary nature of the decision variables x_j.

(b) Maximal covering location problems

A maximal covering location problem aims at locating p facilities in order to maximise demand covered within a predetermined maximum coverage distance. Due to the maximum of p facilities, some demand points may not be covered, as the model differentiates between small and large demand at different demand points.

Two additional input parameters are defined, w_i (demand at point $i \in \mathcal{I}$) and p (number of candidate locations to be established).

Define two binary decision variables x_j and z_i as follows:

$$x_j = \begin{cases} 1 & \text{if a facility is located at candidate site } j \in \mathcal{J} \\ 0 & \text{otherwise} \end{cases}$$

$$z_i = \begin{cases} 1 & \text{if demand point } i \in \mathcal{I} \text{ is covered} \\ 0 & \text{otherwise} \end{cases}$$

The model formulation is then given by

$$\text{Maximise} \quad \sum_{i \in \mathcal{I}} w_i z_i \tag{4.4}$$

$$\text{subject to} \quad \sum_{j \in \mathcal{J}} x_j = p, \tag{4.5}$$

$$z_i \leq \sum_{j \in N_i} x_j \qquad i \in \mathcal{I}, \tag{4.6}$$

$$z_i \in \{0,1\} \qquad i \in \mathcal{I}, \tag{4.7}$$

$$x_j \in \{0,1\} \qquad j \in \mathcal{J}. \tag{4.8}$$

In the above model, the objective function (4.4) maximises the total covered demand. The constraint (4.5) ensures that p facilities are established while constraint set (4.6) guarantees that a demand point is covered by an open facility. The two constraint sets (4.7) and (4.8) define the binary decision variables.

(c) P-centre location problems

The p-centre location problem is also called a minimax location problem, as it aims to minimise the maximum distance between a demand point and the nearest facility. Daskin (2011) defines it as "to minimise the coverage distance such that each demand node is covered within the endogenously determined distance by one of the facilities".

As with the maximal covering problem, two additional input parameters are defined; w_i (demand at point $i \in \mathcal{I}$) and p (number of candidate locations to be established).

Define two binary decision variables x_j and y_{ij} as follows:

$$x_j = \begin{cases} 1 & \text{if a facility is located at candidate site } j \in \mathcal{J} \\ 0 & \text{otherwise} \end{cases}$$

$$y_{ij} = \begin{cases} 1 & \text{if demand point } i \in \mathcal{I} \text{ is assigned to facility } j \in N_i \\ 0 & \text{otherwise} \end{cases}$$

The model formulation is then given by

$$\text{Minimise} \quad M \tag{4.9}$$

$$\text{subject to} \quad \sum_{j\in\mathcal{J}} y_{ij} = 1 \qquad i \in \mathcal{I}, \tag{4.10}$$

$$\sum_{j\in\mathcal{J}} x_j = p, \tag{4.11}$$

$$\sum_{j\in\mathcal{J}} d_{ij}y_{ij} \leq M \qquad i \in \mathcal{I}, \tag{4.12}$$

$$y_{ij} \leq x_j \qquad i \in \mathcal{I}, j \in \mathcal{J}, \tag{4.13}$$

$$y_{ij} \in \{0,1\} \qquad i \in \mathcal{I}, j \in \mathcal{J}, \tag{4.14}$$

$$x_j \in \{0,1\} \qquad j \in \mathcal{J}, \tag{4.15}$$

$$M \geq 0. \tag{4.16}$$

The objective function (4.9) minimises the maximum distance between a demand point and the nearest facility to that demand point. Constraint set (4.10) ensures that each demand point is covered by only one facility, i.e. demand at node i must be assigned to facility at node j while constraint (4.11) ensures that p facilities are established. Constraints (4.12) specify the maximum demand-weighted distance (or time) and constraints (4.13) guarantee that a demand node cannot be assigned to a facility that is not open. Finally, the constraints (4.14) – (4.16) are the integrality and non-negativity constraints.

4.3.2.2. Median-based problems

In order to minimise the weighted average distance costs between demand points and facilities to which demand points are assigned, median-based problems establish facilities at predetermined candidate sites. In this section, and according to the classification scheme in Figure 4.6, two of the popular median-based problems will be presented. As in the previous sections, let $\mathcal{I}$ represent a set of demand points and $\mathcal{J}$ a set of candidate locations. Let d_{ij} be the distance (or time) to travel from demand point $i \in \mathcal{I}$ to candidate location $j \in \mathcal{J}$ and let w_i represents the demand at point $i \in \mathcal{I}$.

(a) P-median location problems

In a p-median problem, the objective is to find the location of p facilities in a network so that the total cost is minimised. The cost to serve a demand point is given as the product of the demand (w_i) and the distance (d_{ij}) between the demand point and the closest facility to that demand point.

An additional input parameter p is defined as the number of candidate locations to be established.

Define two binary decision variables x_j and y_{ij} as follows:

$$x_j = \begin{cases} 1 & \text{if a facility is located at candidate site } j \in \mathcal{J} \\ 0 & \text{otherwise} \end{cases}$$

$$y_{ij} = \begin{cases} 1 & \text{if demand point } i \in \mathcal{I} \text{ is assigned to facility } j \in \mathcal{J} \\ 0 & \text{otherwise} \end{cases}$$

The model formulation is then given by

$$\text{Minimise} \quad \sum_{i\in\mathcal{I}} \sum_{j\in\mathcal{J}} w_i d_{ij} y_{ij} \tag{4.17}$$

$$\text{subject to} \quad \sum_{j\in\mathcal{J}} y_{ij} = 1 \qquad i \in \mathcal{I}, \tag{4.18}$$

$$\sum_{j\in\mathcal{J}} x_j = p, \tag{4.19}$$

$$y_{ij} \leq x_j \qquad i \in \mathcal{I}, j \in \mathcal{J}, \tag{4.20}$$

$$y_{ij} \in \{0,1\} \qquad i \in \mathcal{I}, j \in \mathcal{J}, \tag{4.21}$$

$$x_j \in \{0,1\} \qquad j \in \mathcal{J}. \tag{4.22}$$

The objective function (4.17) of the p-median model minimises the total demand-weighted travel distance (or time) between each demand point and the nearest facility. Constraint set (4.18) ensures that each demand point is covered by only one facility, i.e. demand at node i must be assigned to facility at node j while constraint (4.19) ensures that p facilities are established. Constraints (4.20) specify that a demand node cannot be assigned to a facility that is not open. The constraint sets (4.21) and (4.22) are the integrality constraints for the decision variables x_j and y_{ij}.

(b) Fixed charge facility location problems

The final model in median-based problems is called a fixed charge facility location problem. In these types of problem, an explicit cost of locating a facility at a candidate site is added and the model then aims to minimise the total cost of travelling and opening a facility. A basic uncapacitated fixed charge facility location problem is described in this paragraph.

Two additional input parameters are defined: f_j as the fixed cost (charge) to locate a facility at candidate site $j \in \mathcal{J}$, and v as the (variable) transportation cost per item per distance unit.

Define two binary decision variables x_j and y_{ij} as follows:

$$x_j = \begin{cases} 1 & \text{if a facility is located at candidate site } j \in \mathcal{J} \\ 0 & \text{otherwise} \end{cases}$$

$$y_{ij} = \begin{cases} 1 & \text{if demand point } i \in \mathcal{I} \text{ is assigned to facility } j \in \mathcal{J} \\ 0 & \text{otherwise} \end{cases}$$

The model formulation is then given by

$$\text{Minimise} \quad \sum_{j\in\mathcal{J}} f_j x_j + v \sum_{i\in\mathcal{I}} \sum_{j\in\mathcal{J}} w_i d_{ij} y_{ij} \tag{4.23}$$

$$\text{subject to} \quad \sum_{j\in\mathcal{J}} y_{ij} = 1 \qquad i \in \mathcal{I}, \tag{4.24}$$

$$y_{ij} \leq x_j \qquad i \in \mathcal{I}, j \in \mathcal{J}, \tag{4.25}$$

$$y_{ij} \in \{0,1\} \qquad i \in \mathcal{I}, j \in \mathcal{J}, \tag{4.26}$$

$$x_j \in \{0,1\} \qquad j \in \mathcal{J}. \tag{4.27}$$

The objective function (4.23) minimises the total cost (cost to open a facility and transportation costs). Constraint set (4.24) ensures that each demand point is covered by only one facility while constraint set (4.25) specifies that a demand node cannot be assigned to a facility that is not open. Constraints (4.26) and (4.27) are the integrality constraints that define the binary nature of decision variables x_j and y_{ij}.

In the event of a capacitated fixed charge facility location problem, an additional parameter C_j to define the maximum capacity of each facility $j \in \mathcal{J}$ is needed. The following additional capacity constraint is then added to model (4.23) – (4.27).

$$\sum_{i\in\mathcal{I}} w_i y_{ij} \leq C_j, \quad j \in \mathcal{J} \tag{4.28}$$

Sections 4.3.2.1 and 4.3.2.2 present the model formulations for the most popular and basic models found in the covering-based and median-based facility location problem categories (see Figure 4.6). There is a large number of variants of these models which will not be covered here. Interested readers are referred to the authoritative work by Daskin (2011) on network and discrete location models for an in-depth discussion of the different facility

location models and all their extensions. For facility location problems that are focussed specifically on healthcare related problems, the work of Ahmadi-Javid *et al.* (2017) offers an excellent overview.

4.3 Summary

The aim of this chapter was to introduce facility location problems which are necessary for the development of a rescue facility location model in a disaster-stricken area that will be developed and discussed in subsequent chapters. Brief background information on the importance of healthcare and relief supply facilities was presented, followed by a discussion on facility location models in general. This discussion included a taxonomy and classification of facility location models, as well as detailed formulations for the basic models. Discussions were backed up with examples from the literature.

Chapter 5 Grid-based maze model development: a real-world case study

5.1. Introduction

The main objective of this research study is to design a grid-based maze generator that can be used to represent traversable routes in a disaster-stricken area. Background information on how to generate a grid-based maze (Chapter 2) and solution strategies to solve a grid-based maze (Chapter 3) were presented earlier. In this chapter, a computerised demonstrator is developed to illustrate how the various algorithms can assist humanitarian logistics in relief efforts by determining optimal paths for rescue efforts. A real-world disaster will be used to illustrate the computerised demonstrator and the software developed. In the chapter, a description of the real-world disaster and the data used to generate a grid-based maze will be presented. This will be followed by a description of a matrix-based maze that was constructed in combination with Kruskal's maze generation algorithm. Finally, the implementation of two solution strategies (Lee algorithm and A-star algorithm) will be presented and the results discussed.

5.2. The real-world disaster

The real-world disaster that will be used in this study is Hurricane Katrina that hit New Orleans in late August 2005. New Orleans is a city with approximately 494 000 residents and is situated along the Mississippi River in the state of Louisiana in the United States of America (DeWaard *et al.,* 2016). Figure 5.1 presents a map of New Orleans.

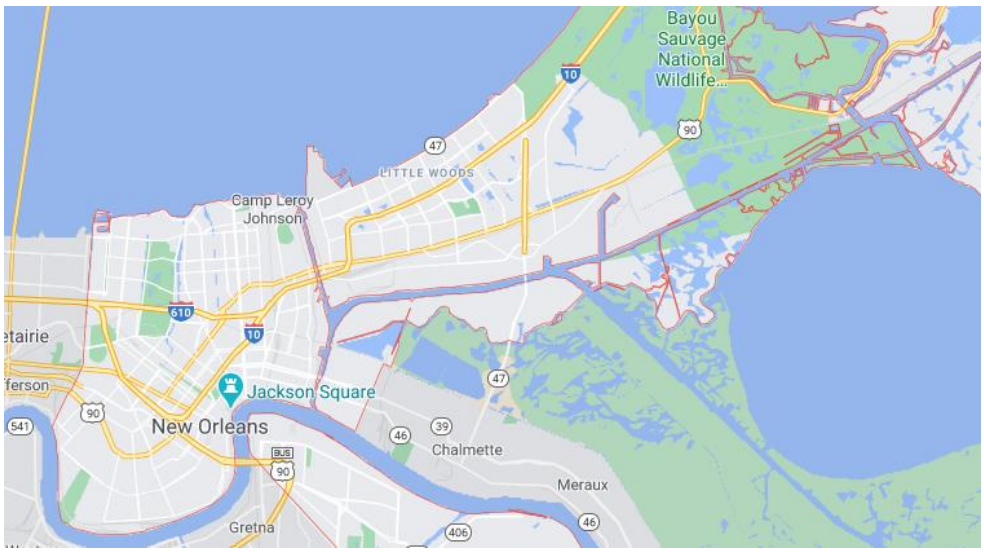


Figure 5.1 New Orleans *Image credit: Google Maps*

Hurricane Katrina was a category 5 cyclone that originated on 23 August 2005 over the Bahamas. On 29 August 2005, the hurricane hit New Orleans, costing 1 388 residents their lives and amassing $108 million in property damage (Sydnor *et al.*, 2017). Figure 5.2 depicts a weather photo of the storm, while Figure 5.3 shows part of the damage that was caused by the hurricane.

Figure 5.2 Hurricane Katrina *(Image: © GOES Project Science Office)*

Figure 5.3 Hurricane Katrina damage *(Image credit: NWS/Lieut. Commander Mark Moran, NOAA Corps, NMAO/AOC)*

A large variety of data, facts and other information was gathered by different organisations and institutions. To be able to use the real data in a maze, it was necessary to find data that accurately depicted the damage caused to the area, especially with regard to location and severity. The data used in this study was gathered and interpreted by the LSU Katrina Survey Team Department of Sociology[1], and consists of post-damage data that accurately shows the severity of the damage in the different areas of the city. This data, which is shown in Figure 5.4, is especially useful for this study, as it enables the generation of a matrix maze. The different coloured dots in Figure 5.4 indicate the severity of the destruction at each area. Green dots indicate that there is no damage at all, while the yellow dots indicate signs of damage, but not complete destruction. Areas marked with orange and red dots are the areas where major damage to structures occurred. This damage varies from partially destroyed buildings to completely demolished buildings and roads. In the context of road or path selection, the green and yellow dots represent accessible areas (possible paths in a maze) while the orange and red dots represent inaccessible areas (walls in a maze).

[1] https://www.lsu.edu/faculty/fweil/KatrinaMaps/index.htm.

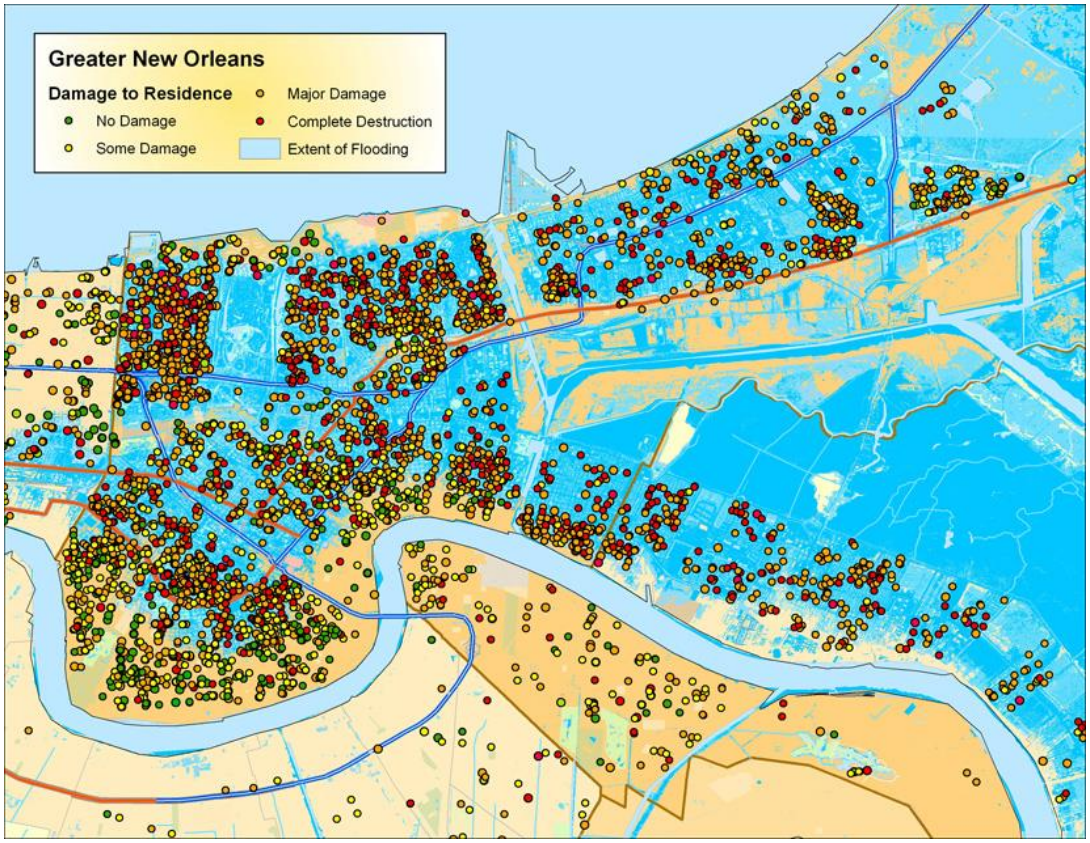


Figure 5.4 Residential damage to Greater New Orleans

In the next section, the data, as depicted in Figure 5.4, will be utilised to develop a maze that will ultimately be used to determine optimal paths in the area.

5.3. Greater New Orleans maze generation

In this section, a maze will be generated to fit a map of the residential damage to the Greater New Orleans region, as shown in Figure 5.4 in the previous section. This maze will then ultimately be used to determine the best possible traversable routes that may be used by humanitarian relief personnel. The maze must accurately depict the inaccessible areas in the disaster region by “constructing walls” according to one of the maze generation algorithms presented in Chapter 2. These maze generation algorithms are based on walls that are randomly constructed; however, in the New Orleans real-world situation walls cannot be erected at random, but should be placed according to the limited possibilities, as dictated by the damage in the area. To provide for this limitation, a matrix is built, based on the available data of the damaged areas. The matrix is then utilised by one of the maze generation algorithms to generate a final maze structure representing the total disaster-stricken area under consideration. The steps (which are described in the remainder of this section) can be summarised as follows:

- Grid placement;
- Develop a matrix to represent the data;
- Generate a maze from the matrix;
- Use a maze generation algorithm (Kruskal's algorithm) to complete the maze; and
- Fit the final maze over the disaster-stricken area under consideration.

5.3.1. Grid placement according to the Hurricane Katrina damage data

In order to generate the required matrix from the data, a grid was placed over the map (presented in Figure 5.4) of New Orleans that contains the residential damage data. The grid size was chosen through experimenting with different sizes. It was decided to use a grid size of 70x70 – this grid size provides for enough detail in each cell to construct a maze and to produce a representative matrix of the disaster region. The map, together with the grid, are presented in Figure 5.5. It should be noted that the grid size in Figure 5.5. is not 70x70, but has been adjusted for visual and presentation purposes.

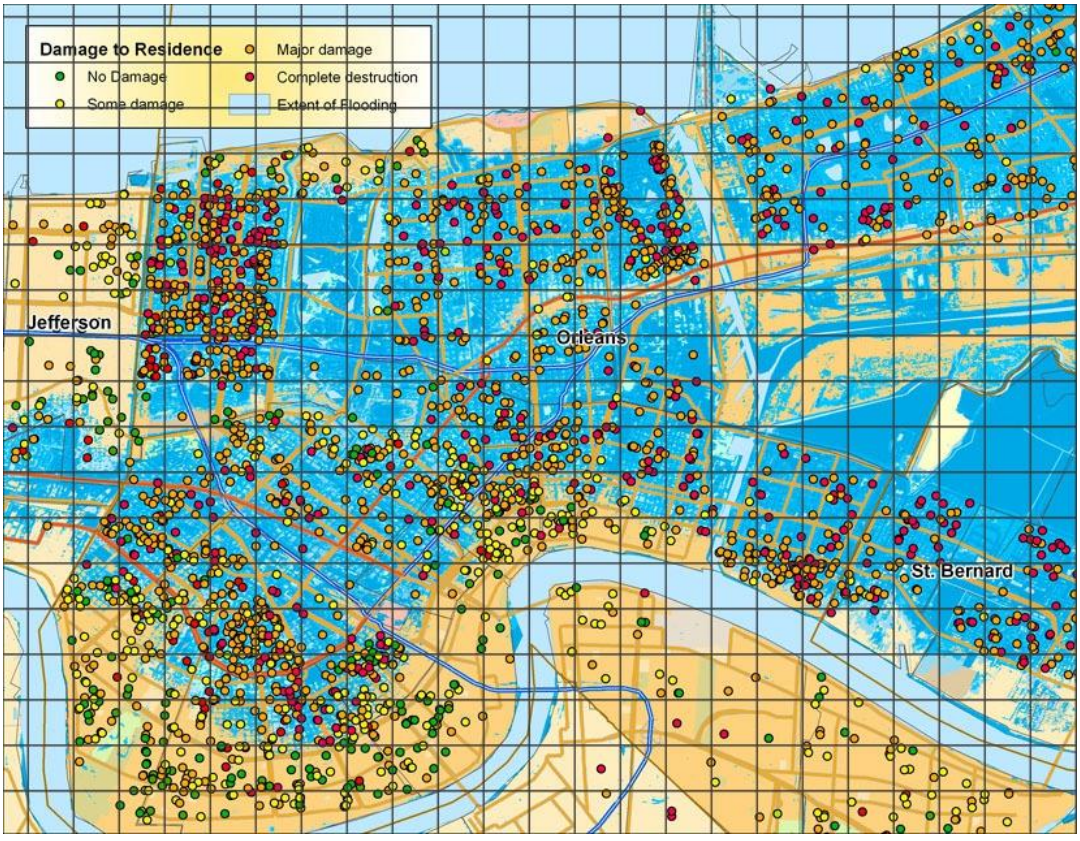


Figure 5.5 Residential damage to Greater New Orleans with grid overlay

The grid developed in this step will be used (in the next section) to construct a matrix to represent the data of the damage in the New Orleans area.

5.3.2. Matrix development to represent the data

To automatically develop a matrix, using the grid map in Figure 5.5, requires image recognition software that can identify the different colour-coded damage points on the map. For this study, the use of such specialised software is not considered, and the matrix is developed manually. The matrix is constructed as follows: Different values are assigned to different combinations of walls per cell, e.g. a 1 in the matrix would indicate the existence of a wall to the left of a cell. The value assignment system used is detailed in Table 5.1.

Table 5.1 Matrix value assignment

Matrix value	Description	Graphical representation	
		Matrix	Wall
0	No walls in the cell	{0}	
1	Wall to the left of the cell	{1}	
2	Wall at the top of the cell	{2}	
3	Wall to the left and the top of the cell	{3}	
4	Inaccessible area	{4}	
5	Relief facilities	{5}	

It should be noted that the values 0 – 3 in Table 5.1 cater for all combinations of walls. For example, should a wall be required to the right of a cell, a value of 1 (wall to the left) can be inserted in the next adjacent cell. Similarly, other combinations can also be addressed by the existing numbers.

Figure 5.6 shows an extract of the development of the matrix. The grid map of the disaster-stricken area (on the left of Figure 5.6) is used to construct the matrix which in turn is used to develop a maze of the area (on the right of Figure 5.6). The maze development will be elaborated on in the next sub-section.

In Figure 5.6, each matrix entry corresponds to a cell in the grid map on the left of the figure. The complete matrix for the 70x70 grid is presented in Annexure A.

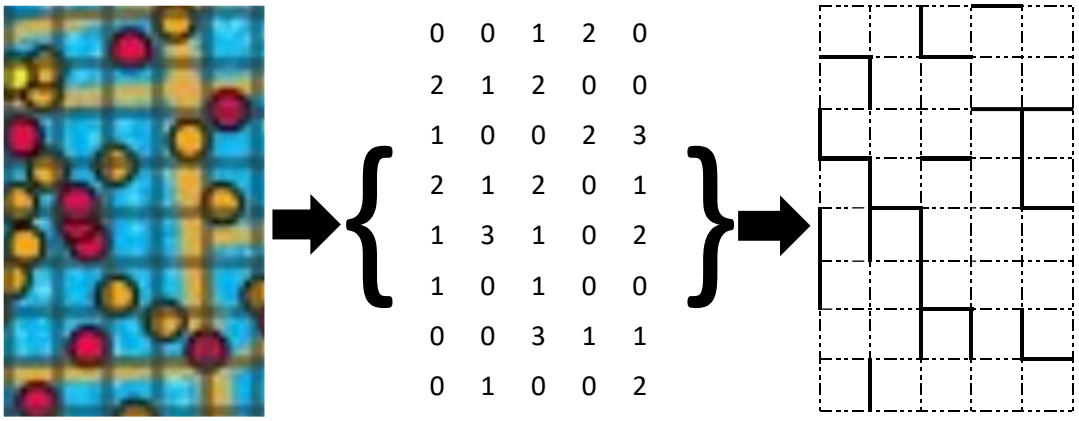


Figure 5.6 Example of the matrix maze generation process

In the next section, a discussion on how the matrix is used to generate a maze is presented.

5.3.3. Generating a maze from the matrix

Most of the discussion in the previous section has already explained how walls for a maze are generated from the matrix that represents the data of the disaster-stricken area. Figure 5.6 (also in the previous section) graphically illustrates the process.

The 70x70 matrix that was constructed is now used to generate a maze structure that represents the entire New Orleans disaster region. A software solution to perform the maze generation from the matrix was created, using Visual Studio and the C# programming language. The algorithm to produce the grid-based maze is detailed in Algorithm 5.1, while the main functional code can be found in Annexure B. The result is a grid-based maze structure that represents the possible traversable routes and the damage caused by Hurricane Katrina in the New Orleans region. Figure 5.7 shows the final result; the grid-based maze for the New Orleans disaster-stricken area that was generated from the 70x70 matrix.

The green squares and numbered blue blocks were inserted for illustrative purposes and are used in the final optimal path generation. The green squares were chosen randomly and represent locations where disaster victims are trapped, i.e. starting points for the maze-solving algorithms that are discussed in the subsequent sections. Similarly, the blue numbered blocks represent medical or relief stations, i.e. possible end points for the maze solving algorithms. In this example (Figure 5.7), five starting locations and six medical or relief stations were given.

Algorithm 5.1: Matrix maze generation

Input: *MAZE* - 2D array of maze matrix
gridPen - used to draw the maze walls
gridBrush - fills inaccessible areas of the map (e.g. river and sea)
HospitalBrush - used to indicate locations of hospitals
Output: g - graphics of completely generated maze
Process: Generating a maze using a matrix

```
1.  For each value in MAZE
2.      if value = 1
3.          g.draw left wall using gridPen
4.      else if value = 2
5.          g.draw top wall using gridPen
6.      else if value = 3
7.          g.draw left wall using gridPen
8.          g.draw top wall using gridPen
9.      else if value = 4
10.         g.fill cell with gridBrush //inaccessible area
11.     else if value = 5
12.         g.fill cell with hospitalBrush //relief facility
13.     end if
14. end for
```

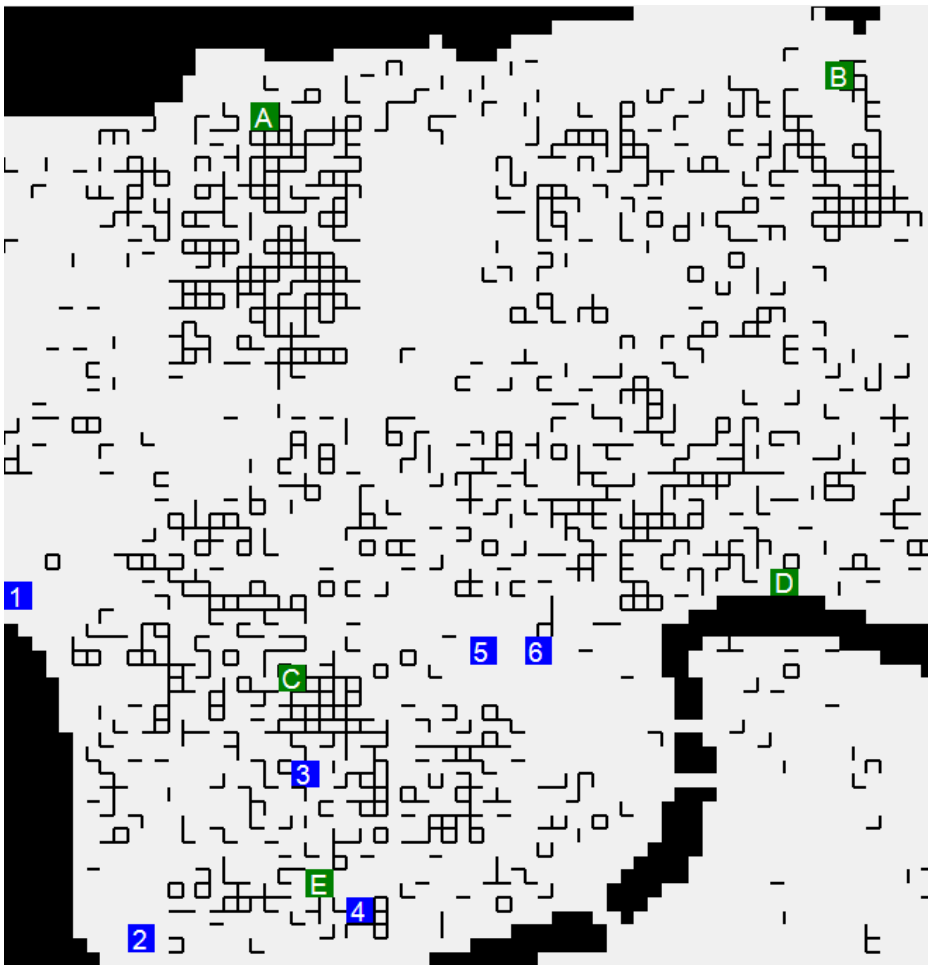


Figure 5.7 Maze structure generated from New Orleans matrix

The generated maze in Figure 5.7 shows the maze walls associated with the 70x70 matrix entries. In the next section, a maze generation algorithm will be employed to complete and fill the gaps in the maze.

5.3.4. Kruskal's maze generation algorithm to complete the maze

The maze structure generated in the previous section (Figure 5.7) contains a large number of empty cells that were not catered for by the matrix wall generation approach. To complete the maze and to eliminate the empty cells, any one of the maze generation algorithms discussed in Chapter 2 can be used to generate a complete maze. It may be argued that it is unnecessary to complete the maze, as the open cells may already indicate areas that are traversable. However, the aim is to ultimately show how maze-solving algorithms can be employed in disaster situations and for these algorithms complete mazes are required. It was decided to use Kruskal's algorithm to generate the complete maze. The algorithm was detailed in Chapter 2 (see Algorithm 2.2). Although this algorithm ensures that a complete maze is generated, it cannot guarantee, in this specific case, that an optimal path through the maze exists. This is due to the fact that the walls already generated from the disaster matrix are fixed and cannot be changed. To avoid dead ends in the maze, the algorithm was adapted to keep the neighbouring cells (of the matrix generated maze) empty. This ensures that there are open paths between the matrix generated maze and the maze generated by Kruskal's algorithm. The pseudocode of the modified Kruskal algorithm is detailed in Algorithm 5.2, while the code used to implement the algorithm is presented in Annexure B.

Algorithm 5.2: Modified Kruskal maze generation algorithm

Input: S - Set of a collection of sets containing cells to explore
X - Set of cells in Matrix Maze
Output: M – Maze is generated
Process: *Generating a perfect maze from a set of cells by Kruskal's algorithm*

```
1.  declare e : edge and c1, c2 : cells
2.  Select a random edge e = (c1, c2) ∈ S
3.  M ← {( c1, c2)}
4.  while number of sets in S >1 do  //all cells do not belong to the same set
5.      Select a random edge e = (c1, c2) ∈ S with c1 and c2 in different sets
6.      If e ∈ X //modified statement
7.          M ← M ∪{empty cell} //ensure no dead-ends occur
8.      esle
9.          M ← M ∪{( c1, c2)}
10.     end if
11.     unify c1 and c2 in S into a single set
12. end while
```

The final and complete maze is shown in Figure 5.8 which is the final maze for the disaster-stricken area that can now be utilised to find optimal paths to relief centres.

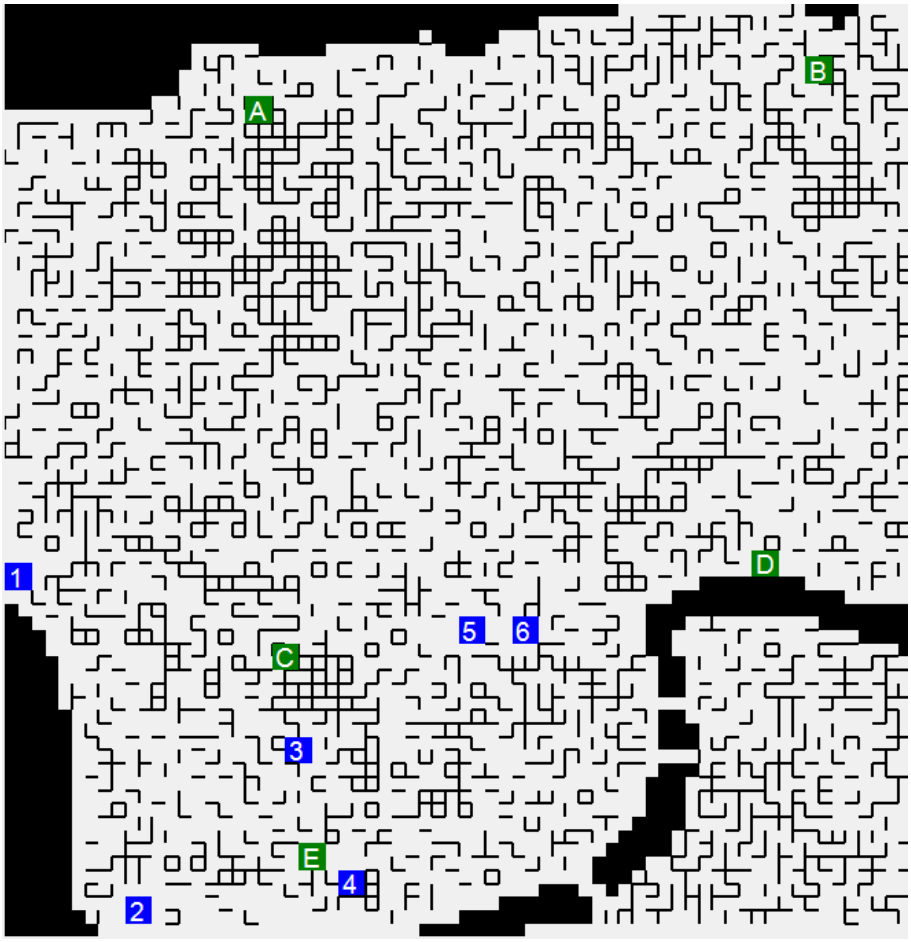


Figure 5.8 Complete maze of the New Orleans disaster-stricken area

5.4. Humanitarian logistics solutions – a software implementation

A software solution was developed to illustrate the use of maze-solving algorithms and to show how humanitarian logistics can benefit from a system that is capable of finding the best traversable path in a disaster-stricken area. In this section, the software implementation will be introduced, and two algorithms (Lee algorithm and A-star algorithm) will be used to solve the New Orleans disaster-stricken area maze generated in the previous sections.

5.4.1. The software solution

The software solution was created using Visual Studio and the C# programming language. Once the maze for the disaster-stricken area has been constructed, as detailed in all the previous sections, the maze is displayed. Several options are then available to the user and when the appropriate options are selected, the maze will be solved, using either the Lee algorithm or the A-star algorithm, depending on the options selected. The final route is then mapped out on the grid-based maze and can then be used by relief workers.

Figure 5.9 shows the user interface and all the options that have to be selected (the options are discussed below the figure).

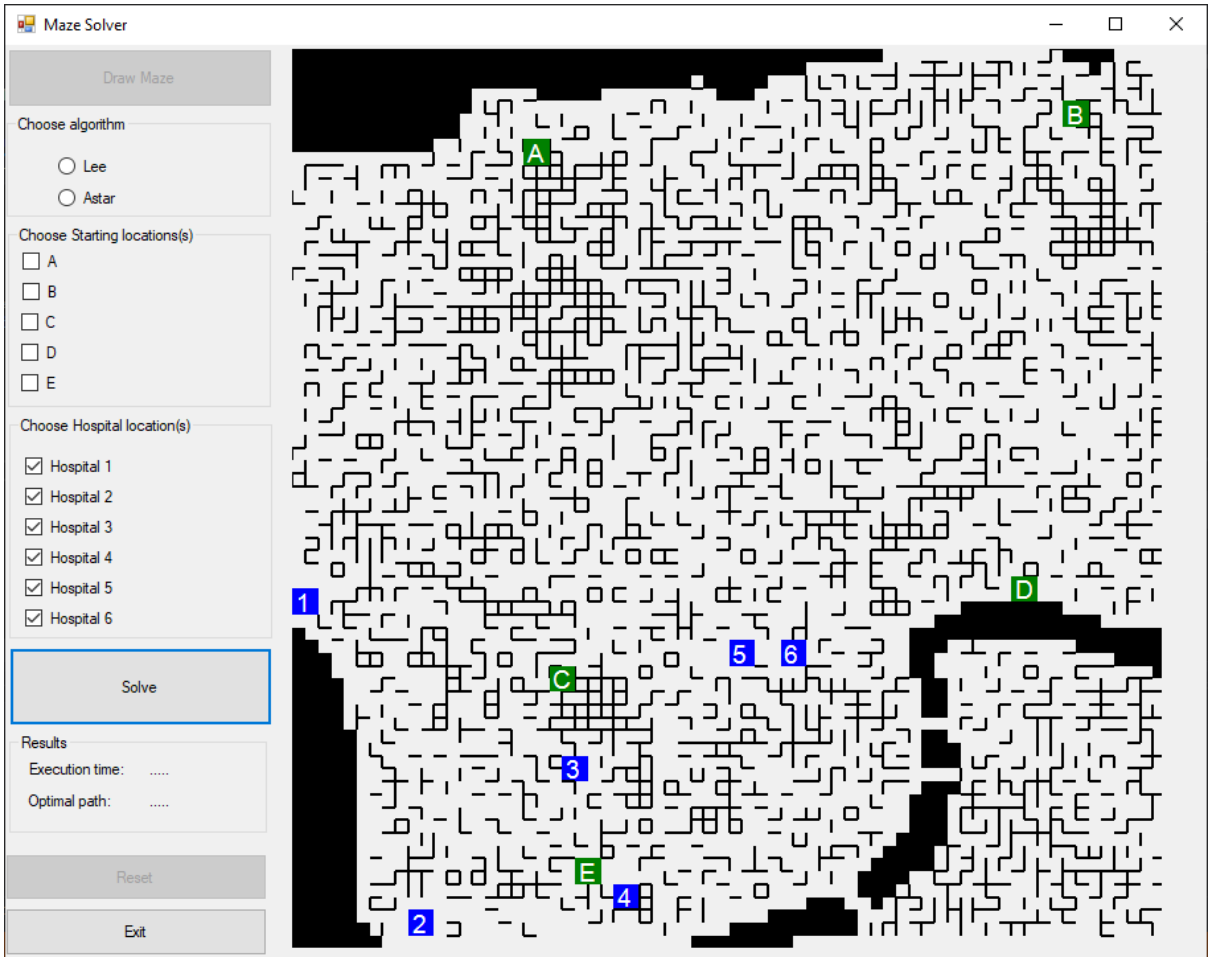


Figure 5.9 Maze solver user interface

On the right-hand side of the user interface in Figure 5.9, the grid-based maze is displayed. For illustration purposes, the five green blocks (labelled A to E) indicate where disaster victims are located (starting points) and the six numbered blue blocks represent relief stations (e.g. medical and supplies) or end points to where victims can be evacuated. These starting and end points can be changed as necessary or as the situation requires. On the left of the user interface are the options that the user must select.

The following options are available:

- Choose algorithm – the user is presented with two maze-solving algorithms, namely the Lee algorithm and the A-star algorithm. Both these algorithms will solve the maze and produce an optimal traversable path and it does not really matter which one is chosen. The choice is for illustrative purposes and to show that both algorithms are applicable.

- Choose starting location – These are the possible starting points (green blocks) for the algorithm to solve the maze and in practice represent potential locations where victims are positioned. In Figure 5.9, five possible locations are listed. This list of options will change depending on where the victims are. The user must select a starting point.
- Choose hospital locations – The hospital locations represent hospitals or relief stations and are the possible end points for solving the maze. Selecting hospitals indicates that the hospitals are operational and can be considered as an end point by the maze-solving algorithm. If left unchecked, the algorithm will ignore the hospital (assuming that it is inaccessible) and attempt to reach a different operational hospital. The hospitals are indicated by the numbered blue squares and may also change, depending on where hospitals are or where temporary hospitals or relief stations are established.
- Solve button – Pressing this button will execute the selected algorithm to solve and determine the optimal route from the selected starting point to the closest operational hospital.
- Results – Indicates the execution time of the algorithm and the distance of the optimal path in terms of the total movements through the maze.
- Reset – Resets the program to its original state.
- Exit – Closes the program.

The following two sections (5.4.2 and 5.4.3), solving the maze by using the two different algorithms will be presented.

5.4.2. The Lee algorithm

The Lee algorithm is the first algorithm illustrated with the real-world data. The algorithm was detailed in Chapter 3 and the pseudocode was presented in Algorithm 3.1 in the same chapter. The main code to implement the algorithm is presented in Annexure B. In this example, the starting point was selected as the location at the top right-hand side (B) of the maze. All six hospitals or relief stations were selected, indicating that anyone of them can be used as an end or evacuation point. Figure 5.10 shows the result of the Lee algorithm. The blue area indicates how the search evolved, as the algorithm explores neighbouring cells. The final path, indicated in red and obtained through a backtracking technique, shows the route to the nearest hospital (number 6).

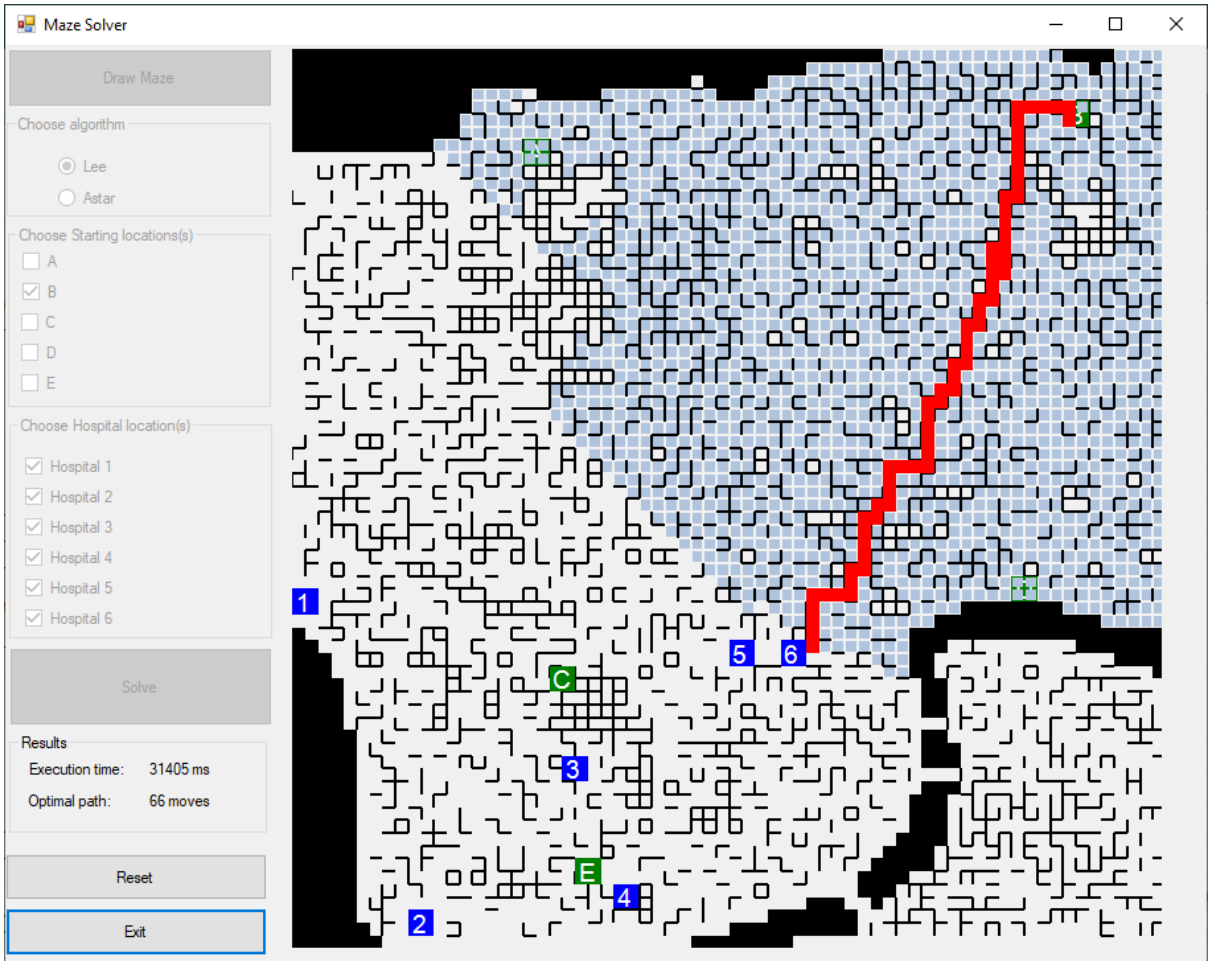


Figure 5.10 Optimal path generated using the Lee algorithm

Figure 5.10 also indicates the execution time and the number of moves necessary to reach the destination. The algorithm solved the maze in 31 405 milliseconds and required 66 moves.

5.4.3. The A-star algorithm

To demonstrate the versatility of maze-solving algorithms, the A-star algorithm is also included as an option in the software. The A-star algorithm selects each cell to explore, based on a heuristic distance from the goal. This means that the algorithm does not blindly search until the goal is reached, but rather directs its search efforts in the direction of the goal, reducing the search space. The pseudocode (Algorithm 3.2), as well as a detailed explanation of the A-Star algorithm were presented in Chapter 3.

Figure 5.11 is a representation of the A-star algorithm applied to the New Orleans disaster-stricken area grid-based maze. In this example, the starting point is again selected as the location at the top right-hand side of the maze. Only one relief station (hospital) is being indicated as active (hospital 1) which will force the algorithm to find the optimal path from the

starting point to this specific hospital. The blue cells indicate the search space, based on how each cell is explored, using a heuristic distance (see Chapter 3 for an explanation of how the cells are selected) and the red path is then the final optimal path generated by the algorithm.

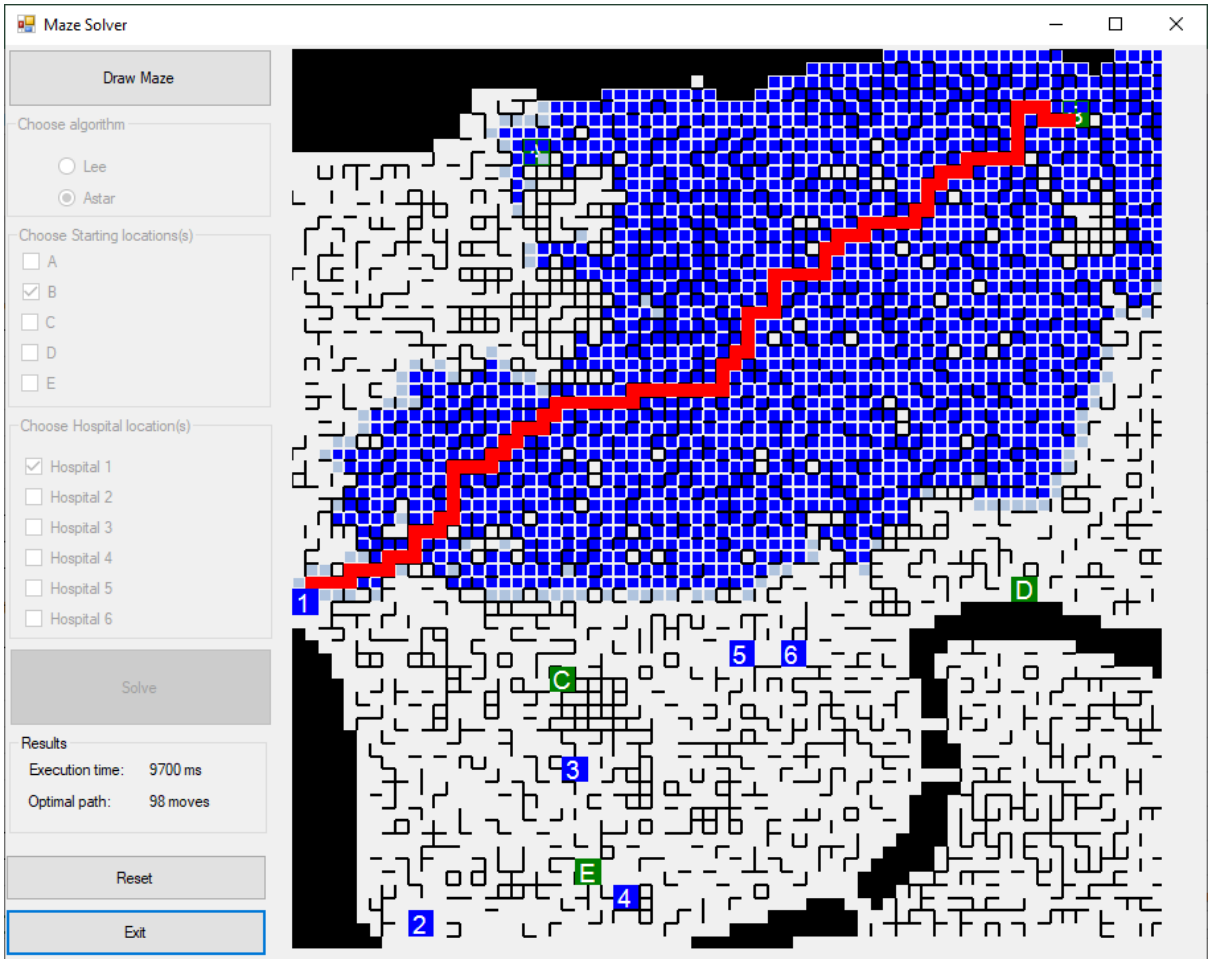


Figure 5.11 Optimal path generated using the A-star algorithm

The execution time and number of moves necessary to reach the destination are shown on the left-hand side of Figure 5.11. The A-star algorithm solved the maze in 9 700 milliseconds and required 98 moves. The code to generate the final solution is presented in Annexure B.

In the following section, a more detailed discussion of the results of the two maze-solving algorithms will be presented.

5.5. Discussion of results

In this section, a general discussion of the results of the grid-based maze software solution applied to the real-world case study is presented. The discussion focusses on aspects, such

as the choice of algorithms (for both generating and solving a maze) as well as technical algorithmic and software development challenges.

5.5.1. Choice of algorithms

There are a number of algorithms that can be utilised to generate and solve a maze. Some of the more popular algorithms are presented and explained in Chapter 2 (to generate a maze) and Chapter 3 (to solve a maze). In this study, specific algorithms are selected to demonstrate how a grid-based maze structure and the associated algorithms can be used to assist in humanitarian logistics.

The maze-generator algorithm

The *Kruskal* algorithm was the preferred choice of a maze-generator algorithm in this study. This choice is motivated as follows.

As described in Section 5.3, a matrix approach for the disaster-stricken areas was used to generate a maze. However, this maze contains a large number of empty cells that were not catered for and to complete the maze it was necessary to employ one of the available maze generation algorithms. The Prim and Kruskal algorithms were the two top choices for generating a maze that can be used, together with the matrix generated maze, to construct the final grid-based maze. As explained in Chapter 2, Prim's algorithm starts with a single randomly selected cell and then explores in different directions to place the walls. This implies that the process of generating the maze has one randomly selected cell as a starting point and that the maze will be generated and completed with this one cell as starting point. Combining Prim's algorithm with the matrix generated maze is unfeasible, as the walls already constructed by the matrix approach (which may not be changed) block the algorithm's path and leave a number of cells unexplored. As opposed to this, the randomisation of a starting location with each iteration in Kruskal's algorithm provides for a method to explore each cell, regardless of the positioning of walls generated by the matrix approach. This suggests that the matrix generated maze and the result of Kruskal's algorithm can be combined to form one complete maze with no empty and unexplored cells. To avoid dead ends, the Kruskal algorithm was adapted to keep neighbouring cells (of the matrix generated maze) open.

The maze-solving algorithm

To illustrate how the maze can be solved to find an optimal path that may be used by relief teams in a disaster-stricken area, two algorithms were implemented in the software solution,

i.e. the *Lee* algorithm and the *A-star* algorithm. The intention of this discussion is not to choose between the two algorithms, but rather to demonstrate the working of each and then, based on the actual situation, leave it to the user to decide which algorithm to employ to find an optimum path. The discussion focuses on two aspects (path length and execution time) and then concludes with some general remarks regarding the two algorithms.

In disaster situations, the length of a path plays a critical role in reaching or evacuating victims or transporting relief items, such as medical supplies and food to the victims. It is important to determine the shortest traversable path in these situations. To determine if there is a significant difference in path lengths between the two algorithms, all possible optimum traversable paths from all starting points (A-E) to all end points (1-6) were generated. This resulted in 30 paths for each algorithm. Path lengths are measured in the number of moves that an algorithm has to perform to reach an end point from a specific starting point. The results are summarised in Table 5.2 and graphically displayed in Figure 5.12. It is clear from both the table and the graph that there are no significant differences in path lengths between the algorithms. The two algorithms also identified the same facility (end point) as the closest facility to each starting point, i.e. A to 1, B to 6, C to 3, D to 6 and E to 4.

Table 5.2 Path lengths generated by the Lee and A-star algorithms

	LEE Algorithm						
From Starting Point	**To Hospital**						
	1	**2**	**3**	**4**	**5**	**6**	**To the closest Hospital**
A	55	87	73	80	60	64	55
B	98	114	97	99	70	66	66
C	31	29	13	26	22	29	13
D	68	72	55	57	30	22	22
E	42	17	8	4	27	31	4

	A-STAR Algorithm						
From Starting Point	**To Hospital**						
	1	**2**	**3**	**4**	**5**	**6**	**To the closest Hospital**
A	54	87	72	77	60	64	54
B	98	114	97	96	69	66	66
C	31	29	12	26	22	29	12
D	68	72	55	54	30	22	22
E	42	17	8	4	27	31	4

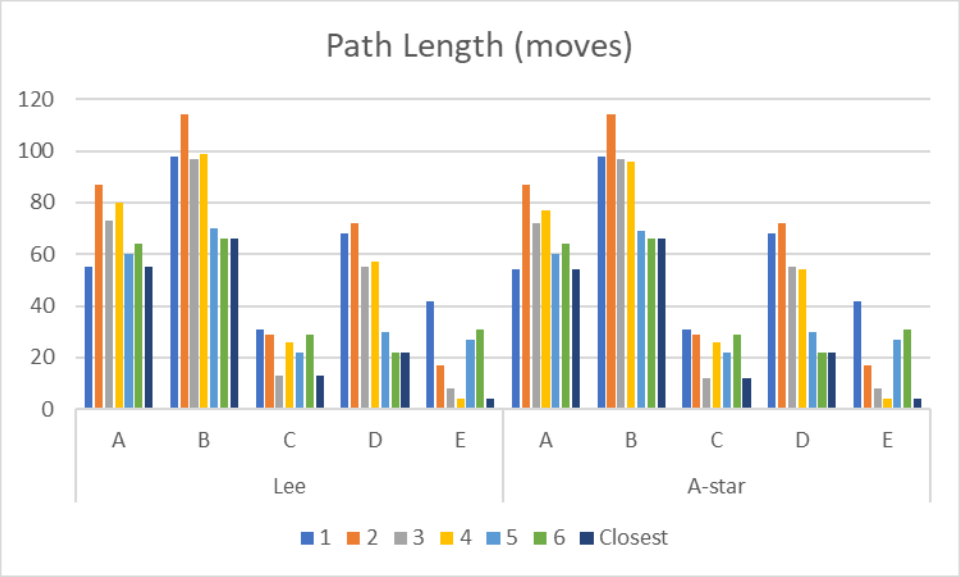


Figure 5.12 Graphic analysis of the path length for the Lee and A-star algorithms

An important consideration in decision support systems, such as the software solution developed in this study, is computational complexity. It is often necessary to find an acceptable trade-off level between computational time and performance. If an algorithm is too complicated or involves too many variables, it may take a considerable amount of time to solve the problem and may render any solution useless if it is not timely. In this study, the model and algorithms are applied to a small problem and computational complexity does not present a problem. However, for larger problem instances, the time to solve the maze may become excessive and in disaster situations where decisions are time dependent, this may present a problem. To highlight the computational aspects of the two algorithms, the execution time (in milliseconds) was recorded for each of the solutions in Table 5.2. The execution times are summarised in Table 5.3 and graphically represented in Figure 5.13.

Table 5.3 Execution time of the Lee and A-star algorithms

	LEE Algorithm						
From Starting Point	To Hospital						To the closest Hospital
	1	2	3	4	5	6	
A	27316	55055	41799	49091	31006	34477	27316
B	50093	58080	50132	51543	31519	31405	31405
C	15624	13753	1681	10623	6479	13661	1681
D	47919	52072	35597	37726	11319	6327	6327
E	19667	5463	1096	356	10934	13449	356

	A-STAR Algorithm						
From Starting Point	To Hospital						To the closest Hospital
	1	2	3	4	5	6	
A	3072	9197	7391	5879	4198	5465	3072
B	9700	16018	13902	11229	6285	5972	5972
C	1034	1049	183	719	400	1075	183
D	3654	5953	4200	3241	1187	490	490
E	2721	468	126	97	1231	1699	97

Figure 5.13 Graphical analysis of the execution time for the Lee and A-star algorithms

It can be seen from Table 5.3 that there is a significant difference between the execution time of the two algorithms. The A-star algorithm is clearly much faster than the Lee algorithm. Figure 5.14 shows the difference in the average completion time of the two algorithms. Although there is a significant difference in execution times, it should be noted that the longer times recorded for the Lee algorithm would still be acceptable, as the milliseconds translated into seconds means that all solutions are obtained within less than one minute.

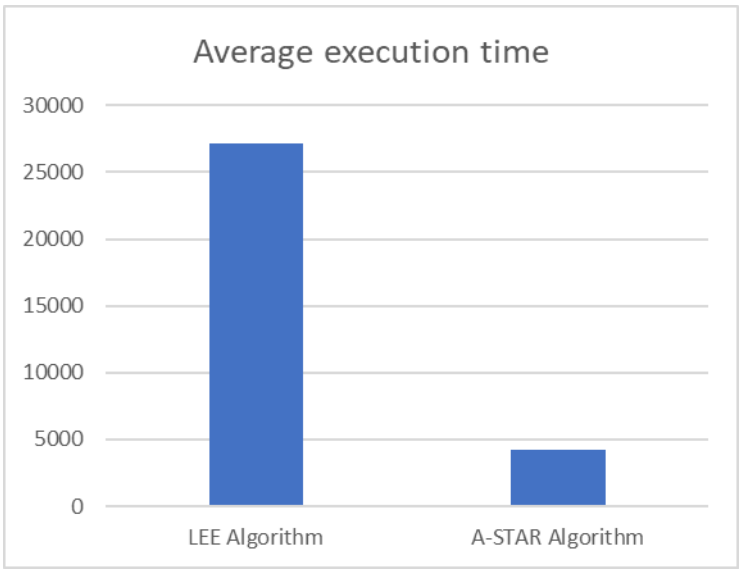


Figure 5.14 Average execution time of the Lee and A-star algorithms

5.5.2. Software development and methodological challenges

The software artefact that was developed performed very well and was able to deliver reliable and accurate results. It displayed a number of advantages and is flexible in respect of the dynamic capability to change starting and end points to determine different optimal paths. A detailed discussion of the contributions will be presented in Chapter 7 and the aim of this section is to highlight some of the challenges that relate particularly to the development of the software solution.

The first methodological challenge was the choice and development of a grid that could be fitted over a map of the disaster-stricken area. Normally, when a grid needs to be placed on a map, the use of geographic information systems (GIS) functions can be employed. A GIS has specialised functions to discretise a map and to place a grid over the map. In this study, a GIS was not available and the size of the grid had to be determined by trial and error.

The matrix used to construct the initial grid for the damaged areas was constructed manually. This may have been done more accurately (and easier) if image processing software was available to determine where walls and paths in the grid should be established.

Due to the combination of the matrix generated maze and the maze generated by Kruskal's algorithm, a number of dead ends occurred and it was necessary to adapt Kruskal's algorithm to provide for these dead ends.

Closely link to the challenge of dead ends is the problem of a valid maze with no path that exists between a starting and an end point. For example, if one of the starting points (one of the green blocks) is situated in an area where there is no path out of the immediate vicinity, the algorithms would not be able to solve the optimum path problem. In this case, the user would have to move the starting point to an open position as close as possible to the original starting point in an effort to guide relief workers to get as close as possible to the victims.

5.5.3. Final remarks

The software solution proved to be successful and delivered the expected results. Positive results were obtained from the process of generating a suitable maze, using a matrix approach together with Kruskal's algorithm. From the discussions above, it is clear that the use of the Lee and A-star algorithms to solve the maze and determine an optimal path was also successful. The choice of algorithm to solve the maze will depend on the real-world situation and the needs of disaster victims and humanitarian logistics personnel. To conclude the discussion on the choice between the Lee and the A-star algorithms, some

final thoughts on the characteristics of the two algorithms are summarised in Table 5.4 below.

Table 5.4 Characteristics of the Lee and A-star algorithms

Lee algorithm	**A-star algorithm**
- Generally slower execution time - Memory intensive - Low complexity - Guarantees to find a solution if one exists	- Generally faster execution time - Memory intensive - Higher complexity - Guarantees to find a solution if one exists

This concludes the general discussion. Further discussion on specific contributions and other advantages will be presented in the final chapter of this study.

5.6. Summary

In Chapter 5, details in respect of a real-world case study in humanitarian logistics based on the techniques and algorithms presented in Chapters 2 and 3 were provided. The focus was on the development of a software solution that implements specific algorithms and demonstrates how these may be used in humanitarian relief efforts. In the next chapter, a solution strategy for the placement of relief facilities will be presented. This strategy will be based on mathematical programming models.

Chapter 6 Relief facility location

6.1. Introduction

In the previous chapter, a real-world case study of the damage caused to New Orleans by Hurricane Katrina was presented and showed how the use of a grid-based maze may be utilised to determine optimal paths to disaster victims. The identified routes can be used either to provide victims with relief supplies or for evacuation to the nearest hospital or relief facility. As alluded to in Section 4.2 (Chapter 4), it is also important to determine optimal locations for additional or temporary healthcare and supply facilities. This is necessary to maintain a humanitarian relief supply chain that can provide the required shelter, medicine, food and other emergency items. The establishment of relief facilities and the determination of optimal traversable routes in a disaster-stricken area are therefore closely linked and often needs to be addressed simultaneously. In this chapter, the application of two general facility location models in the New Orleans area will be demonstrated. Firstly, a discussion of the real-world data on which the facility locations models are based will be given. The two models are then presented, followed by a discussion of the results. Finally, the chapter is concluded with a brief chapter summary.

6.2. The New Orleans data set

The data set used is a real-world New Orleans data set. This data set is the obvious choice to illustrate the facility location models, as it was also used in the grid-based maze solutions presented in Chapter 5. Data required to formulate the facility location models include the demarcation of different districts in the New Orleans area (to determine where to locate facilities), as well as the population figures for each district and neighbourhood within a district (to ensure that the maximum number of people, or possible disaster victims, are covered). The New Orleans data used was obtained from a research centre and is available from the Internet[2]. Population data of 2013-2017 was used, as these were the most correct and complete data sets available.

It is important to note that certain assumptions are made about the data and the model formulation process. These assumptions include

- The facility location models are formulated on a high level and independently from the maze structure used in Chapter 5 – i.e. facilities are located in neighbourhoods and not

[2] https://www.datacenterresearch.org/data-resources/neighborhood-data/

on specific coordinates where they may or may not be located in an inaccessible area according to the maze.

- It is assumed that a facility in a specific neighbourhood will have the capacity to serve that neighbourhood as well as all the immediate adjacent neighbourhoods (this is called the coverage area of a facility).
- The population of a neighbourhood will be treated as an importance weight to ensure that the maximum number of people is covered.
- The cost of establishing a facility is not taken into account and it is assumed that sufficient financial and other resources are available for this. Cost can easily be incorporated into the models (see Chapter 4); however, accurate cost figures are not available and hence the assumption.

The data set consists of a geographical map of the New Orleans area. This geographical area is divided into seven main districts with each district consisting of a number of neighbourhoods. Figure 6.1 shows the area and the seven districts numbered from A to G. There is a total of 52 neighbourhoods in the seven districts. Details of these neighbourhoods, as well as the population sizes of each neighbourhood are presented in Annexure C.

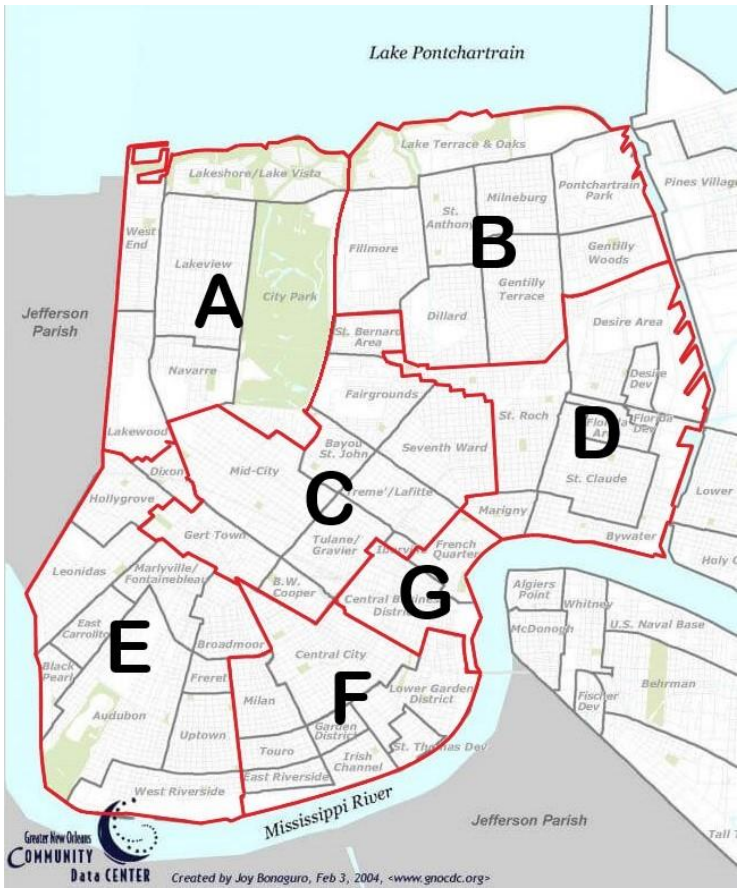


Figure 6.1 The seven districts of the New Orleans area

As mentioned under the assumptions, the coverage area is defined as the selected neighbourhood plus the adjacent neighbourhoods. Consider for example the Gentilly district (District B in Figure 6.1). If a facility is to be located in the neighbourhood Milneburg, then the coverage area is defined as Milneburg, as well as the adjacent neighbourhoods Lake Terrace and Lake Oaks, St Anthony, Pontchartain Park, Gentilly Terrace and Gentilly Woods. This coverage area is illustrated in Figure 6.2.

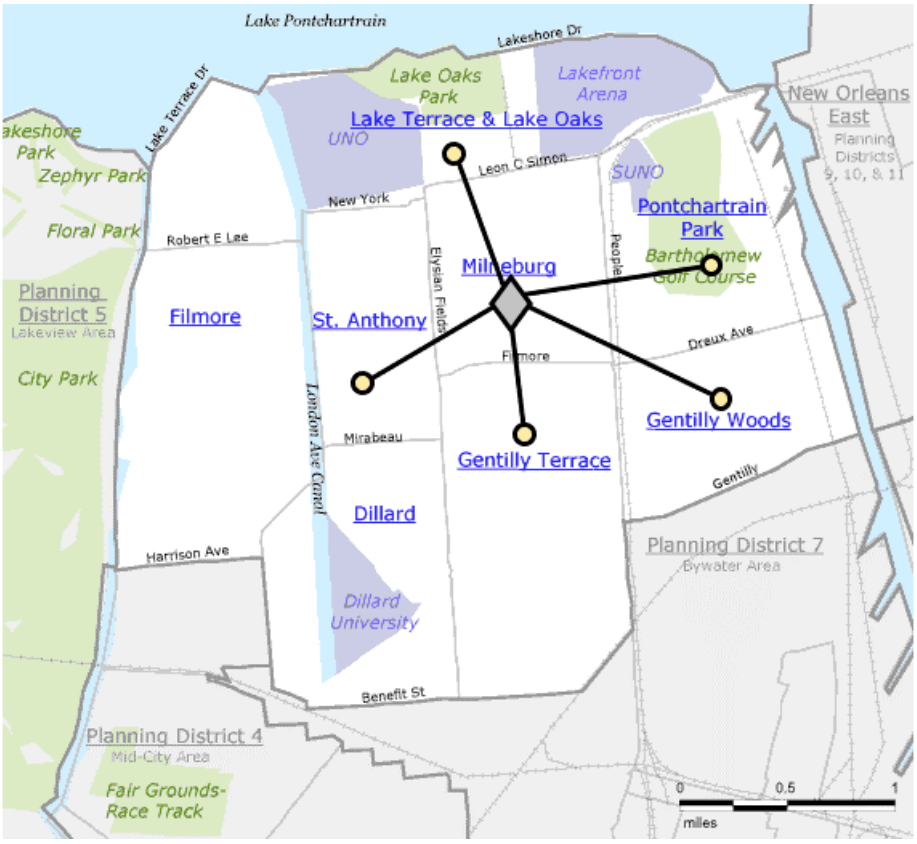


Figure 6.2 Illustration of a coverage area within the Gentilly district

It should be noted that coverage areas are not limited to a specific district. A neighbourhood in one district (e.g. District A) may be adjacent to a neighbourhood in another district (e.g. District B) in which case they will form part of the same coverage area. For modelling purposes, a number assignment scheme is introduced to keep track of different neighbourhoods in different districts. This also provides for conditions of coverage areas that overlap more than one district. The numbering scheme works as follows:

- Assign a value from A to G to each of the seven districts.
- Assign a numeric value to each of the neighbourhoods in each of the districts, starting at one for each district. For example, A3 would indicate District A and neighbourhood 3.

As an example, the Glentilly district in Figure 6.1 was assigned a value of B. Table 6.1 shows the coverage area for each neighbourhood in the district, i.e. the adjacent neighbourhood. Note that not all the adjacent neighbourhoods fall in District B.

Table 6.1 Coverage area and neighbourhood assignment for District B

Value	Neighbourhood	Adjacent neighbourhoods
B1	Lake Terrace & Lake Oaks	B2, B3, B4, B5, A1, A4
B2	Filmore	B1, B3, B6, A1, A4, C1
B3	St. Anthony	B1, B2, B4, B6, B7
B4	Milneburg	B1, B3, B5, B7, B8
B5	Pontchartain Park	B1, B4, B7, B8,
B6	Dillard	B2, B3, B7, C1, D2
B7	Gentilly Terrace	B3, B4, B5, B6, B8, D2, D1
B8	Gentilly Woods	B4, B5, B7, D1

The final data requirement, before the facility models can be formulated, is to create an incidence matrix that shows the relationship between neighbourhoods; in this case, the relationship of being an adjacent neighbour or not. The incidence matrix contains either a value of 0 or 1. A value of 1 indicates that the neighbourhoods are adjacent and a 0 indicates that they are not adjacent. Table 6.2 shows an extract of the incidence matrix. The complete incidence matrix with all 52 neighbours is presented in Annexure C.

Table 6.2 Extraction of the neighbourhood incidence matrix

	A1	A2	A3	A4	A5	A6	B1	B2	B3	B4	B5	B6	B7	B8
A1	0	1	1	1	0	0	1	1	0	0	0	0	0	0
A2	1	0	1	0	1	0	0	0	0	0	0	0	0	0
A3	1	1	0	1	1	1	0	0	0	0	0	0	0	0
A4	1	0	1	0	0	1	1	1	0	0	0	0	0	0
A5	0	1	1	0	0	1	0	0	0	0	0	0	0	0
A6	0	0	1	1	1	0	0	0	0	0	0	0	0	0
B1	1	0	0	1	0	0	0	1	1	1	1	0	0	0
B2	1	0	0	1	0	0	1	0	1	0	0	1	0	0
B3	0	0	0	0	0	0	1	1	0	1	0	1	1	0
B4	0	0	0	0	0	0	1	0	1	0	1	0	1	1
B5	0	0	0	0	0	0	1	0	0	1	0	0	1	1
B6	0	0	0	0	0	0	0	1	1	0	0	0	1	0
B7	0	0	0	0	0	0	0	0	1	1	1	1	0	1
B8	0	0	0	0	0	0	0	0	0	1	1	0	1	0

The incidence matrix and the population figures for each neighbourhood are the complete data set required for the formulation of the facility location models. These models will be presented in the next section.

6.3. The relief facility location models

In Chapter 4, a comprehensive overview of discrete facility location models that may be employed to determine sites for establishing relief facilities was presented. Two facility location problems will be formulated in this section, using the real-world data set discussed in the preceding section. The two models are a *set covering* model and a *maximal covering* model. Both models are solved, using the Solver optimisation add-in facility in Microsoft Excel.

6.3.1. A set covering facility location model

In a disaster-stricken area, the aim is to assist all victims. This means that a sufficient number of relief and supply facilities should be located to ensure that all regions within the disaster area are covered in a timely manner. A set covering facility location model provides a method to minimise the number of facilities while ensuring that each region is covered. Minimising the number of facilities also implies that the cost is minimised; less facilities suggest less cost.

In the context of this study, the objective is to determine the minimum number of neighbourhoods, as well as which neighbourhoods, in the New Orleans area where a relief facility should be established so that all 52 neighbourhoods are covered (based on the adjacency of neighbours). To achieve this, a standard set covering model, as formulated in Section 4.3.2.1 in Chapter 4, can be employed. For the sake of completeness, the model formulation is repeated here.

Let $\mathcal{J}$ represent a set of candidate locations (neighbourhoods) with $\mathcal{N}$ the set of all adjacent neighbourhoods covered by a neighbourhood.

Define the binary decision variable x_j as follows

$$x_j = \begin{cases} 1 & \text{if a facility is located at candidate neighbourhood } j \in \mathcal{J} \\ 0 & \text{otherwise} \end{cases}$$

The model formulation is then given by

$$\text{Minimise} \quad \sum_{j \in \mathcal{J}} x_j \tag{6.1}$$

$$\text{subject to} \quad \sum_{j \in \mathcal{N}} x_j \geq 1, \tag{6.2}$$

$$x_j \in \{0,1\} \qquad j \in \mathcal{J}. \tag{6.3}$$

The objective function (6.1) will minimise the number of facilities required. Constraint set (6.2) guarantees that each neighbourhood is covered while constraint set (6.3) ensures the binary nature of the decision variables x_j. In this model $j = 1, \ldots, 52$ as there are 52 neighbourhoods in the New Orleans area.

Solving the above set covering model (6.1) – (6.3) results in a minimum of ten facilities required to cover all neighbourhoods, based on the adjacency assumption. The ten facilities should be located in the neighbourhoods indicated in Table 6.3. See also Figure 6.3 where the facility locations and neighbourhoods covered by each facility are graphically displayed.

Table 6.3 Minimum number of facility locations in the New Orleans area

Facility number	Neighbourhood
1	Lakeshore/LakeVista
2	Lake Terrace and Oaks
3	Desire
4	St Roch
5	Mid-City
6	Gert Town
7	Central Business District
8	Audubon
9	Uptown
10	Lower Garden District

Figure 6.3 Set covering location model result indicating ten facilities

Note that the neighbourhoods on the right-hand side, as well as at the bottom-right of Figure 6.3 do not form part of the area that is studied. Refer to Figure 6.1 to see the boundaries of the area under study.

The set covering model can also be extended to make provision for a large number of special needs or situations. Some of these extensions are as follows.

- *Exclude or include specific neighbourhoods as possible facility location sites*

There may be situations where it is decided to establish a relief facility in a specific neighbourhood due to practical or other reasons. Consider the neighbourhood Fair Grounds which has been identified as a neighbourhood that must have a relief facility. Assume that Fair Grounds is neighbourhood number z. To achieve this requirement an additional constraint is added to the model (6.1) – (6.3) that is formulated as follows

$$x_z = 1 \tag{6.4}$$

Setting the decision variable x_z that corresponds to neighbourhood z (Fair Grounds), to 1 will force the model to select neighbourhood z as a facility location site. Similarly, adding constraint (6.5) will exclude neighbourhood z as a possible site for locating a facility.

$$x_z = 0 \tag{6.5}$$

These types of constraint are useful in situations where neighbourhoods are inaccessible or when neighbourhoods already have infrastructure available to establish a relief facility.

- *Conditional selections*

A conditional constraint refers to the selection of a specific neighbourhood on the condition that another specified neighbourhood has already been selected. Suppose neighbourhood 1 can only be selected to build a relief station if neighbourhood 2 has already been selected (if neighbourhood 2 is not selected then neighbourhood 1 may not be selected). The following constraint will achieve this.

$$x_1 \leq x_2 \tag{6.6}$$

- *Mutually exclusive selection*

A mutually exclusive selection reflects the contingency that either neighbourhood 1 or neighbourhood 2, but not both, can be selected as a relief facility site. The following constraint may be added to the model to enable a mutually exclusive selection between neighbourhoods 1 and 2.

$$x_1 + x_2 \leq 1 \tag{6.7}$$

Constraints (6.4) – (6.7) are examples of logical decision-making constraints and are extremely useful to decision makers, as it enables a modeller to provide for a large variety of practical situations. Additional logical constraints and further detailed discussions on these types of constraint can be found in Taylor (2019).

6.3.2. A maximal covering facility location problem

As opposed to the set covering model which aims to find a minimum number of facilities to cover ***all*** neighbourhoods, the maximal covering model aims at finding an optimal number of facilities that would cover the ***highest percentage of the total population***. This optimum number of facilities has to satisfy certain limitations such as the availability of resources. Depending on resources, and therefore the number of facilities allowed, not all neighbourhoods may be covered, as the model differentiates between small and large populations in the different neighbourhoods. For example, if the available resources allow for the establishment of only one relief facility, it is clear from the adjacent neighbourhood assumption that not all neighbourhoods can be covered. In this case, the model will select a neighbourhood that will ensure that the highest possible population is covered. The maximal covering model therefore aims at locating p facilities in order to maximise the percentage of population covered, using the adjacent neighbourhood assumption as predetermined measure. The model is formulated and discussed in Section 4.3.2.1 in Chapter 4 and the formulation is repeated here for the sake of completeness.

Let $\mathcal{J}$ represent a set of candidate locations (neighbourhoods) with $\mathcal{N}$ the set of all adjacent neighbourhoods covered by a neighbourhood. Two additional input parameters are defined, w_i (the population of neighbourhood i) and p (the number of candidate neighbourhood locations to be established).

Define two binary decision variables x_j and z_i as follows:

$$x_j = \begin{cases} 1 & \text{if a facility is located at candidate neighbourhood } j \in \mathcal{J} \\ 0 & \text{otherwise} \end{cases}$$

$$z_i = \begin{cases} 1 & \text{if neighbourhood } i \text{ is covered} \\ 0 & \text{otherwise.} \end{cases}$$

The model formulation is then given by

$$\text{Maximise} \quad \sum_{\forall\, i} w_i z_i \tag{6.8}$$

$$\text{subject to} \quad \sum_{j \in \mathcal{J}} x_j = p, \tag{6.9}$$

$$z_i \leq \sum_{j \in \mathcal{N}} x_j \qquad \forall\, i, \tag{6.10}$$

$$z_i \in \{0,1\} \qquad \forall\, i, \tag{6.11}$$

$$x_j \in \{0,1\} \qquad j \in \mathcal{J}. \tag{6.12}$$

In the above model, the objective function (6.8) maximises the total covered population. The constraint (6.9) ensures that p facilities are established, while constraint set (6.10) guarantees that a neighbourhood is covered by an open facility. The two constraint sets (6.11) and (6.12) define the binary decision variables.

The value of p is determined in practice by the resources available to establish facilities. Another way of determining a "good" value for p is to experiment with different values and then select the p-value that covers the highest acceptable population.

In this study, a predetermined p-value was not available and to illustrate the model, a range of p-values from 1 to 10 is used. The model is therefore solved ten times, starting with a p-value of 1 and then incrementing the p-value with 1 to a maximum value of 10. The results for each of the 10 solutions indicate the total population covered (which can be used as an acceptance or rejection factor), the neighbourhood where the facility should be located and the neighbourhoods covered by each facility. Table 6.4 shows the results of the 10 solutions which can also be graphically viewed in Figures 6.4 to 6.8.

Table 6.4 Maximal coverage results

Number of facilities (p-value)	**Total population covered**	**Neighbourhood(s) where facilities are located**	**Number of neighbourhoods covered**
1	60901	Gert Town	10
2	113460	Gert Town; St Roch	21
3	147653	Gert Town; St Roch; Uptown	28
4	174245	Gert Town; St Roch; Uptown; City Park	35
5	196674	City Park; Gentilly Terrace; Gert Town; St. Roch; Uptown	39
6	214370	City Park; Gentilly Terrace; St. Roch; East Carrollton; Central City; Touro	42
7	231450	Lakewood; Lake Terrace & Oaks; St. Roch; 7th Ward; East Carrollton; Central City; Touro	47
8	237217	Lakewood; Lake Terrace & Oaks; St. Roch; 7th Ward; Leonidas; Marlyville Fontainebleau; Central City; Touro	48
9	242835	Lakeview; City Park; Gentilly Terrace; St. Roch; 7th Ward; Leonidas; Marlyville Fontainebleau; Central City; Touro	48
10	248238	Lakeshore Lake Vista; Lake Terrace & Oaks; Mid-City; St. Roch; Desire dev & neighbourhood; Gert Town; Audubon; Uptown; Lower Garden District	52

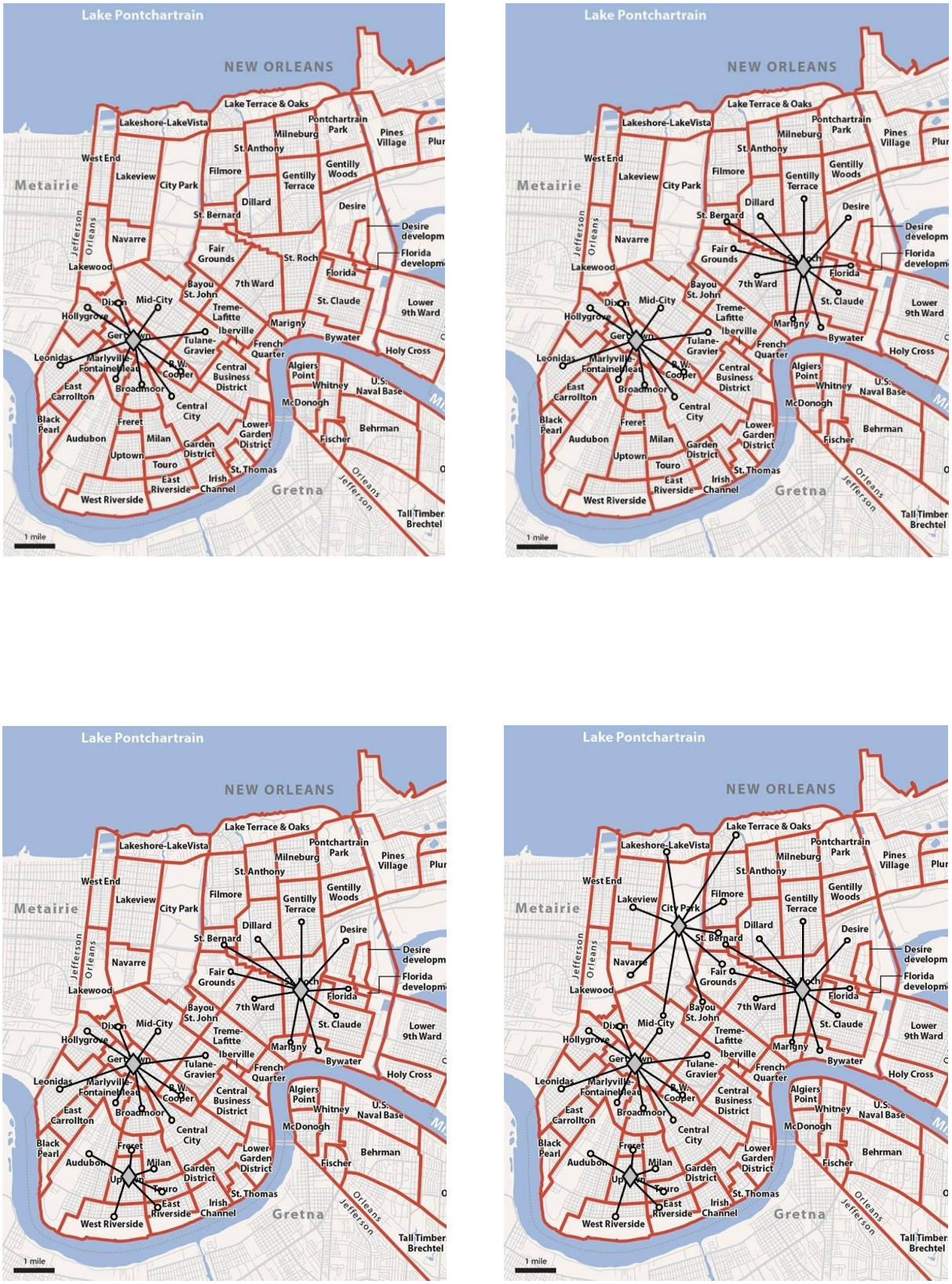


Figure 6.5 Locations of three and four facilities


Lake Pontchartrain
NEW ORLEANS
Metairie
Lake Terrace & Oaks
Lakeshore-LakeVista
Pontchartrain Park
Milneburg
Pines Village
St. Anthony
West End
Filmore
Gentilly Woods
Lakeview
City Park
Gentilly Terrace
Dillard
Desire
St. Bernard
Desire developm
Jefferson
Orleans
Navarre
Fair Grounds
St. Roch
Florida developm
Lakewood
Florida
7th Ward
Bayou St. John
Mid-City
St. Claude
Lower 9th Ward
Dixon
Treme-Lafitte
Hollygrove
Marigny
Gert Town
Iberville
Bywater
Tulane-Gravier
French Quarter
Holy Cross
Leonidas
Marlyville-Fontainebleau
B.W. Cooper
Central Business District
Algiers Point
Whitney
U.S. Naval Base
East Carrollton
Broadmoor
McDonogh
Central City
Black Pearl
Freret
Lower Garden District
Behrman
Audubon
Milan
Garden District
Fischer
Uptown
Touro
St. Thomas
East Riverside
Irish Channel
West Riverside
Gretna
Tall Timbers Brechtel
1 mile


Lake Pontchartrain
NEW ORLEANS
Metairie
Lake Terrace & Oaks
Lakeshore-LakeVista
Pontchartrain Park
Milneburg
Pines Village
St. Anthony
West End
Filmore
Gentilly Woods
Lakeview
City Park
Gentilly Terrace
Dillard
Desire
St. Bernard
Desire developm
Jefferson
Orleans
Navarre
Fair Grounds
St. Roch
Florida developm
Lakewood
Florida
7th Ward
Bayou St. John
Mid-City
St. Claude
Lower 9th Ward
Dixon
Treme-Lafitte
Hollygrove
Marigny
Gert Town
Iberville
Bywater
Tulane-Gravier
French Quarter
Holy Cross
Leonidas
Marlyville-Fontainebleau
B.W. Cooper
Central Business District
Algiers Point
Whitney
U.S. Naval Base
East Carrollton
Broadmoor
McDonogh
Central City
Black Pearl
Freret
Lower Garden District
Behrman
Audubon
Milan
Garden District
Fischer
Uptown
Touro
St. Thomas
East Riverside
Irish Channel
West Riverside
Gretna
Tall Timbers Brechtel
1 mile


Lake Pontchartrain
NEW ORLEANS
Metairie
Lake Terrace & Oaks
Lakeshore-LakeVista
Pontchartrain Park
Milneburg
Pines Village
St. Anthony
West End
Filmore
Gentilly Woods
Lakeview
City Park
Gentilly Terrace
Dillard
Desire
St. Bernard
Desire developm
Jefferson
Orleans
Navarre
Fair Grounds
St. Roch
Florida developm
Lakewood
Florida
7th Ward
Bayou St. John
Mid-City
St. Claude
Lower 9th Ward
Dixon
Treme-Lafitte
Hollygrove
Marigny
Gert Town
Iberville
Bywater
Tulane-Gravier
French Quarter
Holy Cross
Leonidas
Marlyville-Fontainebleau
B.W. Cooper
Central Business District
Algiers Point
Whitney
U.S. Naval Base
East Carrollton
Broadmoor
McDonogh
Central City
Black Pearl
Freret
Lower Garden District
Behrman
Audubon
Milan
Garden District
Fischer
Uptown
Touro
St. Thomas
East Riverside
Irish Channel
West Riverside
Gretna
Tall Timbers Brechtel
1 mile


Lake Pontchartrain
NEW ORLEANS
Metairie
Lake Terrace & Oaks
Lakeshore-LakeVista
Pontchartrain Park
Milneburg
Pines Village
St. Anthony
West End
Filmore
Gentilly Woods
Lakeview
City Park
Gentilly Terrace
Dillard
Desire
St. Bernard
Desire developm
Jefferson
Orleans
Navarre
Fair Grounds
St. Roch
Florida developm
Lakewood
Florida
7th Ward
Bayou St. John
Mid-City
St. Claude
Lower 9th Ward
Dixon
Treme-Lafitte
Hollygrove
Marigny
Gert Town
Iberville
Bywater
Tulane-Gravier
French Quarter
Holy Cross
Leonidas
Marlyville-Fontainebleau
B.W. Cooper
Central Business District
Algiers Point
Whitney
U.S. Naval Base
East Carrollton
Broadmoor
McDonogh
Central City
Black Pearl
Freret
Lower Garden District
Behrman
Audubon
Milan
Garden District
Fischer
Uptown
Touro
St. Thomas
East Riverside
Irish Channel
West Riverside
Gretna
Tall Timbers Brechtel
1 mile

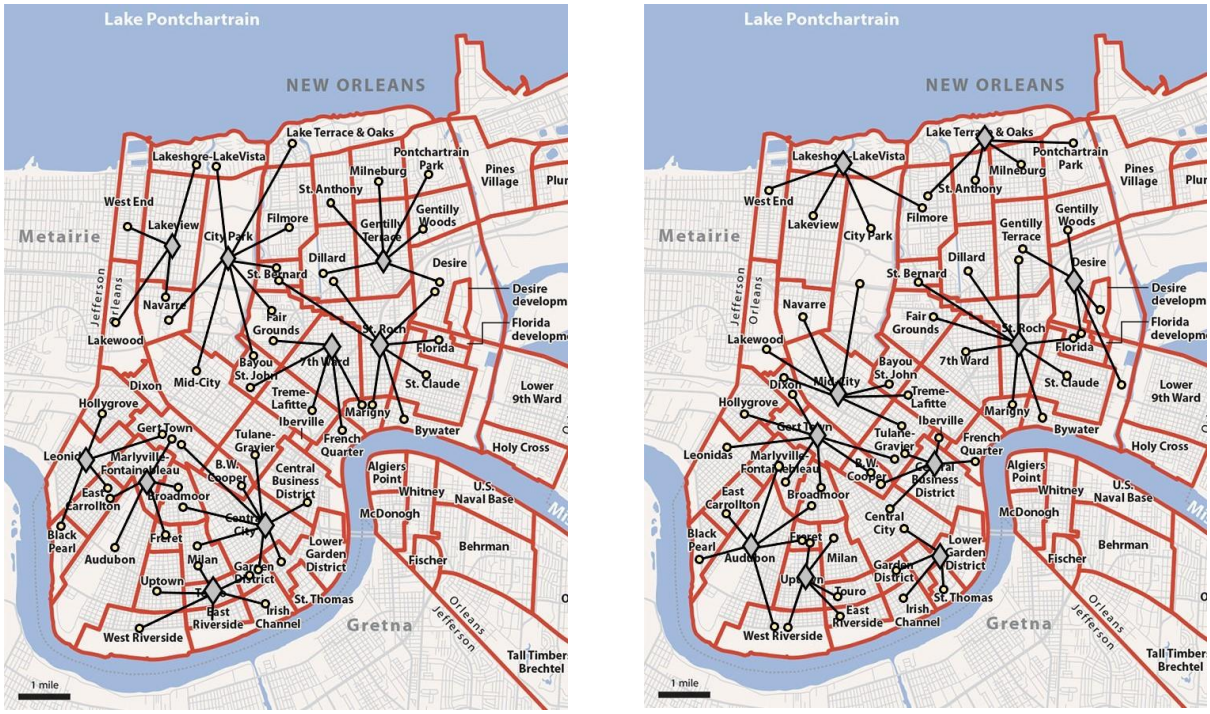


Figure 6.8 Locations of nine and ten facilities

The same extensions, discussed under the set covering model, are applicable here and may be incorporated into the maximal coverage model, based on prior knowledge that the decision maker may have.

6.4. Discussion

Facility location models play an important role in humanitarian logistics, as they provide a means for locating relief facilities in such a way that people or victims can be reached and supported in a timely manner. In the preceding section, two standard facility location models (set covering and maximal covering) were applied to a real-world case study of the New Orleans region.

The aim of this section is not to compare the two models, but rather to illustrate how different types of facility models can complement each other, as well as to show the powerful and appropriate solutions that standard models can provide. Two scenarios were selected to demonstrate this; firstly, to determine the minimum number of facilities required to cover a region (set covering model) and secondly, to maximise the number of people that can be covered by a selected number of facilities (maximal covering model). Both models form part of a class of discrete facility location models and both are formulated as binary (0-1) integer linear programming models.

The *set covering model*, which determines a minimum number of facilities required to cover a region, offers a number of advantages to modellers and decision makers and some of these advantages are illustrated in the application of the model in the New Orleans region.

- The standard set covering model is easy to use and requires only a limited number of features as a data set. In this application, the only requirement is a list of neighbourhoods that are adjacent to each other. Only a basic understanding of linear programming models is needed to understand the mathematical concepts of the model and to be able to define the decision variables.
- Depending on the size of the problem, the model can be solved in a very short time (measured in minutes and seconds), using the Solver add-in of Excel. For larger problem instances, specialised optimisation software such as CPLEX[3] is available to solve the models.
- The use of logical constraints will enhance the capabilities of the set covering models. If prior knowledge exists (e.g. neighbourhoods that are completely inaccessible), the set covering model can easily be adapted to make provision for this limitation. Other types of model adaption or extension are listed in Section 6.3.1.
- In the application described in this chapter, an adjacent neighbourhood measurement is used to determine the coverage area. This measurement can easily be adapted to a distance measurement (all neighbourhoods within a specific distance from the facility are covered, as opposed to only the adjacent neighbours) or a time measurement (all neighbourhoods that can be reached within a pre-specified time period are covered as opposed to only the adjacent neighbours).
- A cost factor can effortlessly be incorporated into the model. By changing the objective function in (6.1) to the objective function in (6.13) where f_i represents the cost of establishing facility i the cost will be minimised in the search for the minimum number of facilities.

$$\text{Minimise} \sum_{j \in \mathcal{J}} f_i x_j \ \forall \ i \tag{6.13}$$

The above arguments confirm that the set covering model offers a number of unique features that can be used in a disaster-stricken area to decide on the minimum number of relief facilities and where to locate them.

The *maximal covering model*, which determines a maximum population covered given a pre-specified number of facilities required, also suggests that a number of advantages can be achieved by modellers and decision makers. Some of these advantages overlap with the

[3] https://www.ibm.com/za-en/analytics/cplex-optimizer

benefits gained from a set covering model, but will be briefly mentioned again for completeness' sake.

- The main advantage of the maximal coverage model (which does not form part of the set covering model) is that experiments can be conducted to determine the best number of facilities, given that a pre-specified percentage or proportion of the total population must be covered. Even if a pre-specified population size is not available, the model can be solved repeatedly with different numbers of facilities (as was described in section 6.3.2). This gives decision makers an opportunity to make an informed decision on the number of facilities (and the cost) and the proportion of the total population that will be covered.

The following benefits are the same as for the set covering model and are only briefly referred to here.

- The maximal covering model is easy to use. In addition to a list of neighbourhoods that are adjacent to each other, the populations of each neighbourhood are also required. A basic understanding of mathematical models is needed to be able to define the decision variables and formulate the model.
- Small and medium-sized models can be solved in Excel with the Solver add-in, but larger problems may require specialised software.
- Logical constraints may be used to cater for prior knowledge.
- Other coverage measurements, such as time and distance may be used instead of adjacency of neighbourhoods.

The two types of facility location model applied in this study have highlighted the important role of these types of mathematical model in humanitarian logistics. The models may be used to locate relief and rescue facilities in disaster situations and provides for a range of options, based on prior knowledge about the disaster and damage caused. The models go hand in hand with determining optimal traversable routes, but may also be applied in other contexts that do not necessarily include disaster scenarios.

6.5. Summary

In Chapter 6, the real-world case study presented in Chapter 5 was expanded by presenting two mathematical models that can be used to locate relief facilities in a disaster-stricken area. The models were applied to the same New Orleans disaster region used in Chapter 5 and illustrate how the optimal minimum number of facilities can be determined, as well as the relationship between the number of facilities and the proportion of the population that is

covered by the facilities. In the next chapter, the final concluding remarks of the study are presented.

Chapter 7 Conclusion

7.1. Introduction

The primary objective of this research was to determine to what extent mathematical modelling techniques can assist and support the activities of a humanitarian logistics chain in a natural disaster scenario. The research question, as formulated in Chapter 1, Section 1.2, is repeated here:

To what extent can a grid-based maze approach, linked with a facility location problem, support the activities of a humanitarian logistics chain in a disaster-stricken area?

To answer the overarching research question, the use of discrete grid-generating algorithms was proposed to construct a grid-based maze that could be used in a real-world disaster scenario to represent accessible and inaccessible areas. Two different algorithmic solutions were then evaluated to solve the maze and thereby assist with the determination of optimum traversable routes that could be used by relief and rescue workers to reach disaster victims. A software solution (system) was built to demonstrate the proposed techniques and algorithms. In addition to this, two different facility location models were also formulated and solved to assist with the establishment of relief facilities. The facility location models provided valuable information with regard to the number of facilities to be established, as well as the specific regions where these facilities should be established. The maximum population covered by established facilities was also determined. The proposed techniques and models were applied in a real-world disaster situation and data obtained from Hurricane Katrina that occurred in 2005 in New Orleans in the United States of America was used. Results obtained from the application of the models and algorithms suggest that the proposed methodology does indeed produce valuable and useful results that are typically required in a humanitarian logistics scenario.

The objective of this chapter is to conclude the study by evaluating the achievement of goals set in Chapter 1. This evaluation is presented in Section 7.2, followed by a summary of the contributions made by the study and problems encountered during the study. Possible opportunities for future research are also highlighted. The chapter is then concluded with some final remarks.

7.2. Evaluation of research goals

The study was guided by the research question formulated in Chapter 1 and repeated above in Section 7.1. To achieve the overall objective and to answer the research question, four

secondary objectives were set. In this section, a brief overview of how the four secondary objectives were achieved is presented.

1. ***Conduct a literature review on maze generation algorithms, solving strategies and facility location modelling problems***

A comprehensive literature review was conducted on each of the topics mentioned in the first secondary objective. Concepts were defined and explained based on the literature and related studies were quoted to support the ideas, models and algorithms proposed in this study. Chapter 2 was devoted to grid-based maze structures and the existing algorithms that may be used to generate the said maze structures. Four of the most well-known and popular maze generating algorithms were discussed and explained, i.e. Prim's algorithm, Kruskal's algorithm, a recursive backtracking algorithm, and the hunt-and-kill algorithm. Appropriate literature references to related studies were also provided. The focus of Chapter 3 was on literature review studies concerned with humanitarian logistics and possible solving strategies and models that could be used to solve maze structures. Four maze solving algorithms were reviewed and explained in detail, i.e. the Lee algorithm, A-star algorithm, flood-fill algorithm and a recursive backtracking algorithm. Finally, in Chapter 4, a literature review of facility location models and the mathematical formulation of such models that can be used to maximise relief efforts in a disaster-stricken area was presented. A brief introduction to healthcare and relief supply facilities was provided, followed by a more comprehensive discussion on discrete facility location models and formulations. Related studies in the literature were also presented.

Three chapters were dedicated to produce a comprehensive literature review of related studies and to explain all modelling techniques and concepts used in the study. The first sub-objective was fully achieved.

2. ***Design and develop a grid-based maze generator to represent traversable routes in a real-world disaster-stricken area***

To ensure that the proposed methodology was applicable, and could be applied to real-world situations, data obtained from Hurricane Katrina that hit New Orleans in the United States of America in 2005 was used. This data set was described in Chapter 5, Section 5.2. To develop a grid-based maze for the New Orleans disaster region, the following five steps had to be completed:

- Grid placement;
- Develop a matrix to represent the data;

- Generate a maze from the matrix;
- Use a maze generation algorithm (Kruskal's algorithm) to complete the maze; and
- Fit the final maze over the disaster-stricken area under consideration.

A software solution (using the C# programming language) was developed to generate the maze that could represent the actual damage caused by Hurricane Katrina in New Orleans. The abovementioned five steps to generate the real-world maze were described in Chapter 5, Sections 5.3.1-5.3.4, while the software implementation of the maze generation was detailed in Chapter 5, Section 5.4.1.

The successful development of a methodology to generate a maze structure that could represent the damage in the real-world disaster area, as well the effective software implementation of the process indicate that the second sub-objective was met.

3. Evaluate and apply two different optimum path-finding algorithms to the grid-based maze in the real-world disaster-stricken area

To illustrate the need, importance and usefulness of the grid-based structure developed in sub-objective two, appropriate maze-solving algorithms were also implemented in the software solution. These algorithms produced optimal traversable routes from a starting point (where victims are located) to and end point (the nearest, or specified, medical facility). Two algorithms were selected for implementation, the Lee algorithm and the A-star algorithm. In Chapter 5, Sections 5.4.2 and 5.4.3, details of the implementation of the two selected algorithms in the software solution were given. In Section 5.5, a full discussion and evaluation of the results of the two algorithms in the software solution were presented.

A successful software solution was created that utilised two different algorithms to find optimal routes in a real-world natural disaster area. The results were meaningful and will be of value to decision makers in humanitarian logistics where optimal traversable routes need to be determined in a short period of time. The third sub-objective was therefore fully achieved.

4. Formulate and implement appropriate facility location models to address relief and rescue station location problems in the real-world scenario

In disaster situations, it is important to determine optimal locations for additional or temporary healthcare and supply facilities. This is necessary to maintain a humanitarian relief supply chain that can provide the required shelter, medicine, food and other emergency items. The establishment of relief facilities and the determination of optimal

traversable routes in a disaster-stricken area are therefore closely linked and often need to be addressed simultaneously. To address the problem of relief facility locations, two standard facility location models were formulated and solved. The same real-world New Orleans data set was used to solve the models. The data set for the facility location models was described in Chapter 6, Section 6.2. The first model formulated and solved was a set covering model. This type of model determines a minimum number of facilities required to cover all people (the complete population) in the disaster-stricken area in a timely manner. The set covering model and results were presented in Chapter 6, Section 6.3.1. The second model is called a maximal covering facility location model and aims to cover the highest percentage of the total population depending on the number of facilities that can be established. The maximal covering model and results were presented in Chapter 6, Section 6.3.2. A discussion of the results and the characteristics of the two models was presented in Chapter 6, Section 6.4.

The models were formulated and solved successfully. Based on the assumption that a facility can cover all adjacent neighbourhoods, it was possible to determine that 10 relief facilities (and the neighbourhoods where they should be established) are needed to cover the entire New Orleans disaster area. The total area consists of 52 neighbourhoods with a total population of 248 238. The maximal coverage model was solved for a range of facilities (i.e. 1 facility, 2 facilities, 3 facilities, …, 10 facilities). The results indicated where (in which neighbourhood) facilities should be established and the maximum population covered for each number of facilities. The fourth sub-objective is deemed to have been fully achieved.

To summarise, the primary research objective (formulated as the research question) and all the sub-objectives set in Chapter 1 were successfully achieved. The final conclusions can therefore be summarised as follows:

- A grid-based maze generator can be developed successfully by combining existing algorithms with a new adapted matrix maze approach. This maze can accurately represent a real-world natural disaster area or region;
- An efficient and accurate software solution can be developed to implement maze solving strategies that can determine optimal traversable routes in a real-world disaster-stricken area; and
- The problem of establishing relief or medical facilities in a real-world disaster region can be solved rapidly and efficiently by formulating and solving standard mathematical programming facility location models.

The contributions that this study has made will be discussed in the next section.

7.3. Contributions

This study has made a number of contributions in both humanitarian logistics and mathematical modelling techniques. These contributions include the following:

- The literature study completed during the research project provided a framework of relevant literature that is organised according to the different themes of the study. This may serve as a central source for other studies that are conducted in the same research area.
- A novel grid-based structure that can represent damage caused by natural disasters was developed. The technique used was novel in the sense that existing grid-generating algorithms do not fully provide for the unique requirements of a disaster area. A matrix grid approach was developed to overcome these problems. To prevent large open spaces in the grid, the matrix grid approach was combined with an existing grid-generating algorithm. This combination of techniques produces a perfect grid that could be utilised for solving other problems.
- A software application was developed that allows users to firstly, specify starting points (where victims are located) and end points (relief or medical facilities) and secondly, to determine an optimal traversable route through the maze in a user-friendly manner. These routes can then be used for evacuation purposes or to send relief supplies to disaster victims.
- Experiments with two facility location models were conducted. The first model was used to determine a minimum number of facilities required to cover the complete disaster area. The second model was employed to demonstrate the maximum number of people covered, as the number of facilities to be established was varied. This provided useful information to decision makers in determining where, and how many relief facilities should be established.
- An extensive explanation was provided on how the facility location models can be extended to deliver even more insightful results, e.g. the use of logical constraints when prior knowledge about the situation is available; and the incorporation of cost into the models.
- All the above contributions were demonstrated, using a real-world natural disaster scenario. This indicates that the techniques, algorithms, models and the results obtained are not simply theoretical results, but that a material contribution to practical situations was made.

7.4. Limitations

The availability of a Geographic Information Systems (GIS) facility that accurately depicts all the damage on a map would have made the construction of a matrix (to develop a grid) much easier. This was done manually and took a considerable amount of time. This manual process also means that the matrix construction was subjective and based on the judgement of the researcher.

It was not possible to obtain additional data, such as cost to establish a relief facility or damage to specific neighbourhoods (i.e. a completely inaccessible neighbourhood cannot be treated as a candidate site for establishing a relief facility). This limitation had an influence on the different types of models and parameters that could be used.

7.5. Future work

In this study, an assumption was made that a relief facility in a specific neighbourhood could cover all adjacent neighbourhoods. Other rules for covering neighbourhoods can be investigated, e.g. a relief facility in a neighbourhood may be able to cover neighbourhoods that can be reached in a predetermined time; or that are within a predetermined distance. Using, and comparing, these different rules would be interesting.

Depending on the availability of appropriate data, the facility location models may incorporate other aspects, such as costs and prior knowledge (hard exclusions or inclusions of neighbourhoods, or minimum thresholds for the maximum number of people that must be covered).

The automation of the matrix grid construction can be achieved by using a GIS or making use of image recognition software.

7.6. Summary

This is the final chapter and concludes the study. A summary of how the research objectives set in Chapter 1 were achieved was presented. This was followed by the contributions of the study and some of the challenges experienced during the study. Finally, opportunities for future research were highlighted.

Bibliography

Acar, M. & Kaya, O. 2019. A healthcare network design model with mobile hospitals for disaster preparedness: a case study for Istanbul earthquake. *Transportation Research Part E: Logistics and Transportation Review,* 130:273-292.

Adamatzky, A. 2012. Slime mold solves maze in one pass, assisted by gradient of chemo-attractants. *IEEE Transactions on Nanobioscience,* 11(2):131-134.

Adamatzky, A., Chiolerio, A. & Szaciłowski, K. 2020. Liquid metal droplet solves maze. *Soft Matter*, 16(6):1455-1462.

Ahmadi-Javid, A., Seyedi, P. & Syam, S.S. 2017. A survey of healthcare facility location. *Computers & Operations Research,* 79:223-263.

Ahmed, W., Najmi, A., Khan, F. & Aziz, H. 2019. Developing and analyzing framework to manage resources in humanitarian logistics. *Journal of Humanitarian Logistics and Supply Chain Management*, 9(2):270-291.

Aki, O. & Güllü, A. 2016. Obtaining path data from maze image using image processing techniques. (*In* Conference proceedings. International Scientific Conference (UNITECH2016), Gabrovo, Bulgaria. Gabrovo, UNITECH. pp. 326-329).

Alsegård, A. 2017. Solving generic mazes containing multiple materials with an AI agent trained with feature based Q-learning. *AI-maze me*. http://adamalsegard.github.io/lists/TNM095.pdf Date of access: 24 Dec. 2019.

An, H.C., Singh, M. & Svensson, O. 2017. LP-based algorithms for capacitated facility location. *SIAM Journal on Computing*, 46(1):272-306.

Amin, S.H. & Baki, F. 2017. A facility location model for global closed-loop supply chain network design. *Applied Mathematical Modelling*, 41:316-330.

Apte, A. 2010. Humanitarian logistics: A new field of research and action. *Foundations and Trends® in Technology*, *Information and Operations Management*, 3(1):1-100.

Ashlock, D., Lee, C. & McGuinness, C. 2011. Search-based procedural generation of maze-like levels. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):260-273.

Babaei, A. & Shahanaghi, K. 2018. A novel algorithm for identifying and analyzing humanitarian relief logistics problems: studying uncertainty on the basis of interaction with the decision maker. *Process Integration and Optimization for Sustainability*, 2(1):27-45.

Balcik, B. & Beamon, B.M. 2008. Facility location in humanitarian relief. *International Journal of logistics*, 11(2):101-121.

Barnouti, N.H., Al-Dabbagh, S.S.M. & Naser, M.A.S. 2016. Pathfinding in strategy games and maze solving using a search algorithm. *Journal of Computer and Communications*, 4(11):15-25.

Batista e Silva, F., Gallego, J. & Lavalle, C. 2013. A high-resolution population grid map for Europe. *Journal of Maps*, 9(1):16-28.

Bejarano, N.C., Ambrosini, E., Pedrocchi, A., Ferrigno, G., Monticone, M. & Ferrante, S. 2014. A novel adaptive, real-time algorithm to detect gait events from wearable

sensors. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 23(3):413-422.

Bekker, J.F. & Schmid, J.P. 2006. Planning the safe transit of a ship through a mapped minefield. *ORiON*, 22(1):1-18.

Benavides, J.E.H., Corredor, D.E.E., Moreno, R.J. & Hernández, R.D. 2018. Flood fill algorithm dividing matrices for robotic path planning. *International Journal of Applied Engineering Research,* 13(11):8862-8870.

Bíl, M., Vodák, R., Kubeček, J., Bílová, M. & Sedoník, J. 2015. Evaluating road network damage caused by natural disasters in the Czech Republic between 1997 and 2010. *Transportation Research Part A: Policy and Practice*, 80:90-103.

Bitterman, N. & Zimmer, Y. 2018. Portable health care facilities in disaster and rescue zones: characteristics and future suggestions. *Prehospital and Disaster Medicine*, 33(4):411-417.

Bollobás, B. 2013. Modern graph theory. New York, NY: Springer Science & Business Media.

Boonmee, C., Arimura, M. & Asada, T. 2017. Facility location optimization model for emergency humanitarian logistics. *International Journal of Disaster Risk Reduction*, 24:485-498.

Boustan, L.P., Kahn, M.E., Rhode, P.W. & Yanguas, M.L. 2020. The effect of natural disasters on economic activity in US counties: A century of data. *Journal of Urban Economics*, 118:1-26.

Brown, M.F. 1992. Does a cognitive map guide choice in the radial-arm maze? *Journal of Experimental Psychology: Animal Behavior Processes*, 18(1):56.

Cai, Y. & Ji, X. 2018. ASA-routing: A-star adaptive routing algorithm for network-on-chips. (*In* Vaidya, J. & Li, J. *eds*. Conference proceedings. 2018 Algorithms and Architectures for Parallel Processing (ICA3PP 2018), Guangzhou, China. Cham: Springer. pp. 187-198).

Calma, J. 2020. What you need to know about the Australia bushfires. https://www.theverge.com/2020/1/3/21048891/australia-wildfires-koalas-climate-change-bushfires-deaths-animals-damage Date of access: 28 Feb. 2020.

Castleden, R. 2012. The Knossos Labyrinth: a new view of the Palace of Minos' at Knossos. London, LDN: Routledge.

Cavdur, F., Kose-Kucuk, M. & Sebatli, A. 2016. Allocation of temporary disaster response facilities under demand uncertainty: an earthquake case study. *International Journal of Disaster Risk Reduction,* 19:159-166.

Chi, H.Y., Tseng, H.Y., Liu, C.N.J. & Chen, H.M. 2018. Performance-preserved analog routing methodology via wire load reduction. (*In*: Conference proceedings. 2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC 2018), Jeju, Korea. New York: IEEE. pp. 482-487).

Christopher, M. 2016. Logistics & supply chain management. 5th ed. Pearson, UK: Financial Times Publishing.

Correia, I. & Saldanha-da-Gama, F. 2019. Facility location under uncertainty. (*In* Laporte, G., Nickel, S. & Saldanha da Gama, F. *eds*. Location science, Switzerland, Cham: Springer. pp. 185-213).

Daneshjo, N., Kralik, M., Petrovčiková, K., Pajerská, E.D. & Paštéka, M. 2019. Avoiding the obstacles in the robot working zone by using the Lee algorithm. *Coordinates*, 13(2):72-83.

Dascioglu, B.G., Vayvay, O. & Kalender, Z.T. 2019. Humanitarian supply chain management: extended literature review. (*In* Calisir, F., Cevikcan, E. & Akdag, H.C., *eds*. Industrial engineering in the big data era. Switzerland, Cham: Springer. pp. 443-459).

Daskin, M.S. 2011. Network and discrete location: models, algorithms, and applications. 2nd ed. New Jersey, HOB: John Wiley & Sons.

Derdikman, D., Whitlock, J.R., Tsao, A., Fyhn, M., Hafting, T., Moser, M.B. & Moser, E.I. 2009. Fragmentation of grid cell maps in a multicompartment environment. *Nature Neuroscience*, 12(10):1325.

DeWaard, J., Curtis, K.J. & Fussell, E. 2016. Population recovery in New Orleans after Hurricane Katrina: exploring the potential role of stage migration in migration systems. *Population and Environment*, 37(4):449-463.

Diamond, D.M., Park, C.R., Heman, K.L. & Rose, G.M. 1999. Exposing rats to a predator impairs spatial working memory in the radial arm water maze. *Hippocampus*, 9(5):542-552.

Duchoň, F., Babinec, A., Kajan, M., Beňo, P., Florek, M., Fico, T. & Jurišica, L. 2014. Path planning with modified a star algorithm for a mobile robot. *Procedia Engineering*, 96:59-69.

Duran, S., Ergun, Ö., Keskinocak, P. & Swann, J.L. 2013. Humanitarian logistics: advanced purchasing and pre-positioning of relief items. (*In* James, H., *eds.* Handbook of global logistics. New York: Springer. pp. 447-462).

Elgaroui, L., Chamberland, S. & Pierre, S. 2019. A new routing metric for real-time applications in smart cities. (*In* Conference proceedings. 2019 IEEE Sustainability through ICT Summit (StICT 2019), Montreal, Canada. New York: IEEE. pp. 1-6).

Fallah Nezhad, M.S., Zarrinpoor, N. & Pishvaee, M.S. 2017. Design of a reliable facility location model for health service networks. *International Journal of Engineering,* 30(1):75-84.

Fankhauser, P. & Hutter, M. 2016. A universal grid map library: Implementation and use case for rough terrain navigation. (*In* Koubaa, A. *eds*. Robot operating system (ROS). Cham, Germany: Springer. pp. 99-120).

Farahani, R.Z., Fallah, S., Ruiz, R., Hosseini, S. & Asgari, N. 2019. OR models in urban service facility location: a critical review of applications and future developments. *European Journal of Operational Research*, 276(1):1-27.

Febliama, A.B., Fitria, N.D. & Handayani, A.N. 2019. The application of a star (A*) algorithm on the Android-based Pacman adaptation educational game as a learning media for SMK. (*In* Conference proceedings. 2nd International Conference on

Vocational Education and Training (ICOVET 2018), Malang, Indonesia. Paris: Atlantis Press. pp. 200-206).

Foltin, M. 2011. Automated maze generation and human interaction. Brno: Masaryk University (Thesis – PhD).

Fong, C.K.K., Li, M., Lu, P., Todo, T. & Yokoo, M. 2018. Facility location games with fractional preferences. (*In* Conference proceedings. 32nd AAAI Conference on Artificial Intelligence. New Orleans, USA. California: AAAI. pp. 1039-1046).

Frey, D.D., Fukuda, S. & Rock, G. 2011. Improving complex systems today. London, England: Springer.

Gendron, B., Khuong, P.V. & Semet, F. 2017. Comparison of formulations for the two-level uncapacitated facility location problem with single assignment constraints. *Computers & Operations Research,* 86:86-93.

Gibbens, S. 2019. Hurricane Sandy, explained. https://www.nationalgeographic.com/environment/natural-disasters/reference/hurricane-sandy/ Date of access: 12 Dec. 2019.

Golabi, M., Shavarani, S.M. & Izbirak, G. 2017. An edge-based stochastic facility location problem in UAV-supported humanitarian relief logistics: a case study of Tehran earthquake. *Natural Hazards*, 87(3):1545-1565.

Gordon, V.S. & Matley, Z. 2004. Evolving sparse direction maps for maze pathfinding. (*In* Conference proceedings. 2004 Congress on Evolutionary Computation (04TH8753), Portland, USA. New York: IEEE. pp. 835-838).

Griffith, D.A., Boehmke, B., Bradley, R.V., Hazen, B.T. & Johnson, A.W. 2019. Embedded analytics: improving decision support for humanitarian logistics operations. *Annals of Operations Research*, 283:247-265.

Grygorash, O., Zhou, Y. & Jorgensen, Z. 2006. Minimum spanning tree based clustering algorithms. (*In* Conference proceedings. 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06), Arlington, USA. New York: IEEE. pp. 73-81).

Gümüş, A.T. & Çelik, E. 2017. A comprehensive literature review for humanitarian relief logistics in disaster operations management. *Journal of Social Sciences*, 16(31):347-389.

Günay, E.E., Park, K., Kandukuri, S. & Okudan Kremer, G.E. 2019. Facility location selection for the humanitarian needs of refugees. (*In* Romeijn, H.E., Schaefer, A., Thomas, R., *eds*. Conference proceedings. IISE Annual Conference and Expo 2019, Orlando, USA. Georgia: Institute of Industrial and Systems Engineers).

Güneş, E.D., Melo, T. & Nickel, S. 2019. Location problems in healthcare. (*In* Laporte, G., Nickel, S. & Saldanha da Gama, F. *eds*. Location science. Switzerland, Cham: Springer. pp. 657-686).

Gupta, B. & Sehgal, S. 2014. Survey on techniques used in autonomous maze solving robot. (*In* Conference proceedings. 5th International Confluence - The Next Generation Information Technology Summit (Confluence 2014), Noida, India. New York: IEEE pp. 323-328).

Gupta, K.D., Andrei, S. & Ahsan, M. 2019. Solving QR code distortions using a recursive-based backtracking algorithm. *Broad Research in Artificial Intelligence and Neuroscience*, 10(2):14-25.

Han, D.M. & Lim, J.H. 2010. Design and implementation of smart home energy management systems based on Zigbee. *IEEE Transactions on Consumer Electronics*, 56(3):1417-1425.

Harabor, D.D. & Grastien, A. 2011. Online graph pruning for pathfinding on grid maps. (*In* Conference proceedings*.* 25th AAAI Conference on Artificial Intelligence*.* San Francisco, USA. California: AAAI. pp. 1114-1119).

Hart, P.E., Nilsson, N.J. & Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100-107.

Hashim, N.M., Shariff, S. & Deni, S.M. 2017. Capacitated maximal covering location allocation problem during flood disaster. *Advanced Science Letters*, 23(11):11545-11548.

Hendrawan, Y.F. 2020. Comparison of hand follower and dead-end filler algorithm in solving perfect mazes. *Journal of Physics: Conference Series,* 1569(2):22-59.

Iqbal, M., Siahaan, A.P.U., Purba, N.E. & Purwanto, D. 2017. Prim's algorithm for optimizing fiber optic trajectory planning*. International Journal of Scientific Research in Science and Technology*, 3(6):504-509.

Ittmann, H.W. 2005. Humanitarian logistics - a new form of logistics? (*In* Conference proceedings*.* 19th SAIIE & 35th ORSSA 2005 Conference, Vanderbijlpark: South Africa. Pretoria: CSIR).

Jabbar, A.M. 2016. Autonomous navigation of mobile robot based on flood fill algorithm. *Iraqi Journal for Electrical and Electronic Engineering*, 12(1):79-84.

Jahre, M., Persson, G., Kovács, G. & Spens, K.M. 2007. Humanitarian logistics in disaster relief operations. *International Journal of Physical Distribution & Logistics Management.* 37(2):99-114.

Jannu, S. & Jana, P.K. 2016. A grid based clustering and routing algorithm for solving hot spot problem in wireless sensor networks. *Wireless Networks,* 22(6):1901-1916.

Jarník, V. 1930. O jistém problému minimálním. *Práca Moravské Prírodovedecké Spolecnosti*, 6:57-63.

Jeong, K. & Kim, J. 2016. Event-centered maze generation method for mobile virtual reality applications. *Symmetry*, 8(11):120.

Jiang, Y. & Yang, M. 2017. Hardware design of parallel switch setting algorithm for Benes networks. *International Journal of High Performance Systems Architecture*, 7(1):26-40.

Jubair, F. & Hawa, M. 2020. Exploiting obstacle geometry to reduce search time in grid-based pathfinding. *Symmetry*, 12(7):1186.

Karatas, M. & Yakıcı, E. 2018. An iterative solution approach to a multi-objective facility location problem. *Applied Soft Computing*, 62:272-287.

Karlsson, A. 2018. Evaluation of the complexity of procedurally generated maze algorithms. Karlskrona: Blekinge Institute of Technology. (Thesis – Bachelor's degree).

Katsigiannis, A., Anastopoulos, N., Nikas, K. & Koziris, N. 2012. An approach to parallelize Kruskal's algorithm using helper threads. (*In* Conference proceedings. 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum, Shanghai, China. New York: IEEE. pp. 1601-1610).

Kim, J.H., Min, K.S. & Yeo, W.Y. 2014. A design of irregular grid map for large-scale Wi-Fi LAN fingerprint positioning systems. *The Scientific World Journal,* 1-13.

Kim, P.H. 2019. Intelligent Maze Generation. Ohio: The Ohio State University. (Dissertation – PhD).

Konrad, M., Szczot, M., Schüle, F. & Dietmayer, K. 2011. Generic grid mapping for road course estimation. (*In* Conference proceedings. 2011 IEEE Intelligent Vehicles Symposium (IV), Baden-Baden, Germany. New York: IEEE. pp. 851-856).

Kruskal, J.B. 1956. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48-50.

Kurosawa, K., Uchiyama, Y. & Kosako, T. 2020. Development of a numerical marine weather routing system for coastal and marginal seas using regional oceanic and atmospheric simulations. *Ocean Engineering*, 195:1-12.

Law, G. 2013. Quantitative comparison of flood fill and modified flood fill algorithms. *International Journal of Computer Theory and Engineering*, 5(3):503-508.

Lee, C.Y. 1961. An algorithm for path connections and its applications. *IRE Transactions on Electronic Computers*, 3:346-365.

Lee, H.L., Lee, C.F. & Chen, L.H. 2010. A perfect maze based steganographic method. *Journal of Systems and Software*, 83(12):2528-2535.

Lee, M.C., Jan, G.E. & Luo, C.C. 2020. An efficient rectilinear and octilinear steiner minimal tree algorithm for multidimensional environments. *IEEE Access*, 8:48141-48150.

Lerch, J.P., Yiu, A.P., Martinez-Canabal, A., Pekar, T., Bohbot, V.D., Frankland, P.W., Henkelman, R.M., Josselyn, S.A. & Sled, J.G. 2011. Maze training in mice induces MRI-detectable brain shape changes specific to the type of learning. *Neuroimage*, 54(3): 2086-2095.

Lindorfer, M., Backfrieder, C., Kieslich, C., Krösche, J. & Ostermayer, G. 2013. Environmental-sensitive generation of street networks for traffic simulations. (*In* Conference proceedings. 7th European Modelling Symposium (EMS2013), Manchester, UK. New York: IEEE. pp. 457-462).

Liu, Y., Cui, N. & Zhang, J. 2019. Integrated temporary facility location and casualty allocation planning for post-disaster humanitarian medical service. *Transportation Research Part E: Logistics and Transportation Review,* 128:1-16.

Lu, D.N., Nguyen, T.H., Nguyen, D.N. & Nguyen, H.N. 2017. A novel traffic routing method using hybrid ant colony system based on genetic algorithm. (*In* Conference proceedings. 2017 International Conference on Information Networking (ICOIN 2017), Da Nang, Vietnam. New York: IEEE pp. 584-589).

Lukosch, H. & Comes, T. 2019. Gaming as a research method in humanitarian logistics. *Journal of Humanitarian Logistics and Supply Chain Management*. 9(3):352-370.

Lynskey, J., Thar, K., Oo, T.Z. & Hong, C.S. 2019. Facility location problem approach for distributed drones. *Symmetry*, 11(118):1-11.

Manen, S., Guillaumin, M. & Van Gool, L. 2013. Prime object proposals with randomized prim's algorithm. (*In* Conference proceedings. IEEE international conference on computer vision, Sydney, Australia. New York: IEEE. pp. 2536-2543).

Mariano, A., Lee, D., Gerstlauer, A. & Chiou, D. 2013. Hardware and software implementations of Prim's algorithm for efficient minimum spanning tree computation. (*In* Schirner, G., Götz, M., Rettberg, A. & Zanella, M.C. *eds*. Conference proceedings. International Embedded Systems Symposium, Paderborn, Germany. Berlin: Springer. pp. 151-158).

Marın-Plaza, P., Beltrán, J., Hussein, A., Musleh, B., Martın, D., de la Escalera, A. & Armingol, J.M. 2016. Stereo vision-based local occupancy grid map for autonomous navigation in ROS. (*In* Magnenat-Thalmann, N., Richard, P., Linsen, L., Telea, A., Battiato, S., Imai, F. & Braz, J. *eds*. Conference proceedings. 11th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP), Rome, Italy. Portugal: Science and Technology Publications. pp. 703-708).

Massazza, A., Brewin, C.R. & Joffe, H. 2019. The nature of "natural disasters": survivors' explanations of earthquake damage. *International Journal of Disaster Risk Science*, 10(3):293-305.

Merriam-Webster Inc. 2020. Merriam-Webster dictionary. https://www.merriam-webster.com/dictionary/natural%20disaster Date of access: 5 Nov. 2020.

Montemanni, R. & Gambardella, L.M. 2005. A branch and bound algorithm for the robust spanning tree problem with interval data. *European Journal of Operational Research*, 161(3):771-779.

Moret, B.M. & Shapiro, H.D. 1991. An empirical analysis of algorithms for constructing a minimum spanning tree. (*In* Dehne, F., Sack, J. & Santro, N. *eds*. Conference proceedings. Workshop on Algorithms and Data Structures, Ottawa, Canada. Berlin: Springer. pp. 400-411).

Munyaka, J.C.B. & Yadavalli, V.S.S. 2020. Decision support framework for facility location and demand planning for humanitarian logistics. *International Journal of System Assurance Engineering and Management*, 11(5):1-20.

Nanda, A. & Rath, A.K. 2018. Fuzzy A-star based cost effective routing (FACER) in WSNs. (*In* Saeed, K., Chaki, N., Pati, B., Bakshi, S. & Mohapatra, D. *eds*. Progress in Advanced Computing and Intelligent Engineering. Singapore: Springer. pp. 557-563).

Naor, M. & Bernardes, E. 2016. Self-sufficient healthcare logistics systems and responsiveness: ten cases of foreign field hospitals deployed to disaster relief supply chains. *Journal of Operations and Supply Chain Management*, 9(1):1-22.

Nelson, M.J. & Smith, A.M. 2016. ASP with applications to mazes and levels. (*In* Shaker, N., Togelius, J., & Nelson, M.J. *eds*. Procedural content generation in games. Switzerland, Cham: Springer. pp. 143-157).

Nestor, J.A. 2002. A new look at hardware maze routing. (*In* Conference proceedings. 12th ACM Great Lakes symposium on VLSI, New York, USA. New York: Association for Computing Machinery. pp. 142-147).

Neumann, F. & Witt, C. 2010. Ant colony optimization and the minimum spanning tree problem. *Theoretical Computer Science*, 411(25):2406-2413.

Niemczyk, R. & Zawiślak, S. 2020. Review of maze solving algorithms for 2D maze and their visualisation. (*In* Zawiślak, S. & Rysiński, J. *eds*. Engineer of the XXI century. Cham: Springer).

Olano, G. 2019. After the storm: Japan's recovery efforts post-Hagibis. https://www.insurancebusinessmag.com/asia/news/breaking-news/after-the-storm-japans-recovery-efforts-posthagibis-189234.aspx Date of access: 20 Dec. 2019.

Orskin, B. 2017. Japan earthquake & tsunami of 2011: facts and information. https://www.livescience.com/39110-japan-2011-earthquake-tsunami-facts.html Date of access: 13 Jul. 2020.

Ortiz-Astorquiza, C., Contreras, I. & Laporte, G. 2018. Multi-level facility location problems. *European Journal of Operational Research*, 267(3):791-805.

Patil, A. & Nipanikar, S. 2017. Implementation of MATLAB based self-healing grid using Prims algorithm. *International Journal of Scientific Engineering and Technology Research*, 6(21):4052-4055.

Pender County NC. 2019. Pender County Hurricane Florence after action report. North Carolina, USA: Pender County Emergency Management.

Pershin, Y.V. & Di Ventra, M. 2011. Solving mazes with memristors: a massively parallel approach. *Physical Review E,* 84(4):1-7.

Polanczyk, M., Strzelecki, M. & Slot, K. 2012. Lee-algorithm based path replanner for dynamic environments. (*In* Conference proceedings. 2012 International Conference on Signals and Electronic Systems (ICSES 2012). Wroclaw, Poland. New York: IEEE. pp. 1-4).

Pradhananga, R., Mutlu, F., Pokharel, S., Holguín-Veras, J. & Seth, D. 2016. An integrated resource allocation and distribution model for pre-disaster planning. *Computers & Industrial Engineering*, 91:229-238.

Prim, R.C. 1957. Shortest connection networks and some generalizations. *Bell System Technology Journal*, 36(6):1389–1401.

Quirin, A., Cordón, O., Guerrero-Bote, V.P., Vargas-Quesada, B. & Moya-Anegón, F. 2008. A quick MST-based algorithm to obtain pathfinder networks (∞, n− 1). *Journal of the American Society for Information Science and Technology*, 59(12):1912-1924.

Quislant, R., Gutierrez, E., Zapata, E.L. & Plata, O. 2018. Privatizing transactions for Lee's algorithm in commercial hardware transactional memory. *The Journal of Supercomputing*, 74(4):1676-1694.

Raghavan, S., Sahin, M. & Salman, F.S. 2019. The capacitated mobile facility location problem. *European Journal of Operational Research,* 277(2):507-520.

Ranade, S. & Manivannan, P.V. 2019. Quadcopter obstacle avoidance and path planning using flood fill method. (*In* Conference proceedings. 2nd International Conference on Intelligent Autonomous Systems (ICoIAS 2019), Singapore. New York: IEEE. pp. 166-170).

Reddy, A.V., Vinoth, G. & Chiranjeevi, G.N. 2018. Implementation of Lee's algorithm for different routing constraints. (*In* Conference proceedings. 3rd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT 2018), Bengaluru, India. New York: IEEE. pp. 2488-2491).

Reeves, M.C. 2019. An analysis of path planning algorithms focusing on A* and D*. Dayton: University of Dayton. (Dissertation - PhD).

Richter, A.R. 2016. Dynamic facility relocation and inventory management for disaster relief. Berkeley: University of California. (Dissertation - PhD).

Ribeiro, G.M. & Laporte, G. 2012. An adaptive large neighborhood search heuristic for the cumulative capacitated vehicle routing problem. *Computers & operations research*, 39(3):728-735.

Ritchie, H. & Roser, M. 2019. Natural Disasters. https://ourworldindata.org/natural-disasters Date of access: 5 Nov. 2020.

Rodríguez-Espíndola, O., Albores, P. & Brewster, C. 2018. Disaster preparedness in humanitarian logistics: a collaborative approach for resource management in floods. *European Journal of Operational Research*, 264(3):978-993.

Saeedi, S., Paull, L., Trentini, M. & Li, H. 2015. Occupancy grid map merging for multiple robot simultaneous localization and mapping. *International Journal of Robotics and Automation*, 30(2):149-157.

Sebatli, A., Cavdur, F. & Kose-Kucuk, M. 2017. Determination of relief supplies demands and allocation of temporary disaster response facilities. *Transportation Research Procedia,* 22:245-254.

See, K.L., Nayan, N. & Rahaman, Z.A. 2017. Flood disaster water supply: a review of issues and challenges in Malaysia. *International Journal of Academic Research in Business and Social Sciences*, 7(10):525-532.

Seker, S. & Aydin, N. 2020. Hydrogen production facility location selection for Black Sea using entropy based TOPSIS under IVPF environment. *International Journal of Hydrogen Energy*. 45(32):15855-15868.

Shaluf, I.M. 2007. An overview on disasters. *Disaster Prevention and Management: An International Journal*. 16(5):687-703.

Shao, J., Wang, X., Liang, C. & Holguín-Veras, J. 2019. Research progress on deprivation costs in humanitarian logistics. *International Journal of Disaster Risk Reduction,* 42:1-12.

Sharma, B., Ramkumar, M., Subramanian, N. & Malhotra, B. 2019. Dynamic temporary blood facility location-allocation during and post-disaster periods. *Annals of Operations Research,* 283(1):705-736.

Shavarani, S.M. 2019. Multi-level facility location-allocation problem for post-disaster humanitarian relief distribution. *Journal of Humanitarian Logistics and Supply Chain Management.* 9(1):70-81.

Shavarani, S.M., Nejad, M.G., Rismanchian, F. & Izbirak, G. 2018. Application of hierarchical facility location problem for optimization of a drone delivery system: a case study of Amazon Prime Air in the city of San Francisco. *The International Journal of Advanced Manufacturing Technology*, 95(9-12):3141-3153.

Sigala, I.F. & Wakolbinger, T. 2019. Outsourcing of humanitarian logistics to commercial logistics service providers. *Journal of Humanitarian Logistics and Supply Chain Management*. 9(1):47-69.

Silva, S., Duarte, D., Barradas, R., Soares, S., Valente, A. & Reis, M.J. 2017. Arduino recursive backtracking implementation, for a robotic contest. (*In* Silva, M.F., Virk, G.S., Tokhi, M.O., Malheiro, B., Ferreira, P. & Guedes, P., *eds*. Conference proceedings*.* 20th International Conference on CLAWAR 2017, Porto, Portugal. Singapore: World Scientific. pp. 169-178).

Siregar, B., Nababan, E.B., Rumahorbo, J.A., Andayani, U. & Fahmi, F. 2018. Nearby search indekos based android using A-star (A*) algorithm*. Journal of Physics: Conference Series.* 978(1):1-6.

Soukup, J. 1992. Maze router without a grid map. (*In* Conference proceedings*.* 1992 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). California, USA. New York: IEEE pp. 382-385).

Sturtevant, N.R. 2012. Benchmarks for grid-based pathfinding. *IEEE Transactions on Computational Intelligence and AI in Games,* 4(2):144-148.

Sudhakar, T.D. & Srinivas, K.N. 2011. Power system restoration based on Kruskal's algorithm. (*In* Conference proceedings*.* 2011 1st International Conference on Electrical Energy Systems. California, USA. New York: IEEE. pp. 281-287).

Sudhakar, T.D. & Srinivas, K.N. 2011. Power system reconfiguration based on Prim's algorithm. (*In* Conference proceedings*.* 2011 1st International Conference on Electrical Energy Systems*.* California, USA. New York: IEEE. pp. 12-20).

Sudhakar, T.D. & Srinivas, K.N. 2010. Prim's algorithm for loss minimization and service restoration in distribution networks. *International Journal of Electrical and Computer Engineering,* 2(1):43-62.

Sukumar, T. & Santha, K.R. 2012. Maze based data hiding using back tracker algorithm. *International Journal of Engineering Research and Applications*. 2248-9622.

Sun, D. & Li, M. 2016. Evaluation function optimization of A-star algorithm in optimal path selection. *Technical Magazine of the Faculty of Engineering Universidad del Zulia*, 39(4):105-111.

Sydnor, S., Niehm, L., Lee, Y., Marshall, M. & Schrank, H. 2017. Analysis of post-disaster damage and disruptive impacts on the operating status of small businesses after Hurricane Katrina. *Natural Hazards*, 85(3), pp.1637-1663.

Taylor, B.W. 2019. Introduction to management science. 13th ed. Harlow, UK: Pearson.

Tjiharjadi, S. & Setiawan, E. 2016. Design and implementation of a path finding robot using flood fill algorithm. *International Journal of Mechanical Engineering and Robotics Research*, 5(3):180-185.

Trivedi, A. & Singh, A. 2018. Facility location in humanitarian relief: a review. *International Journal of Emergency Management,* 14(3):213-232.

Turkoglu, D.C. & Genevois, M.E. 2019. A comparative survey of service facility location problems. *Annals of Operations Research*, 292:1-70.

Walker, R.J. 1960. An enumerative technique for a class of combinatorial problems. (*In* American Math Society, *eds*. Conference proceedings. Proceedings of Symposia in Applied Mathematics, Indonesia. Providence, American Mathematical Society. pp. 91-94).

Wang, S., Austin, P. & Bell, S. 2011. It's a maze: the pore structure of bread crumbs. *Journal of Cereal Science*, 54(2):203-210.

Wang, W., Huang, Y. & Guo, S. 2011. Design and implementation of GPU-based prim's algorithm. *International Journal of Modern Education and Computer Science*, 3(4):55.

Wang, W.C. & Hsieh, M.C. 2018. Applying Prim's algorithm to identify isolated areas for natural disaster prevention and protection. *Engineering,* 10(7):417-431.

Wichapa, N. & Khokhajaikiat, P. 2017. Solving multi-objective facility location problem using the fuzzy analytical hierarchy process and goal programming: a case study on infectious waste disposal centers. *Operations Research Perspectives,* 4:39-48.

Widera, A., Lechtenberg, S., Gurczik, G., Bähr, S. & Hellingrath, B. 2017. Integrated logistics and transport planning in disaster relief operations. (*In* Comes, T., Bénaben, F., Hanachi, C., Lauras, M., & Montarnal, A., *eds*. Conference proceedings. 14th International Conference on Information Systems for Crisis Response and Management. Albi, France. pp. 752-764).

Wu, C.M., Liaw, D.C. & Lee, H.T. 2018. A method for finding the routes of mazes. (*In* Conference proceedings. International Automatic Control Conference (CACS 2018), Taoyuan, Taiwan. New York: IEEE. pp. 1-4).

Wu, Y.R., Tsai, M.C. & Wang, T.C. 2005. Maze routing with OPC consideration. (*In* Conference proceedings. Asia and South Pacific Design Automation Conference (ASP-DAC 2005), Shanghai, China. New York: IEEE. pp. 198-203).

Yadav, V., Bhurjee, A.K., Karmakar, S. & Dikshit, A.K. 2017. A facility location model for municipal solid waste management system under uncertain environment. *Science of the Total Environment,* 603:760-771.

Yu, G., Haskell, W.B. & Liu, Y. 2017. Resilient facility location against the risk of disruptions. *Transportation Research Part B: Methodological*, 104:82-105.

Zehetmeier, D., Böttcher, A., Brüggemann-Klein, A. & Thurner, V. 2019. Defining the competence of abstract thinking and evaluating CS-students' level of abstraction. (*In*: Conference proceedings. 52nd Hawaii International Conference on System Sciences, Hawaii, USA. Hawaii: HICSS. pp. 7642-7651).

Zhang, A., Li, C. & Bi, W. 2016. Rectangle expansion A∗ pathfinding for grid maps. *Chinese Journal of Aeronautics*, 29(5):1385-1396.

Zhang, Y. 2018. Deep reinforcement learning on 1-layer circuit routing problem. Illinois, University of Illinois at Urbana-Champaign. (Dissertation - PhD).

Zheng, C., Liu, H., Ge, M. & Liu, Y. 2019. A novel maze representation approach for finding filled path of a mobile robot. (*In* Conference proceedings. International conference on computer, network, communication and information systems (CNCI 2019), Qingdao, China. Paris: Atlantis Press. pp. 664-673).

Zokaee, S., Bozorgi-Amiri, A. & Sadjadi, S.J. 2016. A robust optimization model for humanitarian relief chain design under uncertainty. *Applied Mathematical Modelling*, 40(17-18):7996-8016.

Annexure A: New Orleans Matrix

In Section 5.3.2, details of the development of a matrix to represent the damage caused by Hurricane Katrina are presented. In Table 5.1 in Section 5.3.2, the matrix value assignment is explained. The complete 70x70 matrix representing the damage to the New Orleans area is presented in this Annexure. The matrix is used to generate a maze for the New Orleans area. See Algorithm 5.1 in Section 5.3.3.

{4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,0,0,0,0,0,0,0,0,0,0,0,3,4,4,4,0,0,0,0,0,0},
{4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,4,0,0,0,0,0,3,0},
{4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,0,4,4,4,4,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{4,4,4,4,4,4,4,4,4,4,4,4,4,0,0,0,0,4,4,4,4,4,0,0,0,0,0,0,0,0,4,4,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,0,0,0,0,0,0,0,0,0,0,0,1},
{4,4,4,4,4,4,4,4,4,4,4,4,4,0,0,0,0,0,0,0,0,0,0,0,0,3,1,0,0,0,0,0,0,0,1,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,3,0,0,0,0,0,0,0,0},
{4,4,4,4,4,4,4,4,4,4,4,4,4,0,0,0,0,0,1,0,0,0,0,0,1,2,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,3,3,1,0,0,0,0,0,0},
{4,4,4,4,4,4,4,4,4,4,4,4,4,0,0,0,0,0,2,0,0,3,1,0,1,0,0,0,0,3,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,1,3,0,0,1,0,2,3,1,0,3,0,0,0,2,3,1,0,0,0,0,0,0},
{4,4,4,4,4,4,4,4,4,4,4,0,0,0,0,0,1,0,0,0,1,2,2,0,0,2,0,0,3,2,0,0,0,0,0,3,0,0,1,0,0,0,1,0,0,3,1,2,0,0,2,0,0,1,1,3,0,1,0,0,0,0,0,3,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,1,0,0,3,0,2,1,0,2,3,1,0,0,1,3,1,0,1,0,0,3,2,0,0,0,0,0,0,1,0,0,2,0,0,3,0,0,0,3,0,0,0,2,0,2,0,3,3,0,0,0,0,3,0,0,0,0,0,0},
{0,0,0,0,0,0,3,3,1,2,0,0,0,1,2,2,0,3,3,3,1,3,2,3,3,0,2,0,0,0,2,1,0,0,0,0,0,0,1,0,3,3,3,1,3,1,2,0,0,0,0,0,3,0,0,1,2,3,1,0,0,0,2,1,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,1,2,3,2,3,3,2,0,3,0,0,0,0,0,0,0,0,0,0,0,2,0,3,2,2,2,1,3,3,1,0,2,0,0,0,2,0,2,1,3,3,3,0,0,3,1,1,0,0,0,0,0},
{1,0,0,1,0,0,0,1,0,3,1,1,0,0,3,1,2,1,3,3,3,2,1,0,2,2,1,0,0,0,0,0,0,0,0,0,3,1,0,2,2,0,3,1,0,3,2,1,0,0,1,3,1,0,0,3,0,1,2,3,1,0,2,3,1,0,0,0,0,0},
{2,0,0,0,0,2,0,0,2,2,3,3,1,0,0,0,1,0,1,3,3,2,0,3,3,3,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,2,0,0,3,3,0,2,2,2,0,1,0,3,2,0,2,3,2,2,3,3,0,0,0,0,0},
{0,0,3,0,0,0,1,1,0,3,1,2,0,0,3,3,1,2,3,3,1,1,2,2,2,3,1,0,0,0,0,0,0,0,0,0,1,2,0,3,3,1,0,2,1,0,2,0,0,0,0,0,0,0,2,0,2,0,0,2,2,1,2,3,1,1,0,0,0,0},
{0,0,0,0,0,0,2,2,0,3,0,1,1,0,2,0,3,0,2,3,1,2,3,0,3,1,0,0,0,0,0,0,0,0,0,0,2,0,0,3,3,2,1,0,0,2,1,2,0,0,0,0,0,0,3,1,2,0,0,2,3,3,3,3,3,3,0,0,0,0},
{0,0,0,0,0,0,0,0,3,3,0,2,1,3,3,2,1,0,1,2,2,2,0,3,3,1,0,0,0,0,0,0,0,0,0,0,2,2,0,1,3,1,0,0,0,1,0,3,1,0,1,1,3,2,2,0,0,0,0,3,2,3,3,3,3,2,3,1,0,0},
{0,0,0,0,0,0,0,0,2,0,0,2,1,1,2,2,0,2,0,1,0,3,1,1,2,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,2,1,0,0,0,1,0,2,0,3,1,0,2,1,0,2,3,1,0,2,3,3,2,2,0,0,0,0,0,0},
{3,0,0,0,0,0,0,0,0,2,0,2,0,3,3,3,3,1,0,3,3,3,1,0,2,2,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,2,0,0,2,0,0,2,0,1,0,2,0,0,0,2,2,0,0,0,0,0,2,0,0,0,0,2,0,0,0,0},
{0,0,0,0,0,1,0,0,0,1,2,0,0,2,2,3,3,0,1,2,3,3,3,3,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,1,1,0,0,0,0,0,0,0,0,0,0,0,0,3,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,3,3,3,3,2,3,3,3,1,0,0,0,0,0,0,0,0,0,0,0,1,2,1,0,3,0,0,0,0,0,0,0,0,0,0,0,3,1,0,2,0,1,0,0,0,2,1,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,3,3,3,2,3,3,3,0,3,2,3,3,2,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,1,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,3,3,1,3,2,1,3,0,3,3,3,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,1,0,0,1,0,0,0,0,0,0,0,0,2,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,2,0,2,0,0,0,0,0,2,2,2,2,0,0,3,3,3,2,3,1,3,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,1,2,1,0,3,3,1,0,0,0,0,0,2,0,0,0,0,3,3,1,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,1,0,0,0,2,1,1,1,1,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,0,2,0,0,2,0,0,0,0,0,0,0,3,1,0,3,3,0,0,0,2,1,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,1,0,0,0,2,1,2,1,0,3,1,2,2,3,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,1,0,3,0,3,2,0,2,2,2,0,3,0,0,0,0,0,0,0,2,0,0,0,0},
{0,0,0,2,0,2,0,0,0,0,0,0,3,3,1,0,2,2,0,3,3,3,3,3,1,0,0,0,3,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,2,1,0,0,0,0,0,3,0,2,1,0,1,0,0,0,0,0,0,0},
{0,0,0,0,0,0,3,0,0,0,0,0,2,2,0,0,2,2,0,2,3,3,2,2,2,0,0,0,0,0,0,0,0,0,0,0,2,0,0,2,2,0,0,1,0,0,0,1,0,0,2,1,0,0,0,0,0,0,0,2,0,0,0,1,0,0,1,0,1,0,0,0},
{0,0,0,0,0,0,2,0,1,0,0,0,2,0,0,0,0,0,0,1,2,0,0,0,0,0,0,0,0,0,0,0,0,3,0,0,0,1,0,3,0,0,0,2,2,0,3,1,0,2,0,0,0,0,0,0,0,0,0,0,2,0,0,1,1,2,0,0,0,0},
{0,0,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,2,0,0,2,0,0,2,0,0,3,3,3,1,0,0,0,0,0,1,0,0,0,1,0,0,2,0,2,0,0,0,0,0,0},
{0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,0,1,0,0,0,3,1,1,0,0,0,0,2,0,0,2,0,0,1,0,0,0,0,1,0,0,0,0},
{0,1,2,2,0,3,3,1,0,0,0,0,0,0,0,0,2,0,0,0,0,2,0,2,0,1,0,0,3,2,0,0,0,0,0,0,0,0,0,0,3,0,0,2,2,1,0,3,0,1,0,0,3,1,3,1,0,0,0,0,2,0,0,0,2,3,0,0,0,0},
{3,0,0,0,0,2,2,0,0,0,1,0,0,0,0,0,0,0,0,0,0,3,1,0,0,1,0,0,3,1,3,0,0,0,2,1,3,1,0,1,0,0,0,0,0,0,2,2,3,0,0,0,3,0,1,0,3,2,1,0,0,0,1,0,0,0,1,0,0,0},
{0,1,2,0,0,0,0,3,0,0,2,0,0,0,0,0,0,0,0,0,0,3,1,3,1,0,0,0,2,0,1,1,0,2,0,0,3,1,2,1,3,1,1,2,3,2,0,0,2,0,3,2,0,2,1,1,2,0,0,0,0,1,0,0,0,2,0,0,0,0},
{3,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,3,0,0,3,1,0,0,0,0,3,0,1,0,0,2,3,2,0,1,0,2,0,1,0,2,0,0,0,0,3,0,0,2,2,1,0,0,0,0,2,2,0,1,0,0,3,1,0,0,0,0},
{2,2,0,0,0,0,0,2,0,0,0,3,0,0,0,0,0,0,0,0,2,0,1,2,0,2,0,0,0,0,0,0,1,3,3,2,0,2,1,0,0,0,3,3,0,0,1,2,2,2,3,3,3,3,2,2,2,0,0,0,3,3,2,0,0,2,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,1,0,2,2,3,1,0,0,3,0,0,0,0,2,0,0,1,0,2,0,2,0,0,0,0,2,3,2,0,0,3,3,2,2,3,0,0,1,0,0,0,1,3,2,1,0,0,0,1,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,1,0,2,0,3,1,0,1,2,0,0,0,1,0,1,0,0,0,0,2,3,0,3,0,1,2,3,2,3,3,2,0,1,0,1,0,2,1,0,0,0,1,2,2,0,0,2,2,0,0,0,3,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,3,1,1,3,3,1,2,0,0,0,0,0,0,0,3,1,2,0,0,0,0,0,0,2,0,1,2,0,3,3,2,3,0,1,3,3,3,1,3,1,0,2,0,2,0,0,0,0,0,0,0,0,2,3,2,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,1,2,3,2,3,2,2,1,1,0,0,0,0,0,3,2,2,0,0,0,2,0,2,1,0,0,0,0,0,0,3,1,2,0,2,3,3,2,2,2,0,0,0,2,1,0,0,2,0,0,0,0,0,0,0,2,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,3,3,0,3,1,1,0,0,0,0,0,0,0,3,1,3,3,0,0,0,2,0,3,1,0,1,0,0,0,2,1,0,3,2,2,0,3,1,0,1,3,0,3,1,0,0,0,1,0,0,0,2,0,3,3,1,0},
{0,0,0,3,1,0,0,0,2,3,3,1,2,1,2,0,0,2,0,2,0,0,0,0,0,0,2,0,2,2,0,0,1,0,0,2,0,2,0,2,2,0,3,3,0,3,0,0,3,0,0,2,2,3,1,1,0,3,1,0,0,0,3,1,0,0,2,2,0,0},
{0,0,0,2,0,0,0,0,0,2,2,2,3,0,1,2,0,0,0,0,0,0,0,0,2,0,0,0,0,0,2,0,2,0,0,1,0,2,0,0,0,0,2,3,1,0,3,0,0,2,0,3,1,3,0,1,0,0,2,0,0,0,2,2,2,0,0,2,0,0,1},
{5,5,0,0,0,0,0,0,0,0,0,2,0,2,3,2,1,0,0,3,1,0,0,0,3,1,3,0,0,1,0,0,0,0,3,3,0,1,3,0,2,0,0,0,0,0,2,3,1,0,0,0,1,0,0,2,0,0,0,0,0,0,1,0,2,0,0,0,0,0,0},
{5,5,0,0,0,0,0,0,0,0,0,2,1,2,2,3,3,3,2,2,1,3,1,2,0,2,0,2,0,0,0,0,0,2,2,0,0,2,0,0,1,0,0,0,0,3,3,3,1,2,0,0,4,4,4,4,4,4,4,0,0,0,0,0,0,0,0,2,1},
{0,0,0,0,0,0,0,3,0,0,0,0,0,0,0,3,2,2,2,3,2,2,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,2,2,2,0,0,4,4,4,4,4,4,4,4,4,4,4,4,4,0,0,0,0,0,0,2,0},
{4,0,0,0,0,0,1,0,0,0,3,3,1,0,0,2,0,3,0,2,0,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,3,1,0,0,2,2,0,0,0,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,0,0},
{4,4,0,0,0,1,2,2,2,0,1,2,1,0,3,0,0,2,2,0,2,2,1,0,0,1,0,0,0,0,0,0,2,0,5,5,0,0,5,5,0,0,0,0,0,0,0,0,4,4,4,0,0,1,0,0,0,0,0,0,0,4,4,4,4,4,4,4,4,4},
{4,4,4,0,0,3,3,1,0,3,3,1,1,0,2,1,3,1,0,3,2,2,0,0,1,2,0,0,0,3,1,0,0,0,5,5,0,0,5,5,0,0,2,0,0,0,0,0,4,4,0,0,2,2,0,0,0,2,0,0,2,0,0,0,4,4,4,4,4,4},
{4,4,4,0,0,2,2,0,0,2,2,2,3,3,0,1,2,0,0,1,3,0,1,2,1,0,0,3,1,2,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,4,4,0,0,0,0,0,0,0,3,1,0,0,0,0,0,0,0,0,0,4,4,4},
{4,4,4,4,0,0,0,0,0,0,0,0,3,1,2,0,1,3,1,0,2,2,3,3,3,3,1,2,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,4,4,0,0,0,0,1,0,2,0,0,0,0,0,0,0,0,0,0,0,0,4},
{4,4,4,4,0,0,0,0,0,0,0,0,3,2,0,3,1,2,2,3,0,3,2,3,1,3,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,4,4,0,0,0,0,2,0,2,0,0,0,0,0,0,0,0,0,0,0,0,0},
{4,4,4,4,0,0,0,1,0,2,0,3,1,0,0,3,1,0,2,0,2,3,3,3,3,3,0,0,0,2,0,0,2,1,0,3,1,0,0,0,0,0,0,0,0,0,0,0,0,0,4,4,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,0},
{4,4,4,4,0,0,0,0,1,0,0,3,1,1,0,2,0,0,2,0,3,2,3,3,2,3,3,0,1,0,2,1,2,0,1,2,2,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{4,4,4,4,4,0,0,1,2,0,2,2,0,3,2,0,2,1,0,2,2,3,2,2,3,3,2,2,1,0,0,0,0,2,0,3,1,2,0,0,0,0,0,0,0,0,0,0,2,1,4,4,0,0,2,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0},
{4,4,4,4,4,0,1,2,0,0,0,0,0,0,0,0,2,1,0,0,0,2,1,2,0,1,2,3,0,0,2,2,2,3,3,2,2,0,0,0,0,0,0,0,0,0,0,0,0,4,4,4,0,0,0,2,0,0,0,0,0,0,1,0,0,0,0,0,0,0},
{4,4,4,4,4,0,2,1,0,2,0,2,0,0,0,0,0,0,2,1,3,5,5,0,1,0,0,3,1,0,2,0,2,2,2,1,0,2,0,0,0,0,0,0,0,0,0,0,0,4,4,4,0,0,0,0,0,1,0,0,0,0,0,3,1,0,0,0,0,0,0},
{4,4,4,4,4,0,0,2,1,0,0,0,0,0,0,0,1,0,0,1,2,5,5,0,0,3,3,1,1,0,0,1,1,2,1,2,0,0,0,0,0,1,3,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,2,1},
{4,4,4,4,4,0,0,2,3,0,0,0,1,0,0,2,0,2,1,2,0,2,1,0,2,0,2,3,1,3,1,2,0,3,3,0,1,0,0,2,2,0,1,0,0,0,0,0,0,4,4,4,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,2,0},
{4,4,4,4,4,0,2,0,1,0,0,0,0,0,0,0,0,0,3,1,0,0,0,3,0,2,3,3,1,2,2,0,0,2,1,2,2,0,0,0,0,2,0,0,0,1,0,0,4,4,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{4,4,4,4,4,0,0,2,0,2,1,0,0,1,0,0,0,0,1,0,0,1,1,2,0,0,2,2,0,2,0,3,3,3,3,0,2,0,0,0,1,1,0,3,1,2,0,0,4,4,4,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0},
{4,4,4,4,4,0,0,0,3,1,0,0,0,1,0,2,1,0,2,0,2,0,0,0,1,0,1,3,1,0,0,3,3,1,3,1,0,0,0,2,2,0,1,2,0,0,0,0,4,4,4,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,3,0},
{4,4,4,4,4,0,0,2,2,0,0,0,0,2,0,0,2,0,0,2,0,0,1,0,3,2,0,2,0,2,2,0,2,0,2,0,0,0,0,1,1,0,0,0,0,0,0,4,4,4,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0},
{4,4,4,4,4,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,2,0,2,0,3,1,2,0,0,0,0,2,0,0,0,0,1,0,0,2,0,0,0,0,0,0,4,4,4,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{4,4,4,4,4,0,2,0,0,0,0,0,0,0,0,2,1,1,0,0,0,3,2,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,1,2,1,0,0,0,0,0,0,4,4,4,4,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,2,0,0},
{4,4,4,4,4,0,0,0,0,0,0,0,3,1,3,1,2,3,0,1,2,0,0,0,0,0,0,0,0,0,1,2,0,0,0,0,2,2,0,2,0,1,0,0,0,4,4,4,4,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0},
{4,4,4,4,4,0,0,0,0,0,0,0,2,0,2,0,3,3,3,3,2,0,0,1,1,5,5,3,1,2,0,0,0,2,0,0,3,1,0,2,0,0,0,0,4,4,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0},
{4,4,4,4,4,0,0,0,0,0,0,0,2,2,0,2,2,2,0,2,0,1,2,1,2,5,5,3,1,0,0,0,1,0,0,0,2,0,0,0,4,4,4,0,0,4,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0},
{4,4,4,4,4,0,0,0,0,5,5,0,0,0,0,2,0,0,0,0,0,2,0,0,1,3,2,3,1,0,0,0,1,0,0,0,0,4,4,4,4,4,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0},
{4,4,4,4,4,4,0,0,0,5,5,0,2,1,0,0,0,0,1,0,0,0,0,2,0,0,0,2,0,0,0,0,0,0,4,4,4,4,4,4,4,4,4,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,0,0,0,0,0,0},
{4,4,4,4,4,4,4,0,0,0,0,0,0,2,0,0,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,4,4,4,4,4,4,4,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0}.

Annexure B: C# code used in software solution

In Chapter 5, a software application that was developed to illustrate how various algorithms may be implemented to assist humanitarian relief activities is presented. The C# software code to implement the different algorithms is presented in this Annexure.

Code to develop a maze based on a matrix – see Algorithm 5.1 in Section 5.3.3

```
private Graphics g;
int[,] emptyList = new int[70,70];
public Coordinate temp = new Coordinate(35, 39, 0);  //2d array [row(y), colummm(x)]
public int row, column, mazeValue = 0;
List<List<Coordinate>> aStarHospitalPaths = new List<List<Coordinate>>();
private int[,] tempMaze = new int[70, 70];

private void DrawMaze()
{
   pictureBox1.Image = new Bitmap(680, 680);//684
   g = Graphics.FromImage(this.pictureBox1.Image);
   Pen gridPen = new Pen(Color.Black, 2);
   SolidBrush gridBrush = new SolidBrush(Color.Black);
   SolidBrush hospitalBrush = new SolidBrush(Color.Blue);
   SolidBrush startingBrush = new SolidBrush(Color.Green);
   float multiplierWidth = (float)pictureBox1.Image.Width / 70;
   float multiplierHeight = (float)pictureBox1.Image.Height / 70;
   int mazeL = maze.GetLength(0);
   // iterate through all i,j locations in the maze
   for (int j = 0; j < maze.GetLength(0); j++)
   {
      for (int i = 0; i < mazeL; i++)
      {
         if (maze[j, i] == 2)
         {
            g.DrawLine(gridPen, i * multiplierWidth, j * multiplierHeight, (i + 1) *
multiplierWidth, j * multiplierHeight);
         }
         else if (maze[j, i] == 3)
         {
         g.DrawLine(gridPen, (i) * multiplierWidth, j * multiplierHeight, (i) *
multiplierWidth, (j + 1) * multiplierHeight);
         g.DrawLine(gridPen, (i) * multiplierWidth, j * multiplierHeight, (i + 1) *
multiplierWidth, (j) * multiplierHeight);
         }
         else if (maze[j, i] == 1) g.DrawLine(gridPen, i * multiplierWidth, j *
multiplierHeight, (i) * multiplierWidth, (j + 1) * multiplierHeight);
         else if (maze[j, i] == 4) g.FillRectangle(gridBrush, i * multiplierWidth, j *
multiplierHeight, multiplierWidth, multiplierHeight);
         else if (maze[j, i] == 5)
         {
            g.FillRectangle(hospitalBrush, i * multiplierWidth, j * multiplierHeight,
multiplierWidth, multiplierHeight);
         }
```

```
      }
   }
}
```

Code to develop a maze using modified Kruskal's algorithm – see Algorithm 5.2 in Section 5.3.4

```
private void DrawKruskalMaze()
{
   Array.Copy(maze, tempMaze,4900);
   int row = 0;
   int column = 0;
   Random randomNumber = new Random();
   int[,] wallArray = new int[70, 70];
   int direction = 0;
   List<List<string>> sets = new List<List<string>>();
   List<string> tempList = new List<string>();
   List<string> tempList2 = new List<string>();
   row = randomNumber.Next(70);//start row
   column = randomNumber.Next(70);//start column
   List<int[]> transitions = new List<int[]>();

   for (int i = 0; i < 70; i++)// Firstly create an array full of walls
   {
      for (int j = 0; j < 70; j++)
      {
         wallArray[i, j] = 3;
         sets.Add(new List<string> {i+","+j});
      }
   }

   while (sets.Count>1)
   {
      while (column != 80) //will always run untill break
      {
         direction = randomNumber.Next(4);
         //Test if wall was already removed
         if (transitions.Contains(new int[] { row, column, direction }))
         {
            row = randomNumber.Next(70);//start row
            column = randomNumber.Next(70);//start column
            break;
         }
         //up
         if ((direction == 0) && ((row - 1) > -1))
         {
            foreach (List<string> set in sets)
            {
               if (set.Contains(row + "," + column) == true)
               {
                  tempList = set;
               }
```

```
            if (set.Contains((row - 1) + "," + column) == true)
            {
               tempList2 = set;
            }
         }
         if (tempList != tempList2)
         {
            sets.Remove(tempList);
            sets.Remove(tempList2);
            sets.Add(tempList.Concat(tempList2).ToList());
            if (wallArray[row, column] == 3)
               wallArray[row, column] = 1;
            else if (wallArray[row, column] == 2)
               wallArray[row, column] = 0;
         }
         transitions.Add(new int[] { row, column, 0 });
         transitions.Add(new int[] { row-1, column, 2 });
         row = randomNumber.Next(70);//start row
         column = randomNumber.Next(70);//start column
         break;
      }
      else if ((direction == 1) && (column + 1 < 70))
      {//right

//The same process is followed for the remaining directions: Left, right, down.

      }

   }
}
tempList = sets.First();
//Combines matrix maze and Kruskal maze
for (int i = 0; i < 70; i++)
{
   for (int j = 0; j < 70; j++)
   {
      if (maze[i, j] == 0)
      {
         if ((i - 1 >= 0) && (j - 1 >= 0) && (j + 1 <= 69) && (i + 1 <= 69))
         {
            if((maze[i,j-1] == 0) && (maze[i-1, j] == 0) && (maze[i, j+1] == 0) &&
(maze[i+1, j] == 0))
            {
               tempMaze[i, j] = wallArray[i, j];
            }
         }
      }
   }
}
Array.Copy(tempMaze, maze, 4900);
DrawMaze();
}
```

Code to solve a maze using the A-star algorithm – see Algorithm 3.2 in Section 3.3.2.

```
public List<Coordinate> Solve(Coordinate startTemp)
{
   List<Coordinate> nextToVisit = new List<Coordinate>();
   int mazeSize = 70;
   int valUp, valLeft, valRight, valDown;
   float multiplierWidth = (float)pictureBox1.Image.Width / mazeSize;
   float multiplierHeight = (float)pictureBox1.Image.Height / mazeSize;
   bool validLeft, validRight, validUp, validDown = false;
   mazeSize = 69;
   Coordinate cur;
   List<Coordinate> visited = new List<Coordinate>();
   SolidBrush movement = new SolidBrush(Color.LightSteelBlue);
   SolidBrush movementVisited = new SolidBrush(Color.Blue);

//A-star algorithm start
   int startRow = startTemp.getRow(), startColumn = startTemp.getColumn();
   float curgValue = 0, successorgCost, curfValue = 0;
   Coordinate smallest = new Coordinate();
   //nextToVisit : open list
   //visited     : closed list
   if (radioButtonAstar.Checked)
   {
      int[,] activeH = ActiveHospitals(); //gets all the active hospials
      Coordinate startCoordinate = startTemp;
      startRow = startCoordinate.getRow();
      startColumn = startCoordinate.getColumn();

      for (int h = 0; h < activeH.Length/2; h++)//runs for each hospital selected
      {
         int goalRow = activeH[h,0];
         int goalColumn = activeH[ h,1];
         nextToVisit.Clear();
         visited.Clear();
         nextToVisit.Add(startCoordinate);
         nextToVisit.First().setfValue(H(goalRow, goalColumn,
nextToVisit.First().getRow(), nextToVisit.First().getColumn()));
         nextToVisit.First().setgValue(0);

         while (nextToVisit.Count != 0)
         {
            List<Coordinate> tempNextToVisit = nextToVisit;
            cur = SmallestfValue(tempNextToVisit);
            nextToVisit.RemoveAt(getNodeIndex(nextToVisit, cur));
            row = cur.getRow();
            column = cur.getColumn();
            mazeValue = cur.getMazeValue(); //value of maze in maze array
            curfValue = cur.getfValue();
            //test bounds
            validUp = (row - 1 >= 0);
            validLeft = (column - 1 >= 0);
            validRight = (column + 1 <= mazeSize);
```

```
            validDown = (row + 1 <= mazeSize);
            g.FillRectangle(movementVisited, ((column) * multiplierWidth) + 1, (row *
multiplierHeight) + 1, multiplierWidth - 2, multiplierHeight - 2);

            if (validUp)
            {
               valUp = maze[row - 1, column];
               if (new int[] { 0, 1, }.Contains(mazeValue) && valUp != 4)
               {
                  successorgCost = cur.getgValue() + 1;//Nature of maze: distance
between neighbor nodes = 1
                  curfValue = H(goalRow, goalColumn, row - 1, column) + successorgCost;
                  startTemp = new Coordinate(row - 1, column, valUp, cur,
successorgCost, curfValue);

                  if (ContainsNode(nextToVisit, startTemp))
                  {
                     if ((G(startTemp) <= successorgCost))
                     {
                        nextToVisit[getNodeIndex(nextToVisit,
startTemp)].setgValue(G(startTemp));
                        nextToVisit[getNodeIndex(nextToVisit,
startTemp)].setfValue(H(goalRow, goalColumn, row - 1, column) + G(startTemp));
                        nextToVisit[getNodeIndex(nextToVisit,
startTemp)].setParent(cur);
                     }
                  }
                  else if (ContainsNode(visited, startTemp))
                  {
                     if ((G(startTemp) < visited[getNodeIndex(visited,
startTemp)].getgValue()))
                     {
                        startTemp.setgValue(G(startTemp));
                        startTemp.setfValue(G(startTemp) + H(goalRow, goalColumn,
startTemp.getRow(), startTemp.getColumn()));
                        startTemp.setParent(cur);
                        nextToVisit.Add(startTemp);
                        visited.RemoveAt(getNodeIndex(visited, startTemp));
                     }
                  }
                  else
                  {
                     nextToVisit.Add(startTemp);
                     g.FillRectangle(movement, (startTemp.getColumn() * multiplierWidth)
+ 1, (startTemp.getRow() * multiplierHeight) + 1, multiplierWidth - 2,
multiplierHeight - 2);
                     pictureBox1.Refresh();
                  }
                  if ((valUp == 5) && (ValidHospital(row - 1, column)))
                  {
                     aStarHospitalPaths.Add(BacktrackPath(cur));
                     break;
                  }
            }
```

```
            //Left
            if (validLeft)
            {
               valLeft = maze[row, column - 1];
               if (new int[] { 0, 2 }.Contains(mazeValue) && valLeft != 4)
               {
//The same process is followed for the remaining directions: Left,right,down.
               }
            }
         visited.Add(cur);
         }
      List<Coordinate> smallestList = aStarHospitalPaths.First();

      foreach (List<Coordinate> test in aStarHospitalPaths)
      {
         if (smallestList.Count() > test.Count())
         {
            smallestList = test;
         }
      }
      return smallestList;
   }
```

Code to solve a maze using the Lee algorithm – see Algorithm 3.1 in Section 3.3.1.

```
   //Lee algorithm start
   else if (radioButtonLee.Checked)
   {
      int[,] activeH = ActiveHospitals(); //gets all active hospials
      Coordinate startCoordinate = startTemp;
      startRow = startCoordinate.getRow();
      startColumn = startCoordinate.getColumn();

      for (int h = 0; h < activeH.Length / 2; h++)//runs for each hospital selected
      {
      int goalRow = activeH[h, 0];
      int goalColumn = activeH[h, 1];
      nextToVisit.Clear();
      visited.Clear();
      nextToVisit.Add(startCoordinate);

      while (nextToVisit.First() != null)
      {
         cur = nextToVisit.First();
         row = cur.getRow();
         column = cur.getColumn();
         mazeValue = cur.getMazeValue();
         nextToVisit.RemoveAt(0);
```

```
            validUp = (row - 1 >= 0);
            validLeft = (column - 1 >= 0);
            validRight = (column + 1 <= mazeSize);
            validDown = (row + 1 <= mazeSize);

            if (!ContainsNode(visited, cur))
            {
               //up
               if (validUp)
               {
                  valUp = maze[row - 1, column];
                  if (new int[] { 0, 1, }.Contains(mazeValue) && valUp != 4)
                  {
                     nextToVisit.Add(new Coordinate(row - 1, column, valUp, cur));
                     g.FillRectangle(movement, (column * multiplierWidth) + 1, (row *
multiplierHeight) + 1, multiplierWidth - 2, multiplierHeight - 2);
                     pictureBox1.Refresh();
                  }
                  if (valUp == 5 && ValidHospital(row - 1, column))
                  {
                     return BacktrackPath(cur);
                  }
               }
               //Left
               if (validLeft)
               {

//The same process is followed for the remaining directions: Left, right, down.

               }

               visited.Add(cur);//add current node to list of visited nodes
            }
         }
      }
   }
```

Code to generate an optimal path through backtracking

```
private List<Coordinate> BacktrackPath(Coordinate cur)
{
   List<Coordinate> path = new List<Coordinate>();
   Coordinate iter = cur;

   while (iter != null)
   {
      path.Add(iter);
      iter = iter.getParent();
   }
return path;
}
```

Code to identify each cell in a maze

```
public class Coordinate
{
   int row;
   int column;
   int mazeValue;
   float gValue, fValue;
   Coordinate parent;

   public Coordinate()
   {
      this.row = 0;
      this.column = 0;
      this.gValue = 0;
      this.fValue = 0;
   }

   public Coordinate(int row, int column)
   {
      this.row = row;
      this.column = column;
   }

   public Coordinate(int row, int column, int mazeValue)
   {
      this.row = row;
      this.column = column;
      this.mazeValue = mazeValue;
      this.parent = null;
   }

   public Coordinate(int row, int column, int mazeValue, Coordinate parent)
   {
      this.row = row;
      this.column = column;
      this.mazeValue = mazeValue;
      this.parent = parent;
   }

   public Coordinate(int row, int column, int mazeValue, Coordinate parent, float
gValue, float fValue)
   {
      this.row = row;
      this.column = column;
      this.mazeValue = mazeValue;
      this.parent = parent;
      this.gValue = gValue;
      this.fValue = fValue;
   }

   public void setParent(Coordinate parent)
   {
```

```
        this.parent = parent;
    }

    public void setgValue(float gValue)
    {
        this.gValue = gValue;
    }

    public void setfValue(float fValue)
    {
        this.fValue = fValue;
    }

    public int getRow()
    {
        return row;
    }

    public int getMazeValue()
    {
        return mazeValue;
    }

    public int getColumn()
    {
        return column;
    }

    public Coordinate getParent()
    {
        return parent;
    }

    public float getgValue()
    {
        return gValue;
    }

    public float getfValue()
    {
        return fValue;
    }
}
```

Annexure C: Facility location data

The data set used in the formulation of the facility location models (Chapter 6) consists of seven districts of the New Orleans area. Each of the seven districts is divided into a number of neighbourhoods – 52 neighbourhoods in total. Table C.1 shows the 52 neighbourhoods in each of the seven districts (numbered A to G). The neighbourhoods adjacent to each neighbourhood are also shown, as well as the population of each neighbourhood.

Table C.1 Adjacent neighbourhood assignments and populations used in the facility location model formulations

Neighbourhood		Adjacent neighbourhoods	Population
A1	Lakeshore/ Lake Vista	A2, A3, A4, B1, B2	3699
A2	West End	A1, A3, A5	4090
A3	Lakeview	A1, A2, A4, A5, A6	8154
A4	City Park	A1, A3, A6, B1, B2, C1, C2, C3, C5	2809
A5	Lakewood	A2, A3, A6, C5, E1, E2	1852
A6	Navarre	A3, A4, A5, C5	2778
B1	Lake Terrace & Lake Oaks	B2, B3, B4, B5, A1, A4	2143
B2	Filmore	B1, B3, B6, A1, A4, C1	5520
B3	St. Anthony	B1, B2, B4, B6, B7	4929
B4	Milneburg	B1, B3, B5, B7, B8	4682
B5	Pontchartain Park	B1, B4, B7, B8,	2243
B6	Dillard	B2, B3, B7, C1, D2	4943
B7	Gentilly Terrace	B3, B4, B5, B6, B8, D2, D1	10564
B8	Gentilly Woods	B4, B5, B7, D1	3270
C1	St. Bernard Area	C2, A4, B2, B6, D2	2707
C2	Fairgrounds	C1, C3, C4, C6, A4, D2	5544
C3	Bayou St. John	C2, C4, C5, C6, C8, A4	4298
C4	Seventh Ward	C2, C3, C6, D2, D6, F1	11062
C5	Mid-City	C3, C6, C7, C8, C10, A4, A5, A6, E2	14333
C6	Treme` /Lafitte	C2, C3, C4, C5, C8, C9, D6, F1	4682
C7	Gert Town	C5, C8, C10, E1, E2, E3, E4, E6, G1	4579
C8	Tulane/ Gravier	C3, C5, C6, C7, C9, C10, G1, F2	3692
C9	Iberville	C6, C8, F1, F2	140
C10	B.W. Cooper Apts.	C5, C7, C8, F2, G1	874
D1	Desire Dev & Neighborhood	D2, D3, D4, D7, B7, B8	2713
D2	St. Roch	D1, D3, D5, D6, D7, B6, B7, C1, C2, C4	7305
D3	Florida Area	D1, D2, D4, D5, D7	1406
D4	Florida Dev	D1, D3, D7	1604
D5	St. Claude	D2, D3, D6, D7	6835
D6	Marigny	D2, D5, D7, C4, C6, F1	3085
D7	Bywater	D1, D2, D3, D4, D5, D6	3700

Table C.1 Continued

E1	Hollygrove	A5, E2, E3, C7	6148
E2	Dixon	A5, C5, C7, E1	1647
E3	Leonidas	E1, E4, E5, E7, C7	6988
E4	Marlyville/ Fontainebleau	E3, E5, E6, E8, E9, C7	6172
E5	East Carrollton	E3, E4, E7, E8	3923
E6	Broadmoor	E4, E8, E9, C7, G1, G2	6982
E7	Black Pearl	E3, E5, E8	1777
E8	Audubon	E4, E5, E6, E7, E9, E10, E11	16516
E9	Freret	E4, E6, E8, E10, G2	1844
E10	Uptown	E8, E9, E11, G2, G3, G6	6255
E11	West Riverside	E8, E10, G3, G6, G7	5114
F1	French Quarter	F2, C4, C6, C9, D6	3117
F2	Central Business District	F1, C8, C9, C10, G1, G5	2970
G1	Central City	G2, G3, G4, G5, E6, C7, C8, C10, F2	14065
G2	Milan	G1, G3, G4, E6, E9, E10	5093
G3	Touro	G1, G2, G4, G6, G7, E10, E11	2870
G4	Garden District	G1, G2, G3, G5, G6, G7	2074
G5	Lower Garden District	G1, G4, G7, G8, F2	6092
G6	East Riverside	G3, G4, G7, E11, E10	2756
G7	Irish Channel	G3, G4, G5, G6, G8, E11	3588
G8	St. Thomas Development	G5, G7	2012

Table C.2 on the next page is an incidence matrix that shows the relationship (adjacency) among all 52 neighbourhoods. A value of one indicates that neighbourhoods are adjacent and a zero indicates that they are not adjacent. The incidence matrix was derived from Table C.1 and is a requirement for both the set covering and maximal coverage facility location models formulated and solved in Chapter 6, Sections 6.3.1 and 6.3.2.

Table C.2 New Orleans adjacency incidence matrix

	Neighbourhood	Population	A1	A2	A3	A4	A5	A6	B1	B2	B3	B4	B5	B6	B7	B8	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	D1	D2	D3	D4	D5	D6	D7	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11	F1	F2	G1	G2	G3	G4	G5	G6	G7	G8
A1	Lakeshore/ Lake Vista	3699	0	1	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
A2	West End	4090	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
A3	Lakeview	8154	1	1	0	1	1	1	0	0	0	0	0	0	0	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
A4	City Park	2809	1	0	1	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
A5	Lakewood	1852	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
A6	Navarre	2778	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
B1	Lake Terrace & Lake Oaks	2143	1	0	0	1	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
B2	Filmore	5520	1	0	0	1	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
B3	St. Anthony	4929	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
B4	Milneburg	4682	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
B5	Pontchartain Park	2243	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
B6	Dillard	4943	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
B7	Gentilly Terrace	10564	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
B8	Gentilly Woods	3270	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C1	St. Bernard Area	2707	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C2	Fairgrounds	5544	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C3	Bayou St. John	4298	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C4	Seventh Ward	11062	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C5	Mid-City	14333	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C6	Treme`/Lafitte	4682	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
C7	Gert Town	4579	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C8	Tulane/ Gravier	3692	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C9	Iberville	140	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
C10	B.W. Cooper Apts.	874	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
D1	Desire Dev & Neighborhood	2713	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1	1	1	0	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D2	St. Roch	7305	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1	0	1	0	0	0	0	0	0	1	0	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D3	Florida Area	1406	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D4	Florida Dev	1604	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D5	St. Claude	6835	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D6	Marigny	3085	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
D7	Bywater	3700	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
E1	Hollygrove	6148	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
E2	Dixon	1647	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
E3	Leonidas	6988	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
E4	Marlyville/ Fontainebleau	6172	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
E5	East Carrollton	3923	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
E6	Broadmoor	6982	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
E7	Black Pearl	1777	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
E8	Audubon	16516	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0
E9	Freret	1844	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0
E10	Uptown	6255	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	1	1	0	0	1	0	0	0
E11	West Riverside	5114	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	1	1	0	0
F1	French Quarter	3117	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
F2	Central Business District	2970	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0
G1	Central City	14065	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	1	1	1	0	0	0	0
G2	Milan	5093	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	1	0	0	0	0	0	0
G3	Touro	2870	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	0	1	1	0	1	0
G4	Garden District	2074	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	1	1	0
G5	Lower Garden District	6092	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	1	1	0
G6	East Riverside	2756	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	0	0	1	1
G7	Irish Channel	3588	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	1	0	1	1	0
G8	St. Thomas Development	2012	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0

Table C.3 presents the results of the set covering model formulated and solved in Chapter 6, Section 6.3.1. The table indicates that a minimum of 10 facilities is required to cover the complete New Orleans area. A value of one in the coverage row indicates that a facility should be established in that neighbourhood, e.g. in neighbourhoods A1, B1, etc. Note that a relief facility in a neighbourhood also covers all the adjacent neighbourhoods which means that the total New Orleans population of 248 238 is covered by the ten facilities.

Table C.3 Set covering location problem results

	A1	A2	A3	A4	A5	A6	B1	B2	B3	B4
Population	3699	4090	8154	2809	1852	2778	2143	5520	4929	4682
Coverage	1	0	0	0	0	0	1	0	0	0

B5	B6	B7	B8	C1	C2	C3	C4	C5	C6
2243	4943	10564	3270	2707	5544	4298	11062	14333	4682
0	0	0	0	0	0	0	0	1	0

C7	C8	C9	C10	D1	D2	D3	D4	D5	D6
4579	3692	140	874	2713	7305	1406	1604	6835	3085
1	0	0	0	1	1	0	0	0	0

D7	E1	E2	E3	E4	E5	E6	E7	E8	E9
3700	6148	1647	6988	6172	3923	6982	1777	16516	1844
0	0	0	0	0	0	0	0	1	0

E10	E11	F1	F2	G1	G2	G3	G4	G5	G6
6255	5114	3117	2970	14065	5093	2870	2074	6092	2756
1	0	0	1	0	0	0	0	1	0

G7	G8	Minimum required facilities		
3588	2012		**Population serviced:**	248238
0	0	10	**Population total:**	248238

Table C.4 presents the results of the maximal coverage facility location model formulated and solved in Chapter 6, Section 6.3.2. The table is organised as follows: The first column indicates the number of facilities that will be established; the second column shows the population that will be covered by that number of facilities; and the remainder of the columns indicates in which neighbourhood the facilities should be established (an entry of 1 indicates a facility location). For example, the first row indicates that one facility will be established, This one facility will cover 60 901 people, and the facility must be located in neighbourhood C7. The second row indicates that two facilities will be established. These two facilities will cover 113 460 people, and the facilities need to be established in neighbourhoods C7 and D2.

Table C.4 Maximal covering location problem results

Nodes	Population covered	A1	A2	A3	A4	A5	A6	B1	B2	B3	B4	B5	B6	B7	B8	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	D1	D2	D3	D4	D5	D6	D7	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11	F1	F2	G1	G2	G3	G4	G5	G6	G7	G8
1	60901	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	113460	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	147653	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
4	174245	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
5	196674	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
6	214370	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0
7	231450	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0
8	237217	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0
9	242835	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0
10	248238	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	1	0	0	0

Annexure D: Confirmation of language editing

This serves to confirm that I, Isabella Johanna Swart, registered with and accredited as professional translator by the South African Translators' Institute, registration number 1001128, language edited the following dissertation:

A grid-based maze approach to humanitarian logistics

by

Thean Olivier

Dr Isabel J Swart

Date: 1 December 2020

23 Poinsettia Close
Van der Stel Park
Dormehlsdrift
GEORGE
6529
Tel: (044) 873 0111
Cell: 082 718 4210
e-mail: isaswart@telkomsa.net