**Comparison of the moore’s method and dijkstra’s algroithm in determing the shortest path between two points on a map**

by

Group 18

Cai, Yuhao

Fan, Hsiao-Tien

Liu, Bowen

Paglia, John Naisby

November 4, 2014

SUNY Binghamton University

Binghamton, NY

TABLE OF CONTENTS

Page

LIST OF TABLES iii

LIST OF FIGURES iv

ABSTRACT vi

CHAPTER 1. INTRODUCTION 1

1.1 Introduction 1

1.1.1 Notes 1

1.1.1.1 Page Two 2

CHAPTER 2. CONCLUSION 3

2.1 Before You Submit 3

2.1.1 Sample Pages 3

REFERENCES 8

LIST OF TABLES

Table Page

[Table 2.1 Sample table 4](#_Toc402129410)

ABSTRACT

# INTRODUCTION

## Introduction

For the project of EECE506 – Mathematical Methods in Electrical Engineering for the fall semester of 2014, Group18 has chosen to investigate different methods used in the identification and optimization of a given path between two points on a map. For the scope of this project, two methods of optimization were investigated and implemented in MATLAB to determine their relative efficiencies. In this chapter, the motivation behind the investigation of such techniques is discussed. A brief overview of the different methods invested in this report is also given.

## Motivation

In recent years, Global Positioning System (GPS) navigation has made a transition onto the smartphone, allowing people easy access to clear directions to their desired locations. Such forms of navigation is also seen in web utilities such as Google Maps, which allows the user to determine their most desired path of travel and plan out their trip before they start on their journey. However, as the processing capabilities of the smartphone are still limited, the efficiency of the algorithm in calculating and optimising the route is of great importance. In order to work towards improving such algorithms, we must first understand how to implement these processes of optimisation, and the advantages and the disadvantages of the different methods currently employed.

## Problem description

For this project, it was decided that maps with nxn number of tiles would be developed, and a starting point and ending point would be determined within this map. This map could be represented by a graph, where adjacent tiles within the map would be connected nodes within the graph. By using established mathematical techniques, it is possible to find the shortest path between the starting and ending nodes.

## Methods currently employed for shortest path problems

Common methods used to find the shortest path within a graph include the Moore’s method and Dijkstra’s algorithm, which are the two methods investigated in this project.

### The Moore’s method

The Moore’s method

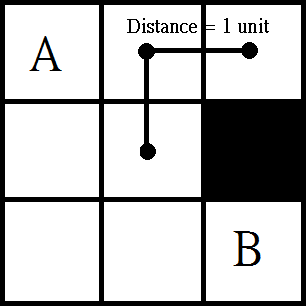
### Dijkstra’s algorithm

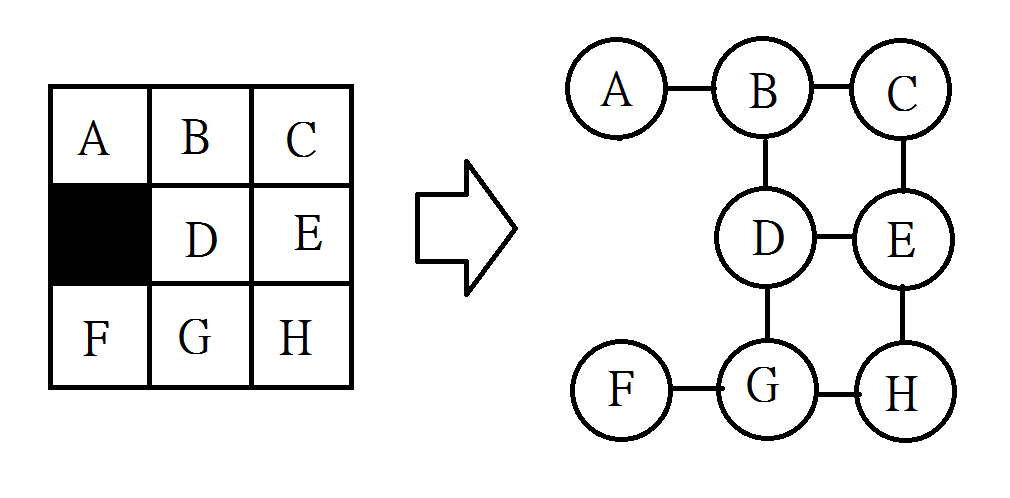
# Problem Formulation

## Literature review

## Mathematical presentation of problem

For the purpose of this project, it was decided that build in MATLAB algorithms would be used to determine the path within a nxn map with the starting and ending tiles being placed on two corners which are diagonal from each other. Obstacles may be present within the graph to limit the number of possible paths. The distance to travel between each tile is 1 unit. This is illustrated in Figure 2.1, where tile A is the starting point and tile B is the ending point.

Figure 2.1 – 3x3 map with starting point A and ending point B placed on opposing corners. The black square represents an obstacle.

Once the map has been determined, it can be transformed into a graph, where mathematical manipulations can be applied to solve for the shortest path between two points. It should be noted that each white tile within this map represent a node within a graph and it is only possible to travel between adjacent tiles. An example of how a map is transformed into a graph is shown in Figure 2.2

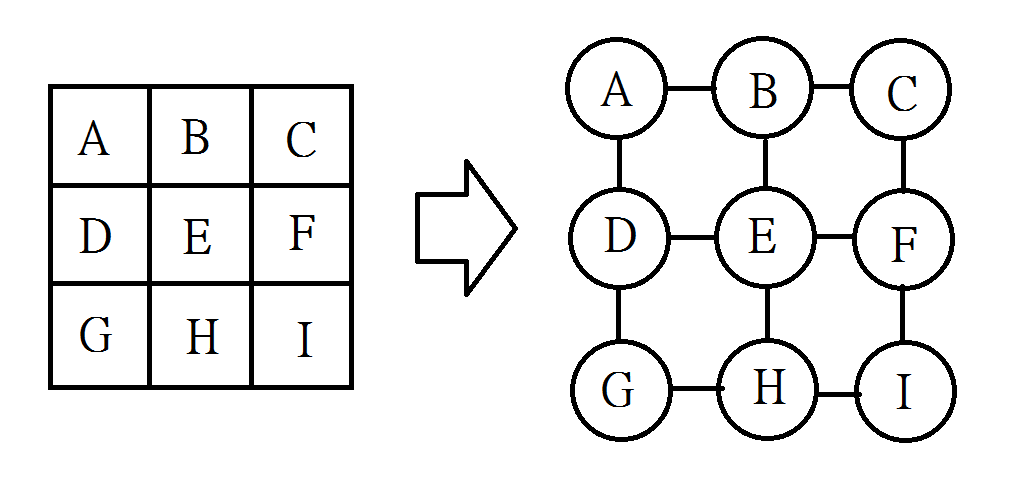
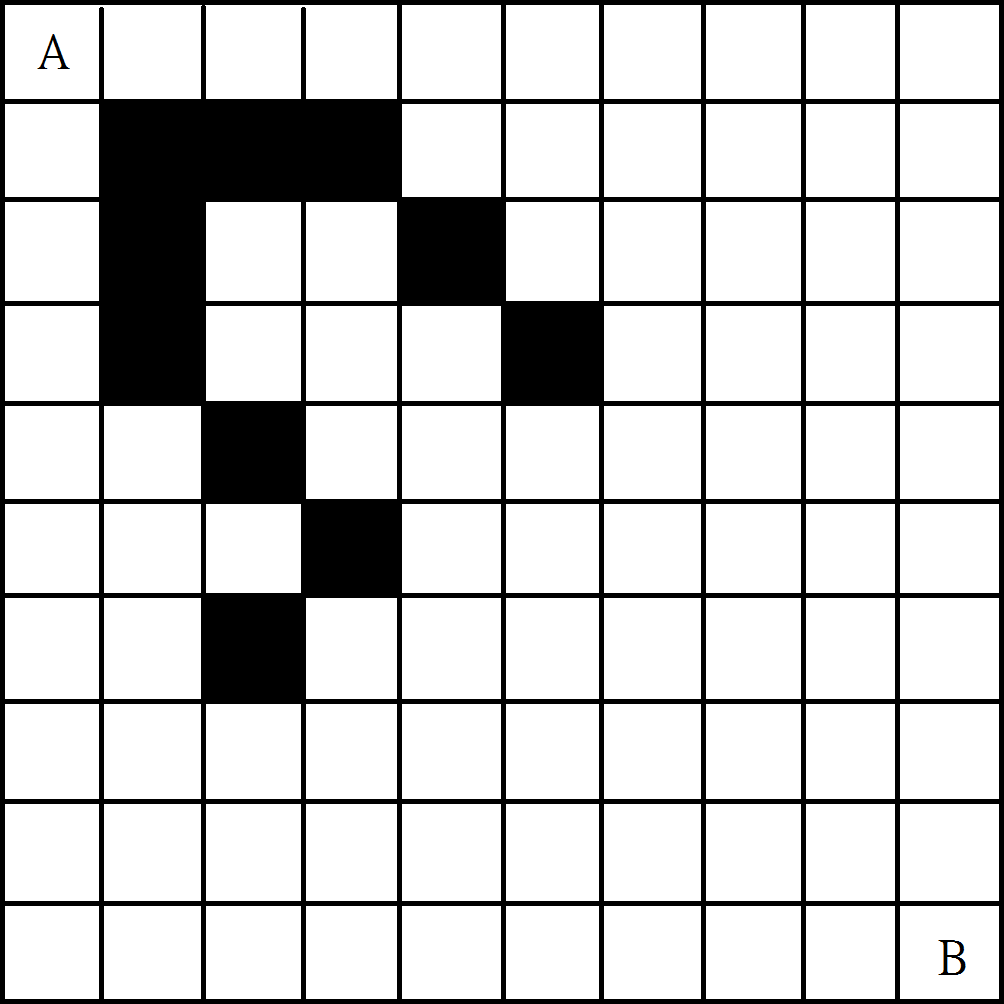
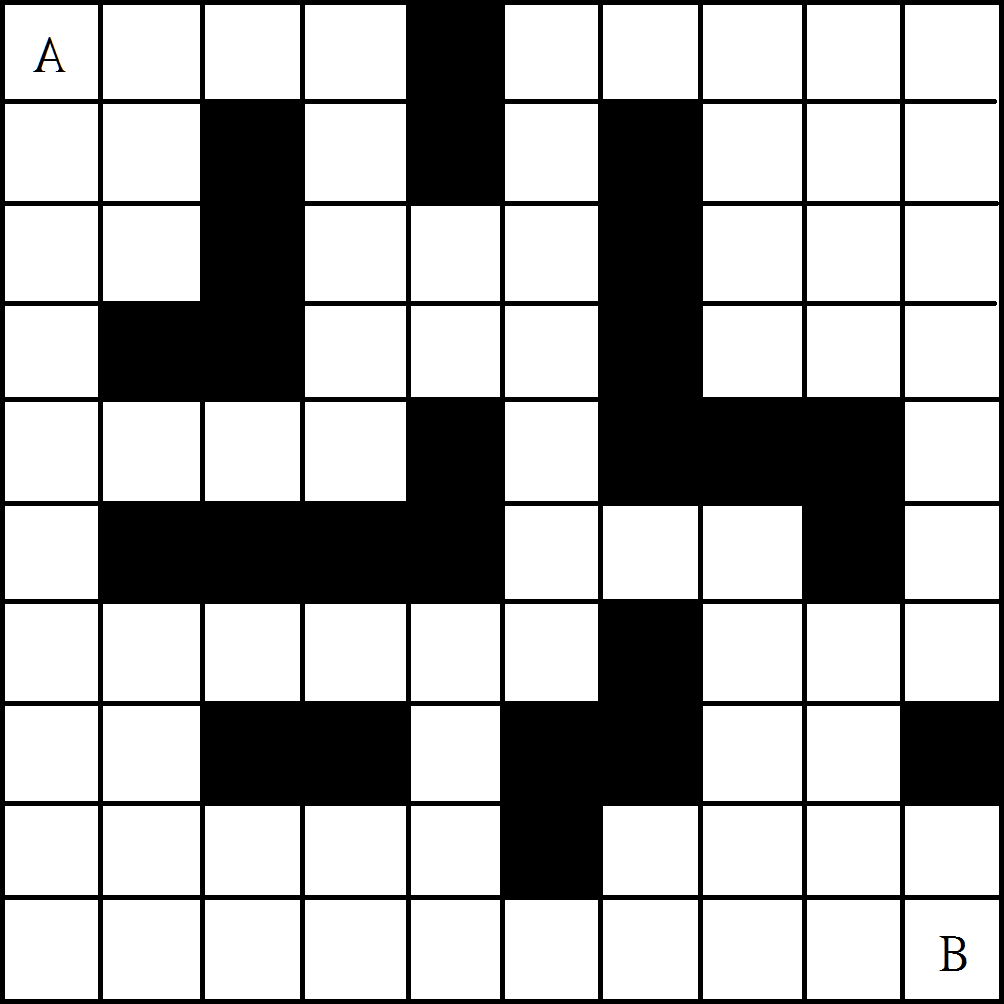
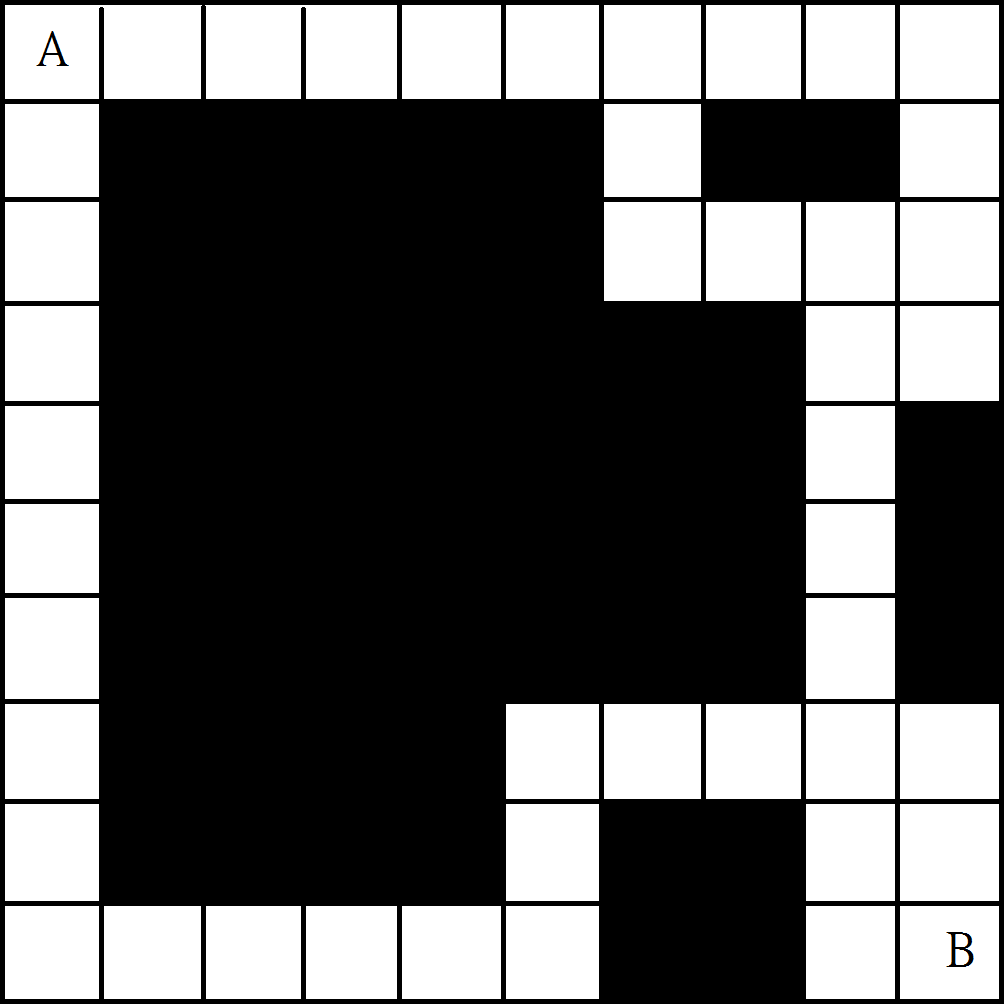
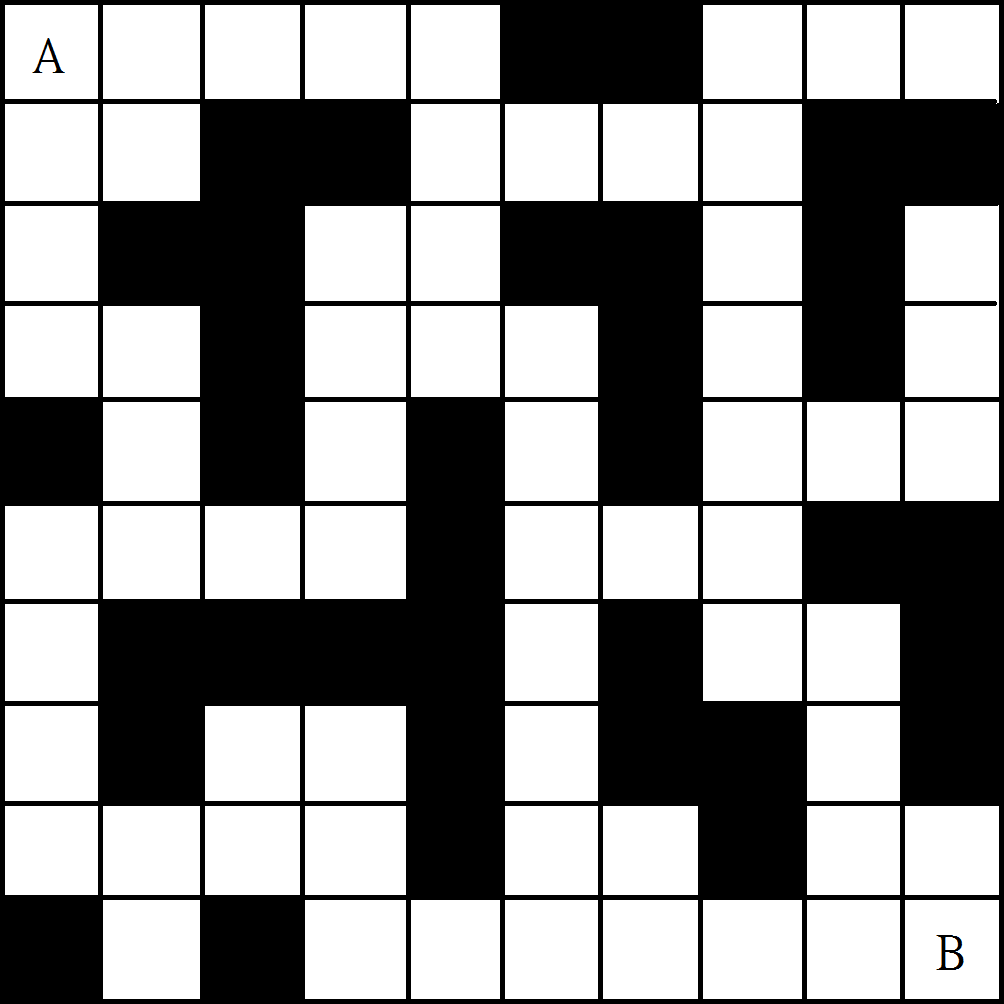
(a) (b)

Figure 2.2

## Maps

Several maps were developed to investigate whether different graph topologies affected the functionality of the algorithms. It was decided that the maps were to be a size of a 10x10 graph, with a certain number of obstacles within this graph. Each obstacle occupies one node within the graph.

It can be seen from Figure 2.2 (a) and Figure 2.2(d) that having two small or too large a number of obstacles could make the task of finding a suitable path redundant. Visually we can see from Figure 2.2(a) that there are many paths that are possible that are equally short. From Figure 2.2(d) we see that if too many obstacles are placed, the problem becomes too simplified, as there are only a few paths that need to be explored. In contrast to this, Figure 2.2(b) and (c) show two maps that have a more reasonable level of complexity, each of which provides many paths to for the algorithm to explore.

1.  (b)

(c) (d)

Figure 2.2 - Maps (a), (b), (c), (d) have been developed with 10, 25, 36 and 55 obstacles respectively.

It was decided that the starting and ending tiles were to be placed on opposing corners. This allows a higher level of complexity to be achieved in a given nxn map, as opposing tiles are furthest apart from each other within a given map.

## Assumptions for this problem

For the purposes of this project, the maps which are analyzed are very simplified compared to real life scenarios. Some assumptions have been made to ensure the consistency and accuracy of the proposed solution. These assumptions have been listed below

* The distance between each node is 1 unit.
* Turing a corner has no impact on the distance travelled.
* The edge between nodes are bidirectional

# Methods

## Comparison of the different methods

In order to compare functionality and efficiency of the Moore’s method and Dikstra’s algorithm, both were implemented using a built in MATLAB function. Each method was used to evaluate predetermined maps and different parameters were measured during this evaluation so each method’s performance can be quantitatively measured. In this chapter, the process of this implementation and the parameters measured is discussed

## Transforming the designed maps into graphs.

In order to solve the shortest path between two points within a map, the map must first be transformed to a graphical form. This is due to the fact that computers do not process information in the same manner as humans. While it is possible for humans to visually assess a map and understand which two adjacent tiles are connected, a computer would not. In order to allow the computer to calculate the shortest path within a map, information describing how the tiles are connected needs to be presented. The way this was implemented is described in this section

### Labeling of nodes

To construct a graph from a map, all the tiles within the map is each labeled with a number. This number is the node label for the specific tile it was assigned to. An example of this is shown in Figure 3.1. As seen in the figure, the top left tile is labeled with the number ‘1’. This particular way of labeling is used to mirror the way MATLAB handles arrays, which allows the algorithm to be more streamline.

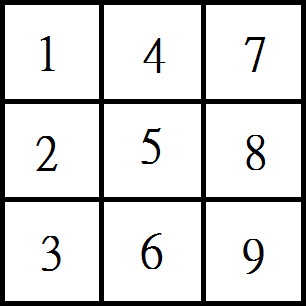


Figure 3.1. Assigning each tile with a node label

### Assess the connections between the tiles

In order for the computer to assess the maps that were designed, it must first be given information about the connections between the tiles. Visually it can be seen from Figure 3.1 that if a connection is made between each adjacent tile, there are 12 connections in total between 9 tiles. However, once the map is large, it becomes impractical to visually asses the connections between the types. In order to process and obtain this information, a MATLAB algorithm was developed to assess any nxn map and output the results to an array. The code for this is included in Appendix A.I.

The information in the array describes which tiles are connected to each other when the map is transformed into a graph. For the function used to assess the shortest path, it requires a n-by-2 array. This function reads the array in the way that the node stored in column 1 is connected to the node stored in column 2. An example of this is shown in Figure 3.2, which is the array generated for the map shown in Figure 3.1

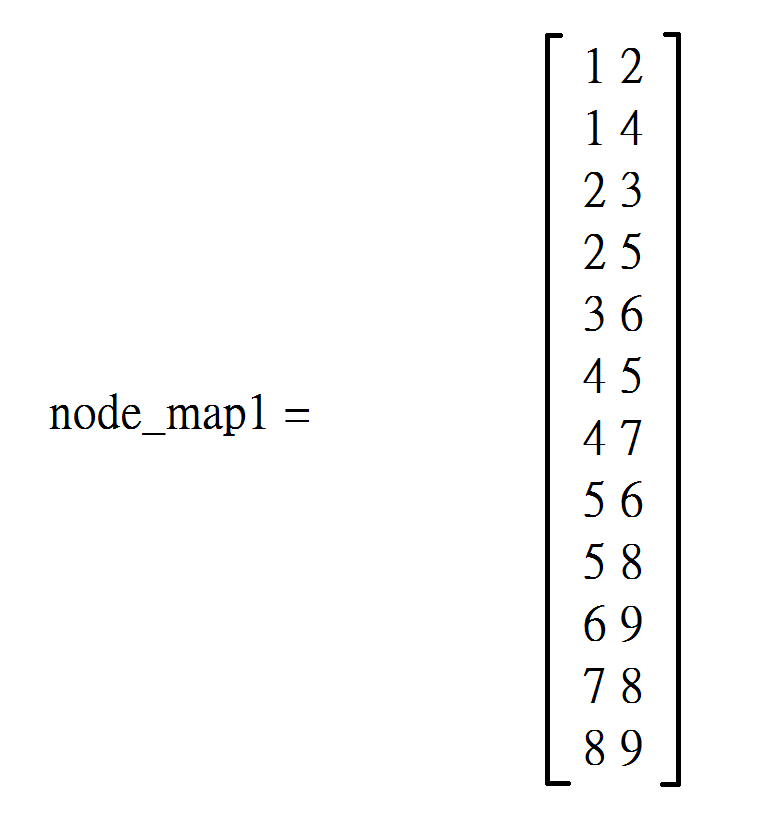
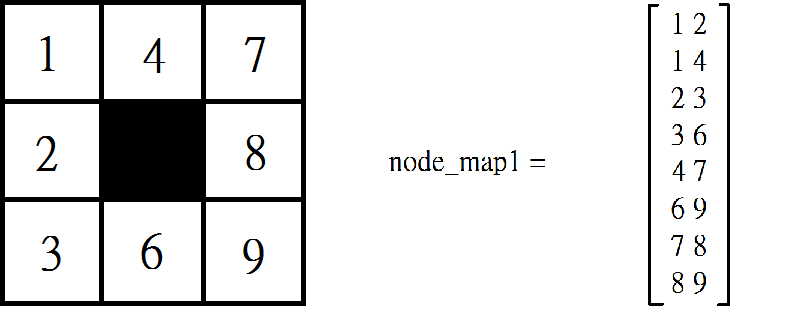


Figure 3.2 Array of graph information describing the nodes which are connected.

Figure 3.2 shows that node 1 is connected to node 2 and node 4. It should be noted that the connection between the nodes is bidirectional, so it does not matter which node is placed in column 1 or 2. For example, the first row in Figure 3.2 is [1 2]. This could have also been recorded as [2 1], and would have no impact on the analysis performed by the proposed MATLAB algorithms.

### Interpreting the obstacles within the map

The next step in converting the maps to graphs is determining which tiles are obstacles. Looking at the map shown in Figure 3.3 there is an obstacle in the center of the map. The corresponding graphical information is also shown. In this case it can be visually assessed that there are 8 connections between 8 tiles. Once again, this assessment was implemented in MATLAB using a custom algorithm so large maps can be assessed. Figure 3.3(b) shows the corresponding array with the graph information when the obstacle is applied.



(a) (b)

Figure 3.3 Map with 1 obstacle and its graphical information in an array

## Moore’s method implementation

Analysis of the graphical information obtained from each proposed map was performed using the Moore’s method within MATLAB with a built in function. Data on the time taken, the optimal path, and how long the optimal path is, was collected. This data is presented and discussed in the results section.

## Dijkstra’s method implementation

Analysis of the graphical information obtained from each proposed map was performed using the Dijkstra’s algorithm within MATLAB with a built in function. Data on the time taken, the optimal path, and how long the optimal path is, was collected. This data is presented and discussed in the results section.

## Data source

The main parameter investigated in this project was the time it took to run the two different algorithms. This data was collected by placing a counter and timer within the MATLAB code. Another factor that was taken into consideration was the effect the complexity of the map had on the performance of the algorithms. This was tested with the maps with different numbers of obstacles. The maps used in this project were individually generated by the project group members.

# Results

## Reference map

For the purposes of verifying the correct implementation of the two methods in MATLAB, the function for implementing both the Moore’s method and the Dijkstra’s algorithm were first applied to the control map shown in Figure 4.1. In this map, there is one obviously longer and one obviously shorter path.

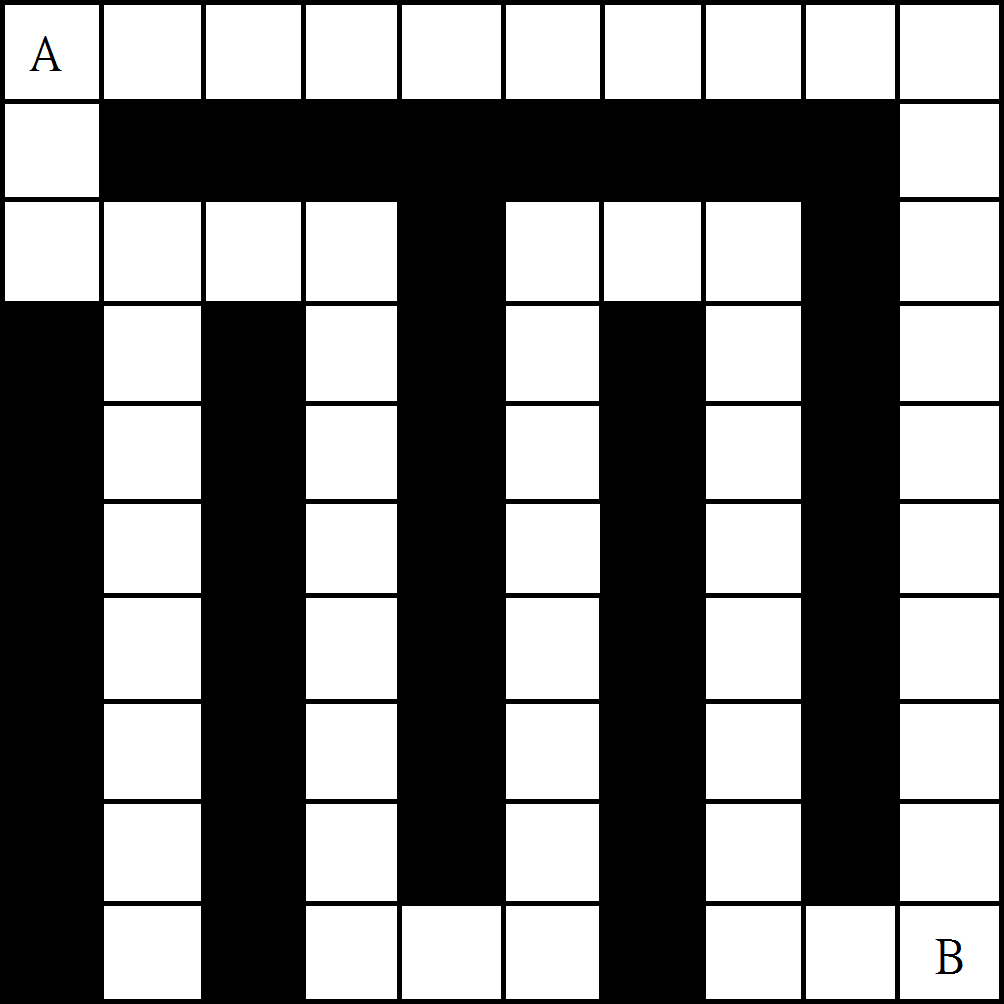


Figure 4.1 Reference map created to verify the correct functioning of the algorithms

Upon the application of the MATLAB functions to the reference map, the same shortest path was found successfully by both methods. The path found is displayed in Figure 4.2

From Table 4.1, it can be seen that while both methods obtained the same shortest path, the time required for the Moore’s Method required less time to implement than the Dijkstra’s Algorithm. Multiple measurements were taken for the time, and the average is presented. The full timing results is presented in Appendix A.II

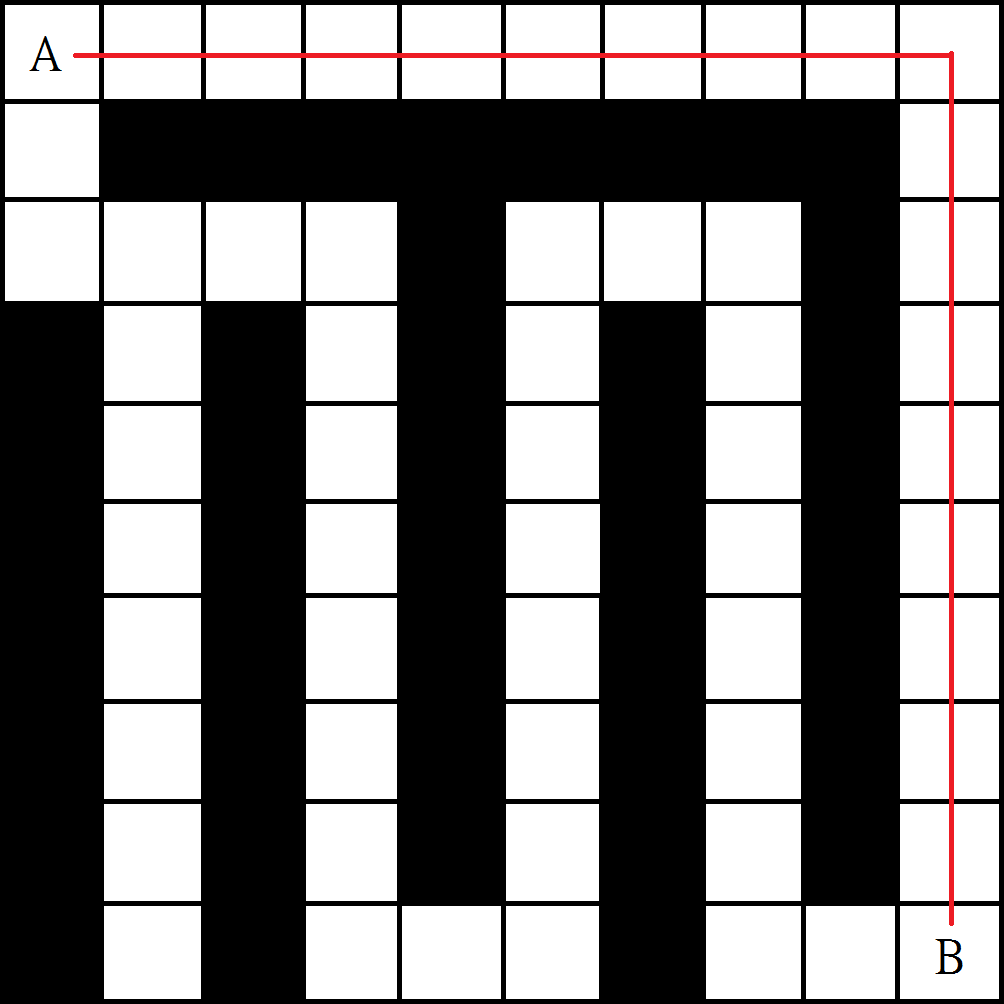


Figure 4.2 Shortest path found by both the Moore’s method and Dijkstra’s algorithm

|  |  |  |
| --- | --- | --- |
|  | Moore’s Method | Dijkstra’s Algorithm |
| Reference Map (average time) | 0.018317s | 0.024174s |
| Shortest path (node – node) | 1-11-21-31-41-51-61-71-81-91-92-93-94-95-96-97-98-99-100 | 1-11-21-31-41-51-61-71-81-91-92-93-94-95-96-97-98-99-100 |
|  |  |  |

Table 4.1 Time taken to find the shortest path with the Moore’s method and the Dijkstra’s algorithm for the reference map

## Maps with different number of obstacles.

For this project, a total of 4 maps with different obstacles were assessed. In this section, the result obtained from the analysis of these maps will be presented. The results presented in this section will be discussed in the discussion chapter.

### 10x10 Map with 10 obstacles

The topology of the map with 10 obstacles is displayed in Figure 4.3 and the timing and shortest path results are displayed in Table 4.2. Multiple measurements were taken for the time, and the average is presented. The full timing results is presented in Appendix A.II

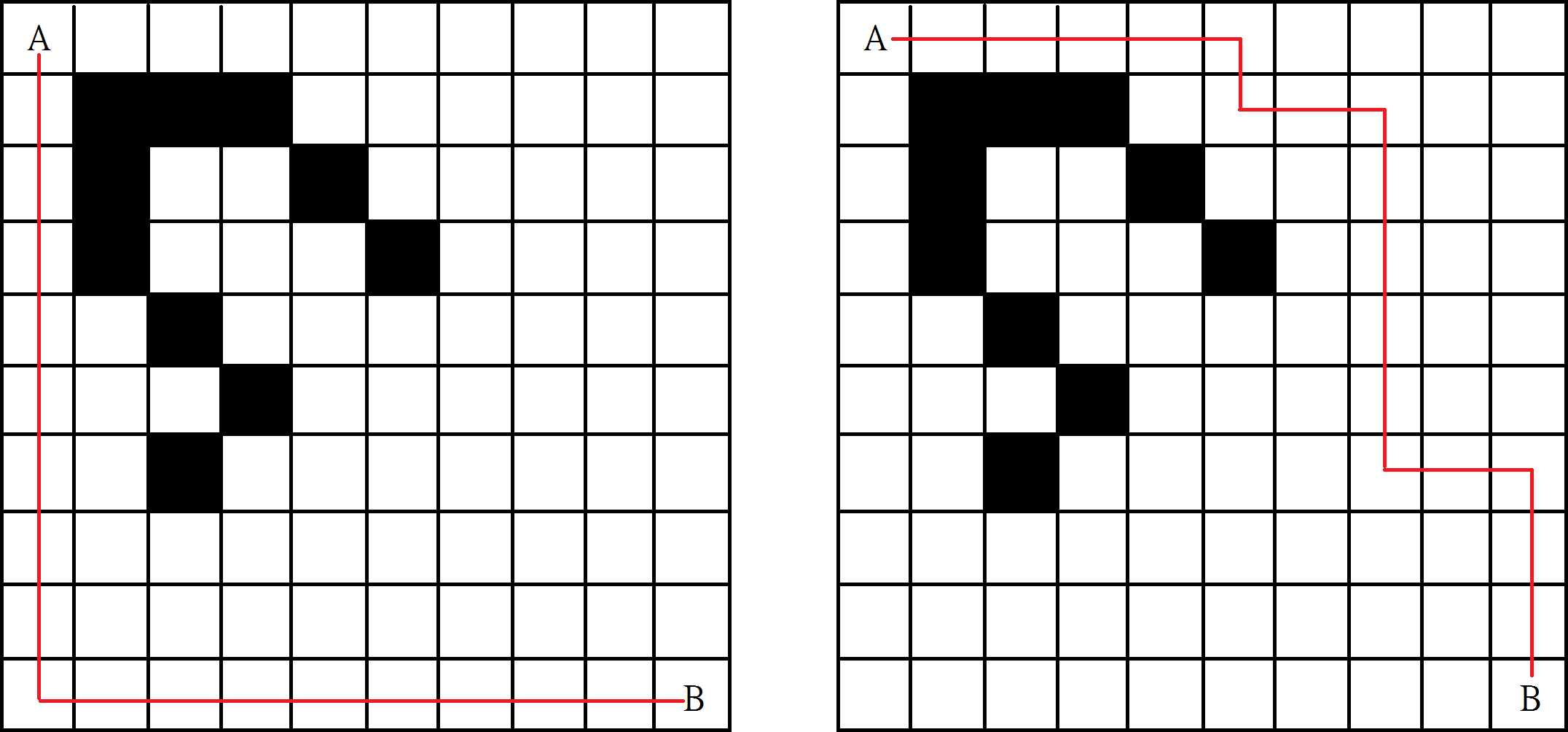


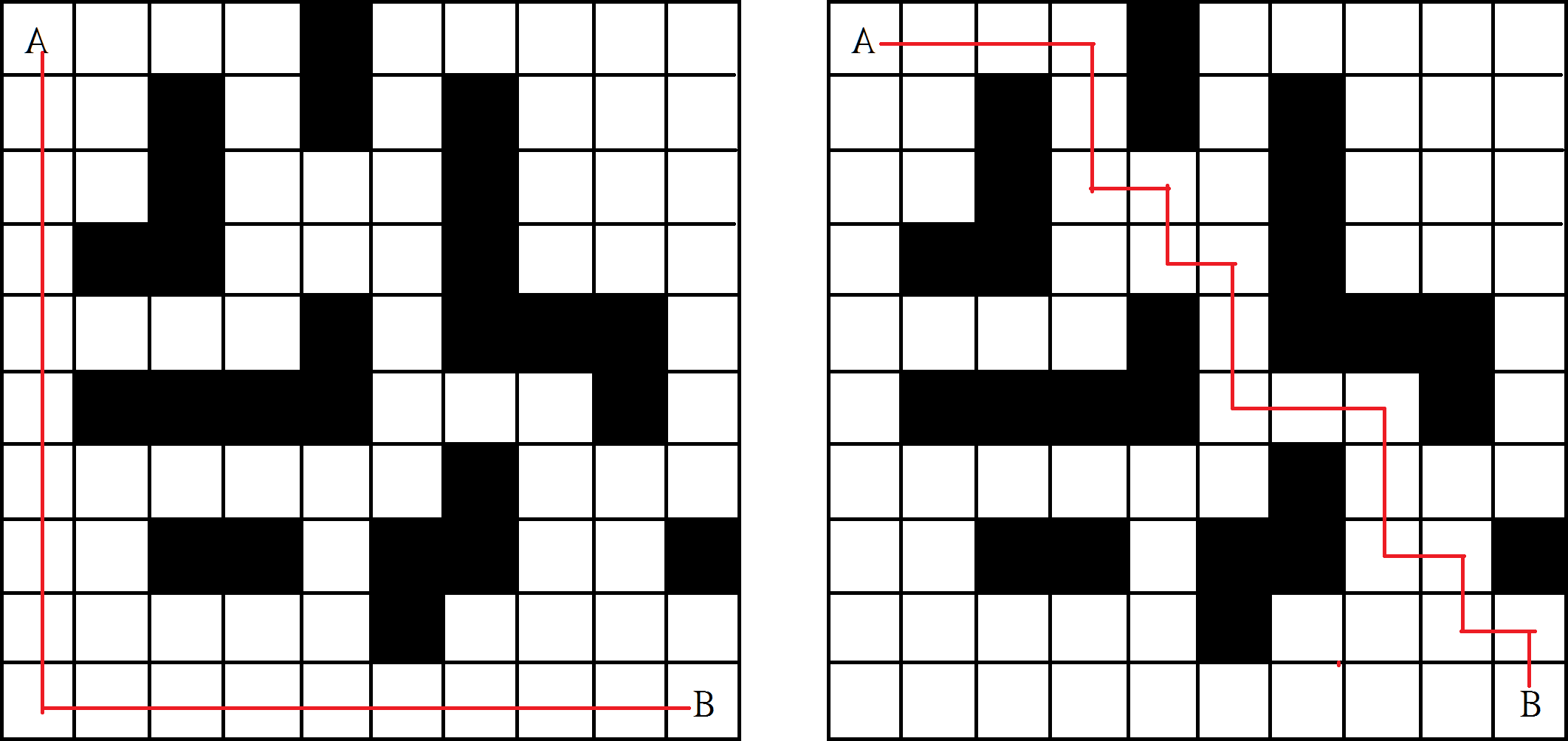
Figure 4.3 10x10 map with 10 obstacles and the shrotest

|  |  |  |
| --- | --- | --- |
|  | Moore’s Method | Dijkstra’s Algorithm |
| Map10 (average time) | 0.005373 s | 0.005400s |
| Shortest path (node – node) | 1- 2- 3- 4- 5- 6- 7- 8- 9-10-20-30-40-50-80-90-100(18 steps) | 1-11-21-31-41-51-52- 62-72-73-74-75-76-77-87-97- 98-99-100 (18 steps) |

Table 4.2 Time taken to find the shortest path with the Moore’s method and the Dijkstra’s algorithm for map (10 obstacles)

### 10x10 Map with 25 obstacles

The topologies of the map with 25 obstacle is displayed in Figure 4.4 and the timing and shortest path results are displayed in Table 4.3. Multiple measurements were taken for the time, and the average is presented. The full timing results is presented in Appendix A.II

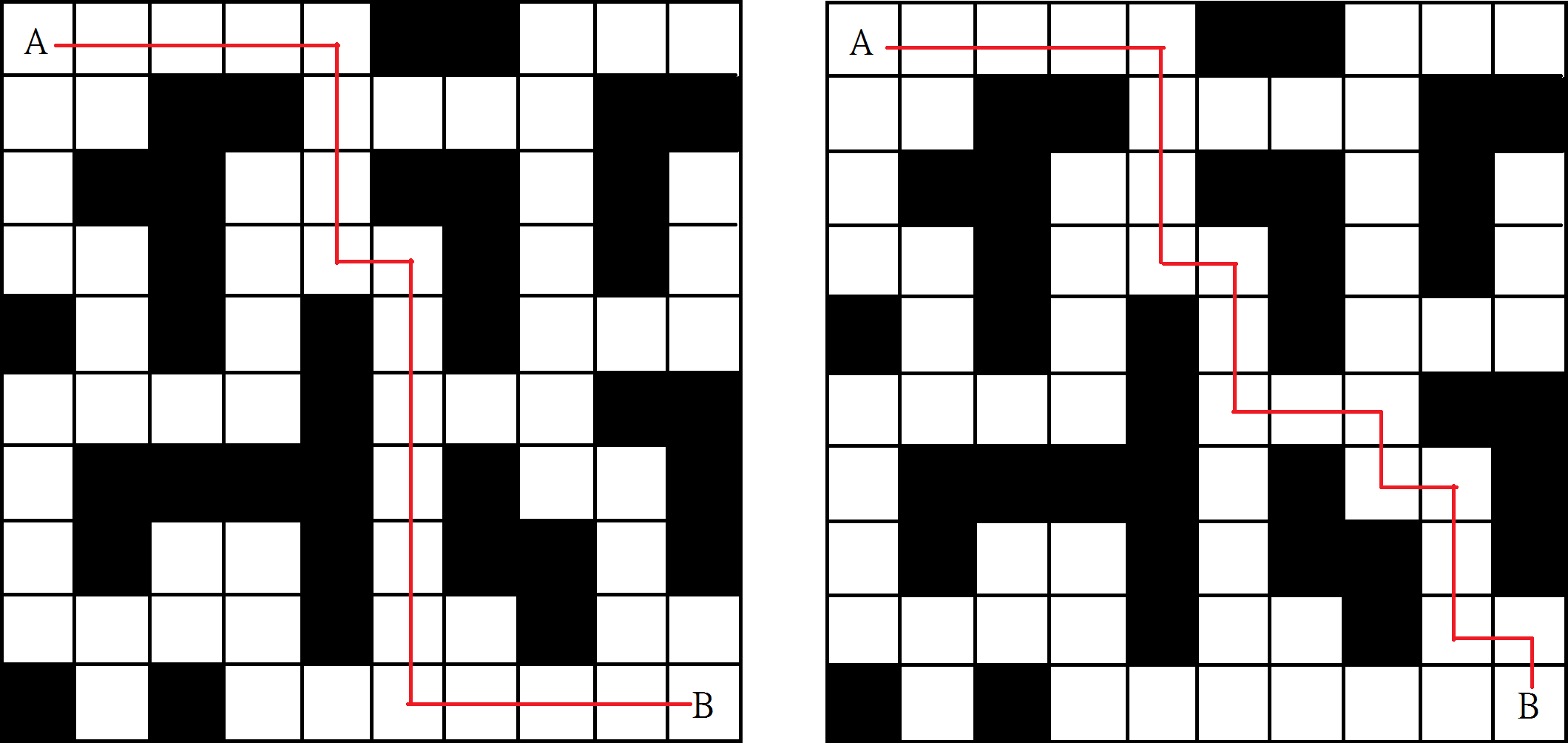


|  |  |  |
| --- | --- | --- |
|  | Moore’s Method | Dijkstra’s Algorithm |
| Map25 (average time) | 0.006422 | 0.005750s |
| Shortest path (node – node) | 1- 2- 3- 4- 5- 6- 7- 8- 9-10-20-30-40-50-80-90-100(18 steps) | 1-11-21-31-32-33-43-44-54-55-56-66-76-77-78-88  -89-99-100 (18 steps) |

Table 4.3 Time taken to find the shortest path with the Moore’s method and the Dijkstra’s algorithm for map (25 obstacles)

### 10x10 Map with 36 obstacles

The topologies of the map with 36 obstacle is displayed in Figure 4.5 and the timing and shortest path results are displayed in Table 4.4. Multiple measurements were taken for the time, and the average is presented. The full timing results is presented in Appendix A.II

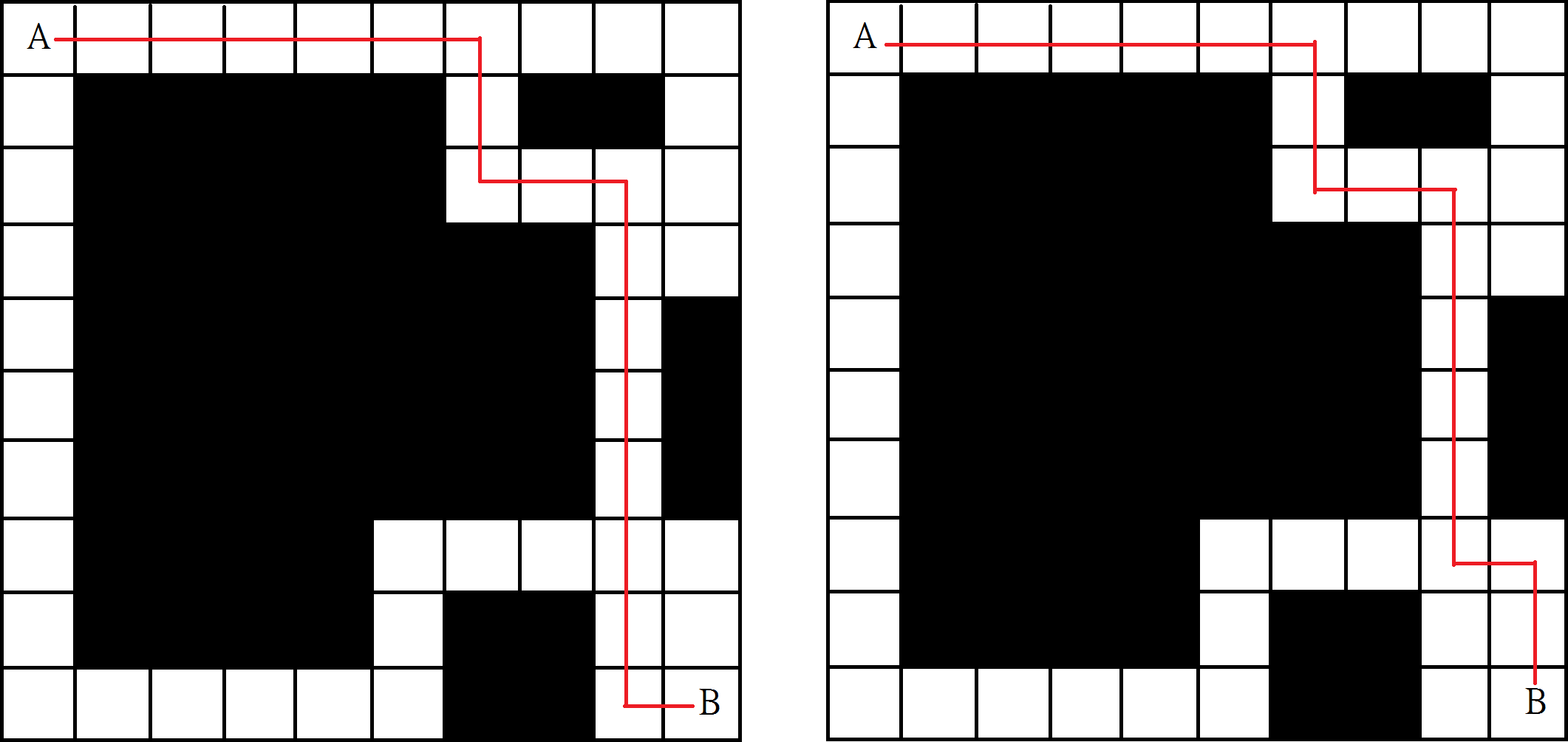


|  |  |  |
| --- | --- | --- |
|  | Moore’s Method | Dijkstra’s Algorithm |
| Map36 (average time) | 0.005192s | 0.005374s |
| Shortest path (node – node) | 1-11-21-31-41-42-43-44-54-55-56-57-58-59-60-70  -80-90-100(18 steps) | 1-11-21-31-41-42-43-44-54-55-56-66-76-77-87-88  -89-99-100(18 steps) |

Table 4.4 Time taken to find the shortest path with the Moore’s method and the Dijkstra’s algorithm for map (36 obstacles)

### 10x10 Map with 55 obstacles

The topologies of the map with 55 obstacle is displayed in Figure 4.6 and the timing and shortest path results are displayed in Table 4.5 Multiple measurements were taken for the time, and the average is presented. The full timing results is presented in Appendix A.II

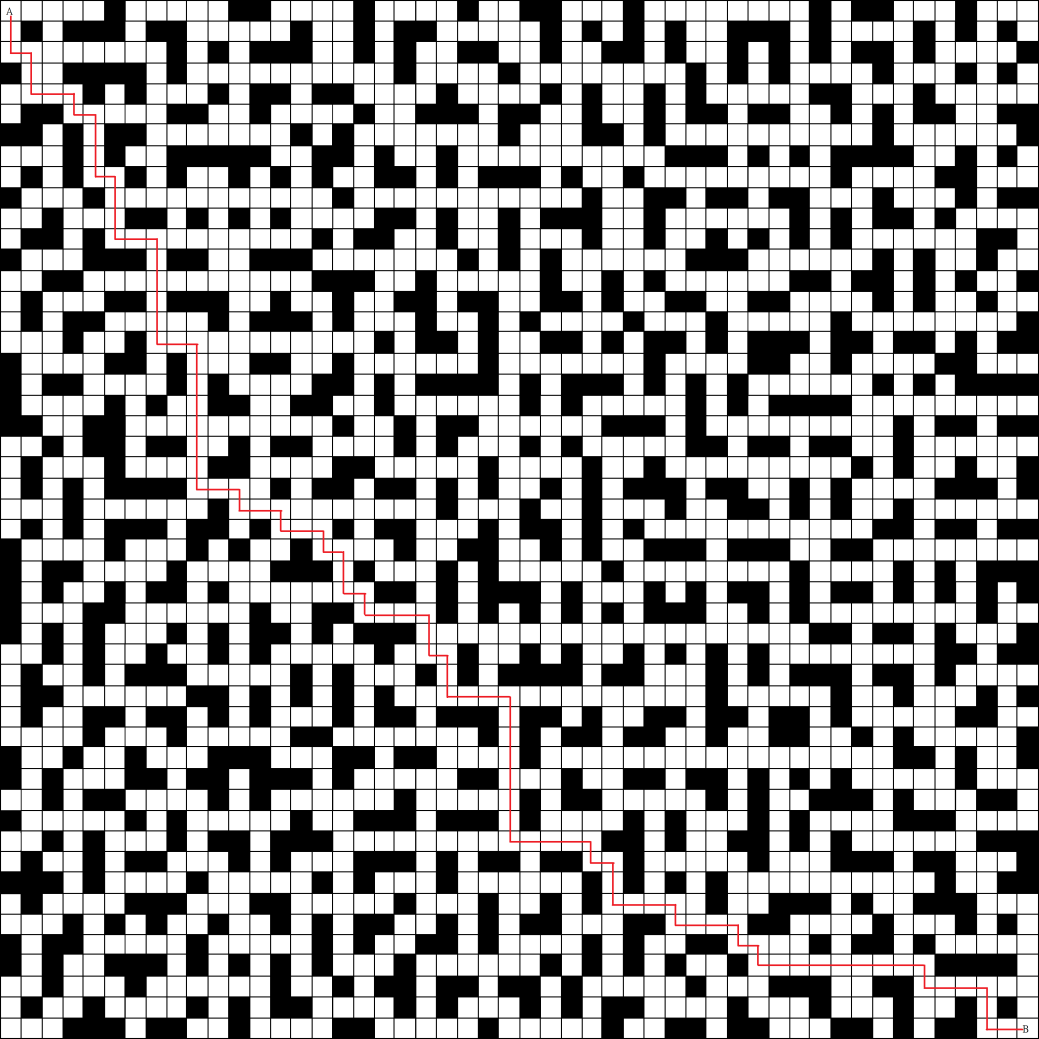


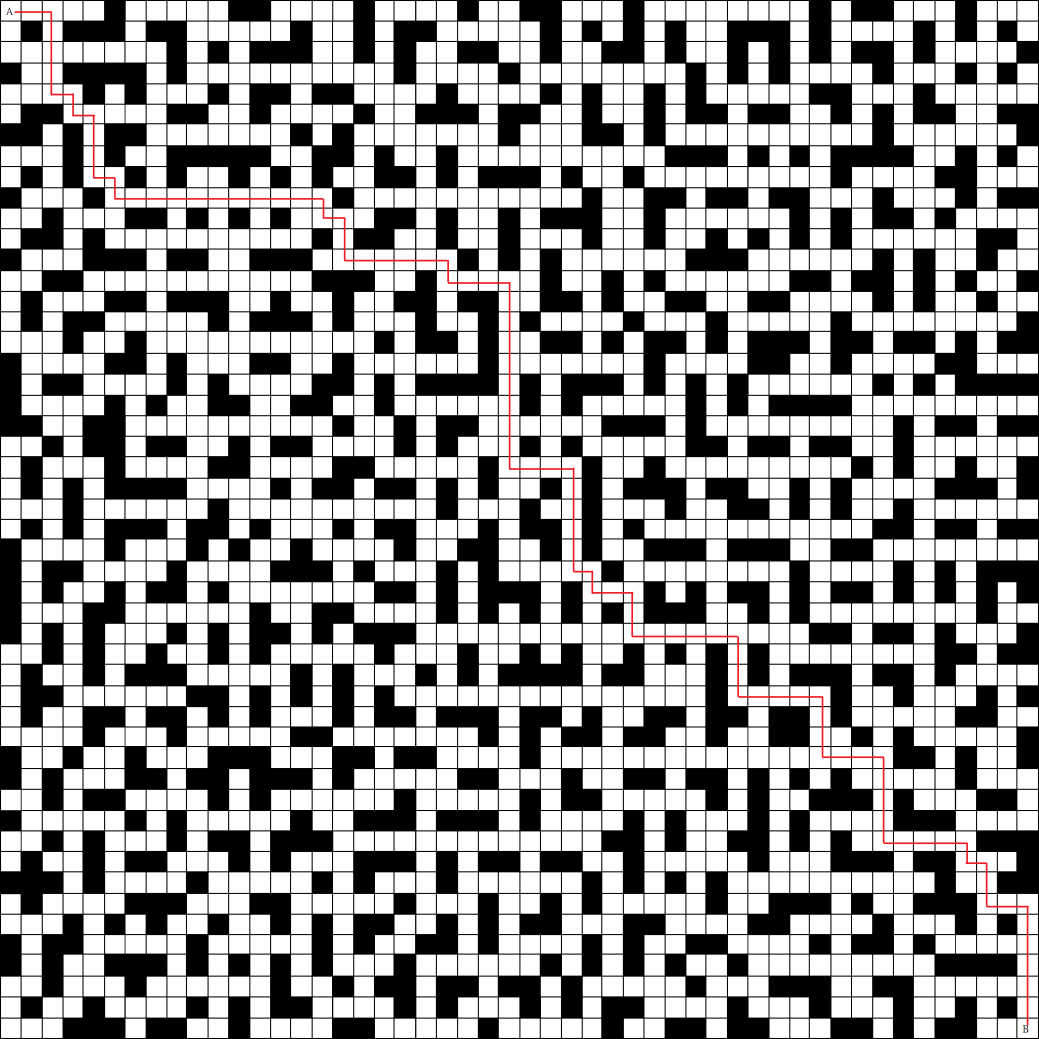
|  |  |  |
| --- | --- | --- |
|  | Moore’s Method | Dijkstra’s Algorithm |
| Map55 (average time) | 0.005399s | 0.005464s |
| Shortest path (node – node) | 1-11-21-31-41-51-61-62-63-73-83-84-85-86-87-88  -89-90-100(18 steps) | 1-11-21-31-41-51-61-62-63-73-83-84-85-86-87-88  -98-99-100(18 steps) |

Table 4.5 Time taken to find the shortest path with the Moore’s method and the Dijkstra’s algorithm for map (55 obstacles)

## 50x50 Map with 900 obstacles

A 50x50 tile map was developed to demonstrate the algorithm’s ability to deal with large systems. The 50x50 tile map had an equivalent graph of 1600 nodes and 900 obstacles. The timing and shortest path results are displayed in Table 4.6. Multiple measurements were taken for the time, and the average is presented. The full timing results is presented in Appendix A.II





|  |  |  |
| --- | --- | --- |
|  | Moore’s Method | Dijkstra’s Algorithm |
| Map55 (average time) | 0.005399s | 0.005464s |
| Shortest path (node – node) | 1-2-3-53-54-55-105-155-156-206-207-208-209-259-260-261-262-312-362-363-364-365-366-367-417-467-468-469-470-471-472-473-474-524-574-575-625-675-676-726-776-777-827-828-829-879-880-930-980-1030 1031-1032-1082-1083-1084-1134-1184-1234-1235-1236-1237-1238-1239-1240-1241-1291-1341-1391-1441-1442-1492-1493-1494-1544-1594-1644-1645-1695-1745-1795-1796-1846-1847-1897-1947-1997-2047-2097-2147-2197-2247-2248-2298-2348-2398-2399-2400-2450-2500 | 1-51-101-102-103-104-105-155-156-206-207-208-209-259-260-310-360-410-460-510-560-610-660-710-760-761-811-812-813-863-913-963-1013-1063-1064-1114-1164-1214-1215-1216-1217-1218-1219-1220-1221-1222-1223-1273-1323-1373-1374-1375-1376-1377-1378-1428-1429-1479-1529-1530-1531-1581-1631-1681-1731-1781-1782-1783-1784-1834-1884-1934-1984-1985-1986-1987-2037-2087-2137-2138-2139-2140-2141-2191-2241-2291-2341-2342-2392-2393-2394-2444-2494-2495-2496-2497-2498-2499-2500 |

# Discussion

## The 10x10 map

## The original investigation of the project involved using maps which were 10x10 tiles. Upon obtaining the results, it was observed that the times required to solve the graphs deduced from these maps are very small. When a measurement was taken for a map multiple times, the fluctuation between the individual readings was quite large, so a clear diction between the different map complexities and the two methods was not able to be made. An attempt to overcome this was made by introducing maps with 50x50 tiles.

## The 50x50 map

## The 50x50 map was developed to both illustrate the algorithm’s ability to handle large graphs and to make the time required for the two methods to be noticeable different. While the

## Complexity of the map

Looking at the results, the first thing that can be observed is quite counter intuitive.

## The analysis and discussion of the results described in chapter 3 will be presented in this section.

# Future work

* Take into account of corners
* Consider optimization by lumping straight line bits and use a weighted dijkstra

# Work Distribution

## Yu Hao Cai

* Investigation of the Moore’s method and Dijkstra algorithm (Introduction section)
* Implementation of Moore’s method in MATLAB.

## Hsiao-Tien Fan

* Construction of 50x50 map
* MATLAB implementation of solution (map to graph, implementation of Moore and Dijkstra’s algorithms)
* Report write up (Introduction, Methods, Results, Discussion)
* Analysis of results
* Results discussion

## Bowen Liu

* Construction of maps
* Investigation of effect of map complexity on possible path calculation

## John Naisby Paglia

* Investigation of the Moore’s method and Dijkstra’s algorithm (Literature review section)
* Construction of maps

# References

# Appendix A