AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

Faculty of Metals Engineering and Industrial Computer Science

Master of Science Thesis

A comparative analysis of the use of maze solving algorithms on the example of a dedicated web application

Author: *Sonia Orlikowska*
Degree programme: *Industrial Computer Science*
Supervisor: *Magdalena Kopernik, PhD, DSc, Assoc. Prof.*

Kraków, 2022

Dedicated to the memory of my late brother, Filip, whose absence created the most difficult maze of my life. Let this work be the proof that I found a way out of it.

Abstract

In this Master's Thesis, three maze generators and three maze solvers have been investigated and described. Chapter 2 describes in detail the theoretical foundations derived from Graph Theory, necessary for a precise description of the essence of the studied problems. It shows the most important parameters describing graphs and mazes, and includes all definitions. All used algorithms are described in detail, along with the listings.

Maze problems and related algorithms are very popular in real-life applications. Chapter 3 presents in detail where discussed algorithms are used, and how people benefit from them.

One of the main purposes of this work was also to provide an accurate framework for defining the maze's complexity. The most common idea behind measuring maze complexity derives only from one concept work. Chapter 4 proves that there are more ways to define maze complexity and that measures of complexity derived directly from Graph Theory can be used with great success.

Chapter 5 presents the collected results and their analysis. This chapter summarizes all obtained work objectives that are consistent with the assumed objectives. The analysis of the results shows that it is possible to create a framework that allows distinguishing between different mazes based on their parameters. It is also feasible to evaluate maze complexity by more than the McClendon's measure. Also, Shannon's entropy may be used as a reliable complexity measure for the studied maze generators. Finally, the conclusions of the results allow providing an answer that the Dijkstra solver is the best solver for evaluated algorithms, not only for perfect mazes but also for more complicated ones.

Chapter 6 presents a detailed description of a web application developed for this work. A lightweight application created in JavaScript allows visualising the results of the maze generators and solvers described in this work. In the application, it is possible to verify how the described parameters affect the appearance of the maze, their solutions and the required solution time on dedicated charts, tables and figures.

The results and conclusions presented in this work can be used by others to expand research in this area. The framework presented here can be applied to other types of maze generators and solvers to validate conclusions for other maze problems.

Key words: Maze, Aldous-Broder, Binary Tree, Recursive- Backtracker, BFS, Dijkstra, Astar, McClendon Complexity, Shannon's entropy complexity measure

Contents

List of Symbols

G	graph
V	set of vertices
v	vertex
n	vertex neighbour
E	set of edges
e	edge
$A(i, j)$	adjacency matrix
B	binary tree
T	spanning tree
$d(v)$	vertex degree
$P(k)$	degree distribution
n_k	k-degree vertex
C_i	clustering coefficient
$H(M)$	Shannon entropy
r	maze density
p_l	path length
M	maze
h	hallway
W_h	subset of all v in h
S	maze solution
p	maze starting vertex
A^*	Astar Algorithm
BFS	Breadth First Algorithm
$HTML$	HyperText Markup Language
CSS	Cascading Style Sheets
DOM	Document Object Model
CI/CD	Continuous Integration/Continuous Delivery
SD	Standard Deviation

1. Introduction

Maze has a long history spanning thousands of years. It intrigued ancient philosophers, artists, and scientists. In the modern days, we can easily say that mazes are everywhere. From children's puzzles, traced by finger, Pac-man game and psychology experiments on mice in a laboratory to the movie Labyrinth from 1986. But the omnipresence of mazes is even greater. Mazes also intrigued scientists who are still studying them carefully. It was soon noticed that it may also present the maze construction as a graph. Every problem which may be presented as a graph might be considered in some ways as a maze. It opens an enormous variety of real-life applications of maze theory such as navigation systems, transportation route planning systems, building complexity in video games, solving networking and electrical problems and describing complex systems in physics and chemistry. To its popularity, it can be stated with ease that studying the maze generating and solving algorithms, searching for difficulty measures, and searching for a new better solution for many real-life applications is important, both for specialists and society.

1.1. Motivation

This thesis analyses algorithms which are widely used in different areas of life and technology. Modern, rapidly changing world and the drive for new technologies pose a challenge to the commonly used solutions. The main goal of this work is to assess the possibility of creating a framework that allows to distinguish between different types of mazes and to decide which solving algorithm will be the best for a specific solution. The aim is to understand which parameters have the biggest impact on the solution time. Evaluated parameters in this work are vertices degree ratio, McClendon's complexity measure, average path length and Shannon's entropy. Three generating algorithms were tested Binary Tree Algorithm, Aldous-Broder Algorithm and Recursive-Backtracker Algorithm. They were tested in two variants, one generic one with only one solution, and the second one with added cycles and directions. There were also three solving algorithms tested Dijkstra's Algorithm, BFS Algorithm and A^* Algorithm with Manhattan Distance as a heuristic function. Those algorithms were chosen because of their wide popularity in different applications but also because of their low algorithmic complexity which allowed to generate a lot of data fast which was also important due to a lot of testing required.

During the literature research, it was challenging to find works which are focused on formal analysis of maze complexity and maze classification other than fixed on McClendon's measure. Most of the research is centred on analyzing perfect mazes complexity based on McClendon's and one other maze

parameters using only one solving algorithm [**2**, **4**]. On the other hand, there is also a lot of research on maze-solving algorithms comparison and evaluating their performance, but without reference to the analysis of the characteristic of the problem[**3**]. There is a study made by A. Karlsson [**1**] who focuses on a comparative analysis of three maze-generating algorithms but in his work, he only uses one maze-solving algorithm. Moreover, none of those studies applies cycles or direction to mazes. Therefore the fundamental idea of this work was to combine two main approaches in maze study into one. To compare and analyze different maze-generating algorithms and their solution time in reference to different solving algorithms. This approach offers a new input of contribution to this field of research.

1.2. Research Problem

Taking into account the findings of the studies described in the motivation part, this study's objective is to address the following research questions:

Q1. What is the relation between the maze features, generated by Binary Tree, Aldous-Broder and Recursive-Backtracker algorithms, and their completion time obtained, when solved using a BFS, Dijkstra and A^* algorithms?

Q2. Which maze parameters best describe the complexity of a problem in terms of time completion?

Q3. Which maze features are the best to distinguish different types of mazes?

1.3. Thesis Layout

This thesis is divided into seven chapters: Introduction, Theoretical Background, Maze Real Life Related Problems, Maze Complexity Problems, Results, Analysis and Discussion, Web Application Implementation and Conclusions. Chapter 2 provides a necessary overview of maze algorithms and graph theory used in this work. Chapter 3 summarize the real-life application of algorithms assessed in this work. Chapter 4 describes in detail the concepts and methods of building a complexity measure for mazes. Chapter 5 presents the results and a detailed comparative analysis of implemented algorithms. Chapter 6 presents the results of a created dedicated web application.

2. Background

In this chapter, following subjects will be covered: theoretical background of maze-generating algorithms, maze-solving algorithms, and other theoretical concepts from the graph theory required to better understand the problems included in this work. Moreover the concept of determining the difficulty of a maze will be introduced.

2.1. Graph Theory

In the following section, the graph theory most important mathematical concepts, and a naming convention to follow in this paper will be established.

Definition 1. ***A Set*** is an object of distinct elements where no element is a set itself[**5**].

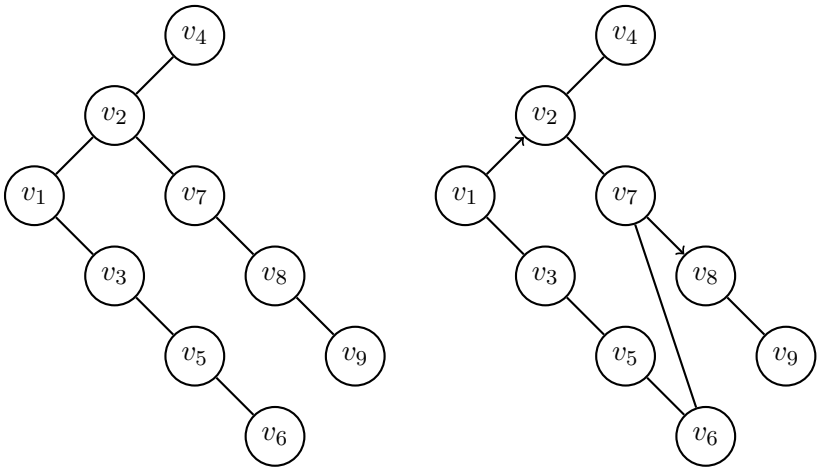
Definition 2. ***A Graph*** is an object comprising two sets called vertex set V and edge set E. V is a finite, nonempty and the E may be empty. A graph usually denoted as $G = (V, E)$[**5**].

In this work, two interesting subgroups of graphs will be discussed: directed graph, and cyclic graph. By applying *direction* to edges of a *undirected*, *acyclic* graph presented on Figure 2.1(a), we are receiving a *directed graph* presented on Figure 2.1(b). A cyclic graph consists of at least one single *cycle*, which means at least 3 vertices connect in a closed chain Figure 2.1(b).

Definition 3. ***An Adjacency Matrix*** of a graph $G = (V, E)$ is a representation in which we number the vertices in some arbitrary way e.g. $1, 2, 3, \ldots, |V|$. The representation of a Matrix of consisting $|V|x|V|$ such that:

$$A(i,j) = \begin{cases} 1, if(i,j) \in E, \\ 0, otherwise \end{cases}$$

Figures 2.2(a) and 2.2(b) are the adjacency matrices of graphs presented on Figure 2.1(a) and 2.1(b) respectivly. The adjacency matrix of a graph requires $O(|V|^2)$ memory, independent of the number of edges in the graph.



(a) (b)

Figure 2.1. In this picture different types of graphs are presented. Graph (a) is an undirected, acyclic graph, where *v* denotes a vertex. Graph (b) has two directed edges marked by an arrow, and contains one cycle $x_1 \to x_2 \to x_7 \to x_6 \to x_5 \to x_3$.
Source: developed by the author.

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

(a) (b)

Figure 2.2. Examples of different adjacency matrices: adjacency matrix (a) is a matrix of graph presented in Figure 2.1(a), adjacency matrix (b) is a matrix of graph in Figure 2.1(b).
Source: developed by the author.

Definition 4. ***Density r*** of a graph defines how complete the graph is. We define density as the number of edges divided by the number called possible. The number of possible is the maximum number of edges that the graph can contain. If self-loops are excluded, then the number possible is:

$$\frac{n(n-1)}{2} \tag{2.1}$$

Where:
n is the number of vertices in a graph.

If self-loops are allowed, then the number possible is:

$$\frac{n(n+1)}{2} \tag{2.2}$$

Definition 5. ***A Free Tree*** is an undirected, acyclic, connected graph. Let $G = (V, E)$ be an undirected graph. Properties of a tree [**6**]:
− *G is a free tree,*
− *every two vertices in G are connected by a unique path,*
− *G is connected, but if any edge is removed from E,the graph becomes disconnected,*
− *G is connected, and* $|E| = |V| - 1$*,*
− *G is acyclic, and* $|E| = |V| - 1$
− *G is acyclic, but if we add any edge to E, the graph contains a cycle.*

Definition 6. ***A Binary Tree*** B is a tree in which each vertex has no more than two subordinate vertices. It is composed of three disjoint sets of vertices: a root vertex, a binary tree called its left subtree, and a binary tree called its right subtree [**7**].

Definition 7. ***A Spanning Tree*** T is an acyclic tree which connects all the vertices in the graph *G*. The minimum-spanning problem is a problem of determining the tree *T* whose total weight is minimized [**8**].

Definition 8. ***A Path*** in a graph G is a sequence of vertices $v_1, v_2, \ldots, v_k$. In cyclic mazes, paths can be infinite. The shortest path is a path with the lowest cost between any two given vertices [**9**].

Definition 9. ***A Shortest Path Problem*** is finding for a given graph $G = (V, E)$, a shortest path from any 2 given nodes *u* to *v*. Shortest-paths algorithms typically rely on the property that the shortest path between two vertices contains other shortest paths within it. The shortest path cannot contain any cycles [**5**].

Definition 10. ***A Cell*** is a single vertex in the maze matrix. The position of a cell is given by its id eg. for a cell with a position a_{11} in a grid, we will note the id as "$1\#1$", A cell is also the smallest element of the maze. The cell keeps the following information: its coordinates, the number of neighbours and their's position relative to the cell.

Definition 11. ***A Degree*** of a vertex is denoted as $d(v)$ and it describes the number of adjacent cells [**10**].

Definition 12. ***An Average Degree*** $\bar{d}$ for a given graph is given by [**10**]:

$$\bar{d} = \frac{r}{n-1} \tag{2.3}$$

Where:
n is a number of vertices in the graph.

Definition 13. ***A Dead End*** is defined as a node with a degree $d(v) = 1$. In the maze it is a cell that is linked to only one adjacent node.

Definition 14. ***A Fork*** is defined as a node with a degree $d(v) = 2$. In the maze, it is a cell that is linked to two adjacent nodes.

Definition 15. ***An Intersection*** is defined as a node with a degree $d(v) = 3$. In the maze, it is a cell that is linked to three adjacent nodes.

Definition 16. ***A Cross*** is defined as a node with a degree $d(v) = 4$. In the maze, it is a cell that is linked to four adjacent nodes.

Definition 17. ***A Grid*** in this thesis is considered as a square matrix. Its size defines the size of a maze $n \times m$. The grid keeps the information about each cell and its relative positions in an array.

Definition 18. ***A Move*** is considered as a transition from one cell to one of its closest neighbours. In this work, we are using only NSWE moves presented in Figure 2.3. Diagonal moves are forbidden.

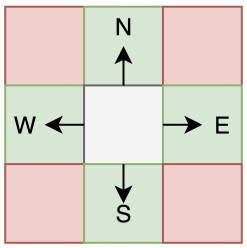


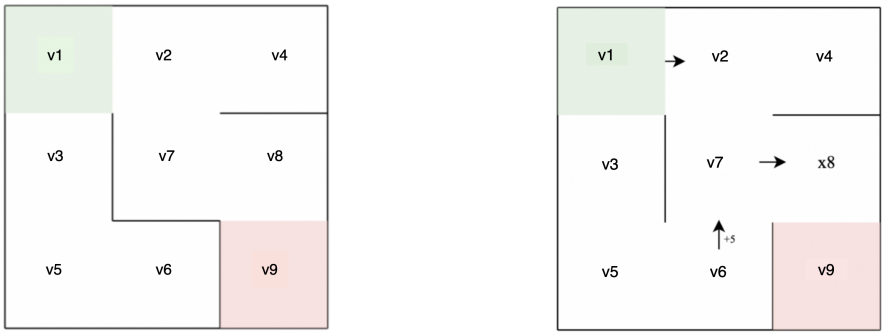
Figure 2.3. Allowed moves.
Source: developed by the author

Definition 19. ***A Maze*** can be considered as a graph, where each intersection is a vertex, and the path between them is an edge.

In this thesis a few types of mazes will be considered:
– perfect maze,
– directed maze,
– cyclic maze.
The perfect maze is a maze with only one path between any two given nodes, a directed maze is be a maze with some paths directed in a certain direction, and a cyclic maze is be a maze with at least one cycle. Different types of mazes are presented in Figure 2.4.

Definition 20. ***A Perfect Maze*** is a maze with only one path between any two given nodes.

Definition 21. ***An Unperfect Maze*** is a maze with more than one path between any two given nodes.



(a) (b)

Figure 2.4. Examples of different mazes. In subfigure (a) an undirected, acyclic maze is shown. In subfigure (b) a maze with a cycle is presented. The maze in subfigure (a) corresponds to the graph in Figure 2.1 (a), and the maze in subfigure (b) corresponds to the graph in Figure 2.1(b). Each cell in a maze is considered as a vertex v.
Source: developed by the author.

Definition 22. ***A Texture*** is a general term that refers to the style of the passages of a maze, such as how long they tend to be and which direction they tend to go. Some algorithms will tend to produce mazes that all have similar textures [**15**].

Definition 23. ***A Canadian Traveller Problem (CTP)*** is a problem of finding the shortest path in a given, known graph with changing conditions in it. The objective of this problem is to find the best solution in the environment which is interfering with malicious intention.

Definition 24. ***A Travelling Salesman Problem (TSP)*** is a problem of finding the shortest path between a given list of nodes in the graph.

2.2. Maze Generation Algorithms

This chapter describes the algorithms that were implemented for this work and were subjected to further comparative analysis. For each algorithm, there is a listing of pseudocode provided along with a picture of the maze generated by it.

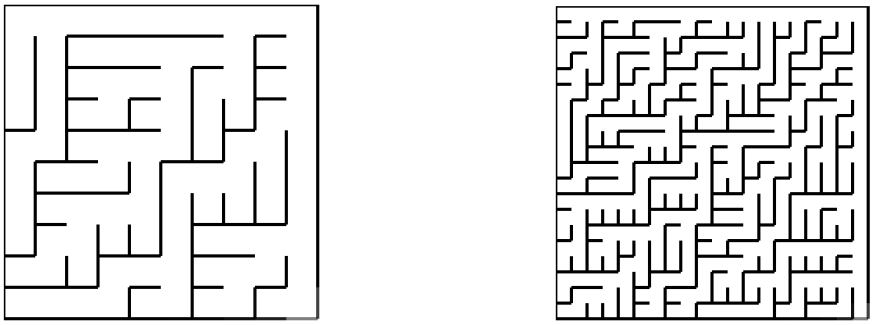
2.2.1. Binary Tree

The Binary Tree algorithm [**16**] is the simplest, fast and efficient algorithm for generating a maze. It does not require a lot of memory because it only needs to remember one cell at any time. In a given grid, for each cell, algorithm decides whether to carve a passage north or east (or any two other directions south/west, south/east etc.) between two adjacent cells. The algorithm produces a diagonally biased perfect maze which, in other words, is a random binary tree. For building the whole maze, the algorithm does not require holding the state of the whole grid. The algorithm only looks at one cell at a time. The time complexity for the Binary Tree generator is $O(|V|)$. A pseudocode for a Binary Tree algorithm is

described in Listing 2.1. Examples of mazes produced by this algorithm are presented in Figure 2.5.

Listing 2.1. Pseudocode for a Binary Tree Algorithm. Developped by the author, based on [**16**].

```
\begin{algorithm}
        \FOREACH cell in the grid
                \STATE let neighbours = [];
                \STATE neighbours.push(cell.north);
                \STATE neighbours.push(cell.east);
                \STATE let index = Math.floor(Math.random() * neighbors.length);
                \STATE let neighbor = neighbors[index];
                \STATE cell.link(neighbor);
        \ENDFOREACH
\end{algorithm}
```



(a) (b)

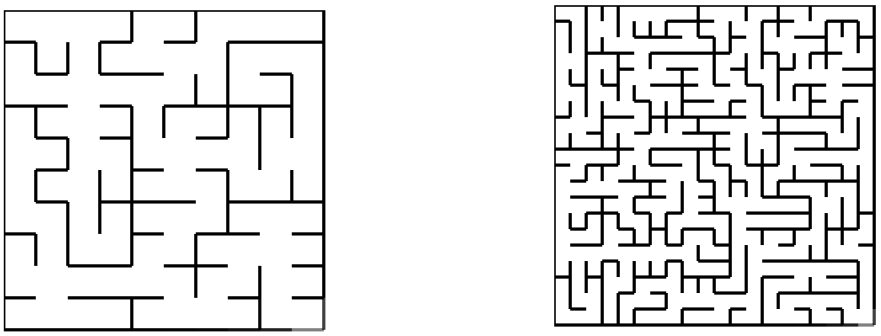
Figure 2.5. Examples of different mazes generated by the Binary Tree algorithm implemented for this work. In subfigure (a) a maze of size 10×10, and in subfigure (b) of size 20×20.
Source: developed by the author.

2.2.2. Aldous-Broder

The Aldous-Broder is a well-known algorithm for generating uniform spanning trees (USTs) based on random walks. This means that the maze is perfect and unbiased [**17**]. The algorithm is highly inefficient but does not require a lot of memory. In a given grid, the algorithm randomly chooses any cell, and for this cell randomly chooses a neighbour and if this neighbour was not previously visited, the algorithm links it to the prior cell. It is repeated until every cell has been visited. To build a spanning tree, the random walk needs to visit every vertex of the graph at least once. The time complexity for the Aldous-Broder generator is $O(|V|^3)$. In Listing 2.2 the pseudocode for an Aldous-Broder algorithm is described. Examples of mazes produced by this algorithm are presented in Figure 2.6.

Listing 2.2. Pseudocode for an Aldous-Broder algorithm. Developped by the author, based on [**17**].

```
\begin{algorithm}
        \STATE let cell = grid.get_random_cell();
        \WHILE unvisited cell in the grid
                \STATE let neighbours = cell.neighbours
                \STATE let index = Math.floor(Math.random() * neighbours.length);
                \STATE let neighbour = neighbours[index];
                \IF neighbour has no links
                        \STATE cell.link(neighbour);
                \ENDIF
                \STATE cell = neighbour;
        \ENDWHILE
\end{algorithm}
```



(a) (b)

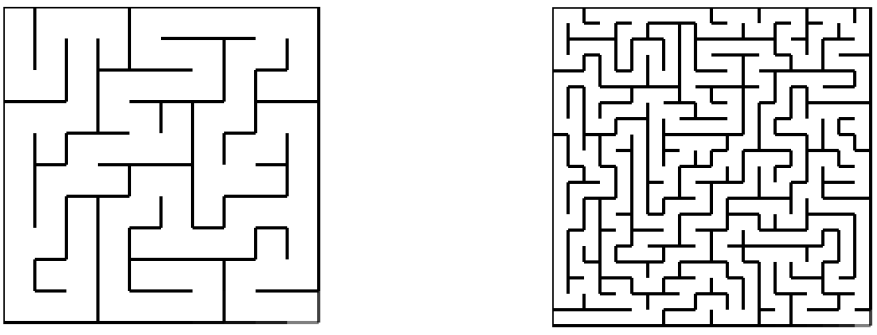
Figure 2.6. Examples of different mazes generated by the Aldous-Broder algorithm implemented for this work. In subfigure (a) a maze of size 10×10, and in subfigure (b) of size 20×20.
Source: developed by the author.

2.2.3. Recursive-Backtracker

The Recursive Backtracker is one of the Depth First Search algorithm (DFS) which may be also used for generating mazes. It generates perfect mazes with a small ratio of dead-ends in a maze. Its main disadvantage is that it requires a lot of memory, so it is not fast or efficient[**18**]. The algorithm starts at the randomly selected cell and carves its way until it must "turn around" and backtracks to the nearest "not carved yet" cell. This process continues until it discovers all the vertices that are reachable from the source vertex. The time complexity for the Recursive-Backtracker generator is $O(|V| + |E|)$. In Listing 2.3 the pseudocode for a Recursive Backtracker algorithm is described. Examples of mazes produced by this algorithm are presented in Figure 2.7.

Listing 2.3. Pseudocode for a Recursive-Backtracker algorithm. Source: developped by the author, based on [**18**].

```
\begin{algorithm}
        \STATE let cell = grid.get_random_cell();
        \STATE let stack = [cell]
        \WHILE stack.length > 0
        \STATE let current_cell = stack[stack.length - 1];
        \STATE let neighbors = current.neighbors();
                \IF neighbors.length == 0
                        \STATE stack.pop()
                \ELSE
                        \STATE let neighbor = neighbors[Random]
                \STATE current.make_link(neighbor)
                \STATE stack.push(neighbour)
\end{algorithm}
```



(a) (b)

Figure 2.7. Examples of different mazes generated by the Recursive Backtracker algorithm implemented for this work. In subfigure (a) a maze of size 10×10, and in subfigure (b) of size 20×20.
Source: developed by the author.

2.3. Maze Solving Algorithms

2.3.1. Breadth-First Search Algorithm

BFS is one of the simplest algorithms for searching a graph. As already mentioned, each maze may be considered as a graph, so from now on we will call BFS a solving algorithm or simply a solver of a given maze. From graph theory, we can state that for a given graph $G = (V, E)$, and distinct source vertex p, BFS explores the edges of G to „visit' each vertex directly connected with p. The algorithm also produces a BFS tree with p root that contains all reachable vertexes. The shortest path between p and any vertex v in G is a simple path in the BFS tree, that is, a path containing the smallest number of edges [**16**]. The BFS implemented in this work is described in Listing 2.4.

Listing 2.4. Pseudocode for a BFS algorithm. Source: developped by the author, based on [**16**].

```
\begin{algorithm}
        \STATE let stack = new Array();
        \STATE stack.push(maze.startcell);
        \STATE let currentcell = startcell;
        \WHILE (!finished){
                STATE let cellAdjacents = grid.getAdjacents(currentcell);
                \FOR EACH cell in cellAdjacents{
                        \IF cell == !visited{
                                \STATE cell == visited;
                                \STATE cell.parent = current;
                                \STATE stack.push(cell);
                        }
                }
                \STATE current = stack.pop();
        }
\end{algorithm}
```

2.3.2. Dijkstra Algorithm

Dijkstra is a solving algorithm for single-source shortest-path problems. We can apply it on a directed graph $G = (V, E)$ with a constraint of no negative edges. It repeatedly chooses the closest vertex in $V - S$ to add to set S. Where *S* is a set of vertices whose final shortest-path weights from the source *p* have already been determined. The algorithm floods the graph so it uses a greedy strategy. The Dijkstra Algorithm implemented in this work is described in Listing 2.5.

Listing 2.5. Pseudocode for a Dijkstra's algorithm.Source: developped by the author, based on [**18**].

```
\begin{algorithm}
        \STATE let distances = new Distances();
```

```
        \STATE let frontier = new Array();
        \WHILE unvisited cell in the grid
        \FOREACH linked cell in frontier
                \STATE linked_cell.distance = cell.distance +1;
                \STATE distances.set_cell(linked_cell);
                \STATE frontier.push(linked_cell);
    \ENDFOREACH
\RETURN distances;
\end{algorithm}
```

2.3.3. A* Algorithm

A^* algorithm is one of the most powerful path-finding algorithms. It uses the same functions derived from the previously described Dijkstra Algorithm. A^* combines the information that Dijkstra's Algorithm uses, meaning choosing the vertex which is close to the starting point and additionally implementing a new type of information, which is heuristic. That means choosing nodes which are estimated to be close to the ending point q. In the standard terminology used when considering A*, $g(v)$ represents the exact cost of the path from the starting point p to any vertex v, and $h(v)$ given by equation (2.5) describes the heuristic estimated cost from vertex v to the goal q. In each loop, the algorithm minimizes the function $f(n)$ given by equation (2.4). The A^* Algorithm implemented in this work is described in Listing 2.6.

$$f(n) = g(v) + h(v) \tag{2.4}$$

Where:
$g(v) = |v - p| + |v - n|$,
$|v - p|$ it is a distance from the starting point p to any current vertex v,
$|v - n|$ it is a distance from any current vertex v to its neighbour n.

The heuristic cost from neighbour vertex v to goal vertex q is given as a:

$$h(v) = |q.x - n.x| + |q.y - n.y| \tag{2.5}$$

Where:
x and y are grid coordinates of vertices.

Listing 2.6. Pseudocode for a A* algorithm. Source: developped by the author, based on [**19**].

```
\begin{algorithm}
        \STATE let openlist = new Array();
        \STATE let closelist = new Array();
        \STATE let startcell = maze.startcell;
        \STATE let goalcell = maze.goalcell;
        \STATE startcell.set_g_score();
        \STATE startcell.set_f_score();
```

```
        \STATE openlist.push(startcell)
        \STATE let finished = false;
        \WHILE (!finished)
                \STATE let currentcell = openlist
                .find_cell_with_lowest_fvalue();
                \STATE let neighbours = currentcell.get_links();
                \IF currentcell == goalcell
                        finished = true;
                        closelist.push(currentcell);
                \ELSE
                \FOREACH neighbour => neighbours
                \IF inEitherList(openlist, closelist)
                        \STATE g_score = calulate_gscore(cell);
                        \STATE f_score = calculate_fscore(cell);
                        \STATE parent = setParent(cell);
                        \STATE openlist.push(cell)
                \ENDIF
                closelist.push(currentcell);
                openlist.remove(currentcell);
        \ENDFOREACH
\end{algorithm}
```

3. Maze solving real live related problems

This chapter presents some of the most interesting and widely used real-life applications of maze-solving algorithms presented in this work. As studying the algorithms might be interesting in itself, the examples below prove that it might be beneficial and necessary when thinking about improving and inventing new technologies. The following examples are largely based on Graph Theory, so they are considered in the context of the algorithms and methods presented in this paper. Although each of these problems has already been thoroughly described and their practical applications exist, they are still problems for which better, more accurate solutions and analyses are sought.

3.1. Shortest Path Problem

The most important application from the point of view of the usefulness of the algorithms described in this paper is the problem of finding the shortest path. All described algorithms can solve it, some under certain conditions. The shortest Path Problem as described by Definition 9 emphasises finding the shortest connection between two vertices in a graph. This concept is easily applicable to many real-life problems, such as: finding the shortest, most convenient path from point A to point B on a map, and solving the routing problem of finding the best path for data package transfer. The concepts of navigation, path planning for robots, or solving routing problems are not yet closed subjects to study. New better, more efficient solutions are being sought.

3.1.1. Navigation

A map may be considered a weighted-directed-cyclic graph. There could be many different paths from city A to city B, some of them use highways, some smaller roads, some roads may go through mountains, and some might be closed, or with heavy traffic. Map navigation is an essential part of people's lives. Studying methods and algorithms which could differentiate the problems by their complexity can contribute to finding new solving methods, which will contribute to creating better, more precise and efficient ways of navigation. Especially in areas where human life or health may be at stake, such as in maritime or inland navigation, where many obstacles must be taken into consideration [**19**].

3.1.2. Path planning

Another example where graph algorithms are widely used and which is a very dynamically developing field of engineering is path planning for robots, drones and autonomous vehicles. Path planning is a robotic problem of finding a path for a robot in a partially known or unknown environment which could also be changing. The most commonly used algorithm in path planning is A^* algorithm [**3**]. The goals of this problem may be different, sometimes it will be finding the shortest route, sometimes the optimal route, sometimes the easiest route, or the fastest route. The most challenging part of path planning is a situation where the environment is not known, and the robot learns about it through sensors trying to get to the destination [**20**]. For most of the practical applications, the scenario of an unknown or partially unknown environment is more relatable and solution-seeking. Path planning consists also of other problems that must be considered besides finding the shortest path. The path planning for robots must evaluate the possibilities of changing the path due to the emergence of additional obstacles. This is the issue of the so-called Canadian Path Traveler described by Definition 23. Self-driving robots, vehicles and drones are already in use on and under the ground, underwater but also in space. The study of solving algorithms is crucial for finding optimal paths of movement and quick assessment in dynamically changing situations. Another important aspect is also a quick assessment of the complexity of a problem to solve. Many aspects of path planning allow good enough solutions (vacuum cleaners), others require more precise ones (warehouse robots), and some require sophisticated solutions (city delivery robots)[**22**]. Therefore it is important to study the best solution approach for each environment.

3.1.3. Networking

Another widely used example which is based on the shortest-path algorithm is OSPF a routing protocol for IP networks. It is widely used in bigger TCP/IP internetwork, to exchange routing information. It is based on the Dijkstra algorithm, a router sets itself as a root of a network tree and computes the shortest path between each pair of nodes in the network. The SPF algorithm must ensure that routing information is quickly assessed in case of routers are being moved or going down. This feature is known as Fast Convergence. The SPF algorithm also guarantees that the routing tables contain the shortest (least-cost) paths and that routing loops are excluded [**23**].

3.2. Other applications of graph algorithms

3.2.1. Web page scraping

Another very popular use of graph algorithms, such as BFS, is web scraping [**24**]. Web scraping is a method of web page semi-structured data retrieval. In recent years, it has become a powerful tool for many companies to obtain large amounts of information at a low cost. It is an area that is dynamically changing all the time and where two opposing forces are at work. On the one hand, giants such as Google or Facebook, which are collecting data from millions of users, try to prevent other companies from using their data out of charge. On the other hand, companies are trying to collect the data that is made available

to the public on a mass scale. It creates a need for constant assessment, adaptation and improvement of crawling algorithms.

3.2.2. Video games

In the video games industry graph algorithms are commonly used, and sometimes are the backbones of the project itself. There are two main areas where graph algorithms are used. First, for world map creation, maze-generating algorithms are used for creating interesting and challenging maps. Usually, worlds built in computer games are very large and complex. The game world maps have several basic functions, firstly they guide all character's paths, secondly, they drive the narrative and game mechanics by challenging their players with movement and completing missions in a timed and space chronology. In the modern world of game development, the issue of programming the movement of all characters in the game is a problem closely related to programming maze-solving algorithms. These challenges put a lot of emphasis on the effectiveness of the solutions. Games must run in real-time, usually with very strictly limited CPU capabilities. At the same time, the problems must be interesting and challenging enough to satisfy even the most demanding players. All algorithms described in this work are used in game development [**25**].

4. Maze complexity problems

One of the main purposes of this thesis is to discuss the complexity of a maze. This chapter will provide a variety of maze and graph complexity definitions which are implemented and compared in Chapter 5. Firstly the methods of complexity measure, derived from graph theory will be discussed followed by some existing concepts of measuring the complexity of a maze. Thinking about a maze and its complexity all different types of features and problems might be considered. It may be the difficulty of finding the way out, or difficulty of moving from point A to point B, or the difficulty of generating a particular maze. This chapter will provide a theoretical background for studying the maze complexity and in Chapter 5 detailed analysis based on examples will be provided.

4.1. Complexity measures in Graph Theory

In this section approaches derived from graph theory, which describe and measure the complexity of a graph are presented. Conventionally, graph complexity in graph theory is evaluated by a degree distribution, a clustering coefficient, an edge density. Another approach derived from classical information theory is to generate graphs with some particularities while being random in all other aspects and then compare and decide whether a particular characteristic is typical among a group of graphs or not. There is also a recent, advanced idea to use a principle of maximum entropy to estimate the algorithmic complexity of a graph. The main idea of the maximum entropy concept is that the more statistically random graph is, the more typical. [**11**]. Studying the complexity of a graph is an important part of understanding it. By studying the complexity, we are acquiring knowledge about the system, how it could evolve, what are bottlenecks and variabilities, and how we can solve the problems posed by the system. Each application may understand the complexity differently. Each graph system, network and maze will cause different challenges and solutions to it.

4.1.1. Outlining graph parameters

Definition 25. ***A Degree Distribution*** $P(k)$ is defined as a proportion of vertex with a degree k to all vertices in a graph by the equation (4.1).

$$P(k) = \frac{n_k}{n} \tag{4.1}$$

Definition 26. ***A Clustering Coeficient*** C_i is defined as a proportion of vertex links to vertex possible links. The coefficient for an undirected graph might be given by $C_i = \frac{k_i(k_i-1)}{2}$ where k_i are the

neighbours of vertex v_i. The average clustering coefficient is given by:

$$\bar{C} = \frac{1}{n}\sum_{n=1}^{n} C_i \tag{4.2}$$

Definition 27. ***A Graph Entropy*** *"Graph entropy represents information-theoretic measures for characterizing networks quantitatively"[**12**].* It combines graph information and probabilistic distribution of vertices [**13**].

4.1.2. Shannon's entropy

An Adjacency matrix (Definition 3) of a maze may be also represented as a string. Adopting this, another maze complexity measure may be introduced. Shanon entropy derives directly from Boltzmann entropy in thermodynamics. "Shannon's concept of information entropy quantifies the average number of bits needed to store or communicate a message."[**11**]. Studying complexity, the Shannon entropy measures how complex the string of a graph problem must be to avoid losing any information about its state. The main concept is that the information is built by n different symbols and can not be stored in less than $log(n)$ bits. Shanon entropy $H(M)$ of the object $M(R, p(x_i))$ is given by (4.3)[**11**]:

$$H(M) = -\sum_{r=1}^{r} p(x_i)\log_2 p(x_i) \tag{4.3}$$

Where:
R, is a set of possible outcomes, e.g. All possible adjacency matrix of size m,
$p(x_i)$ is a probability of outcome R,
$r = |R|$.

4.2. Maze complexity measures

This section, summarize the characteristics which impact the complexity of a maze, such as maze size, average path length, density, and McClendon's complexity measure. It provides an overview of different approaches and tries to compare them. All parameters presented in this chapter are evaluated in Chapter 5.

4.2.1. Independant maze parameters

Size is one of the most indisputable complexity factors of a maze. A maze is described as per Definition 19. In Figure 4.1 two mazes with different sizes are presented. It is almost too easy to think that a small maze is a simple maze, and a huge maze is a difficult one.

(a) (b)

Figure 4.1. Examples of different size mazes. Figure (a) it is an example of the Aldous-Broder maze of size 6×6, Figure (b) of size 18×18
Source: Developed by the author, based on Aldous-Broder Algorithm Listing 1.2.

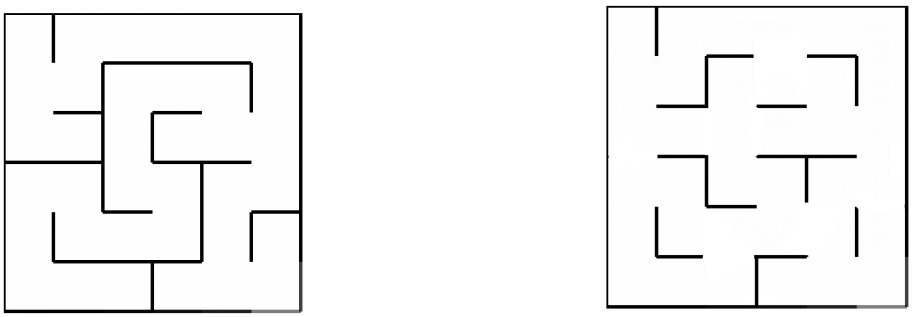
Another key characteristic determining the complexity of a maze is the average length $\bar{p}_l$ of the paths in a maze. The longer the path, the bigger the risk of following a faulty road to a solution. Path is given by Definition 8. Its beginning is always a start cell and it leads to each dead-end in acyclic mazes. In cyclic mazes, paths can be infinite according to Definition 4. In Figure 4.2 two mazes of the same size but different average path lengths are presented.

(a) (b)

Figure 4.2. Examples of different average path length mazes. In figure (a) it s an example of the Aldous-Broder maze with $\bar{p}_l = 9.42$. Figure (b) it s an example of a Binary Tree maze with $\bar{p}_l = 10.8$
Source: Developed by the author, based on Aldous-Broder Algorithm Listing 1.1 and Binary Tree Algorithm 1.2.

Density for an acyclic graph is given by the equation (2.1), and density for a cyclic maze is given by the equation (2.2). It describes the ratio between the number of all possible connections and the existing number of connections (edges).

In Figure 4.3 two mazes with different densities are presented.



(a) (b)

Figure 4.3. Examples of different density mazes. In Figure (a) maze with density r $= 0.40$ is presented, and in Figure (b) maze with density r $= 0.50$.
Source: Developed by the author, based on Aldous-Broder Algorithm Listing 1.1

4.2.2. McClendon's measure

Few sources are indicating a quantitative study of the measure of maze complexity. One of the most cited works in this field is a McClendon [**14**] study of maze difficulty and complexity. McClendon's work treats maze complexity and difficulty in a continuous measure using continuum theory. The main presuppositions of the work are that the maze is a perfect maze type and there are two distinguished pairs of points (p, q) in the maze M called gates. Where p is an entrance and q is an exit. Hallways h build a maze. Where hallways are a subset K of M with the $d(v) = 2$. A subset $W_h = w_1, w_2, \ldots, w_n$ of h incorporates all points of h. A trail is a path in the maze built by hallways. The branch is any trail intersecting the solution S of a maze. Each branch in M is connected to S by a point v_i in I which is an intersections set $I = v_1, v_2, \ldots, v_n$. The McClendon's complexity of a hallway h is given by:

$$\gamma(h) = D(h) \sum_{i=1}^{i} \frac{\theta(w_i)}{d(w_i) \cdot \pi} \tag{4.4}$$

Where:
$D(h)$ is an arclength of h,
$\theta(w_i)$ is the absolute value of the difference in the radian measures between the directions $V(t_i)$ and $V(t_{t+1})$,
$d(w_i)$ is a length of a arc between w_{i-1} and w_i in W_h.

The McClendon's complexity of a maze M is given by:

$$\gamma(M) = \log\big[\gamma(S) + \sum_{i=1}^{i} \gamma(B_i)\big] \tag{4.5}$$

Where:
$\gamma(S)$ is a complexity of a solution of the maze,

$\gamma(B_j) = \sum_{i=1}^{i} \gamma(h_i)$ is a complexity of a branch B_j.

Using equation (4.5) requires knowledge about the solution of the maze. To avoid this, we should use the extrinsic approximation of the above method which is given by:

$$\gamma(M) \approx \log[\sum_{i=1}^{i} \gamma(h_i)] \tag{4.6}$$

Where:
$\gamma(h_i)$ is the complexity of an i^{th} hallway h_i.

In this thesis we are using the square grid to generate and solve mazes. As a result of a uniform grid, the McClendon measure will be simplified, and calculated as:

$$\gamma(M) \approx \log[\sum_{i=1}^{i} \frac{h(i)_l \cdot \mathcal{T}}{2}] \tag{4.7}$$

Where:
$h(i)_l$ is the total length of the hallway,
$\mathcal{T}$ is the total number of L turns in the hallway, and each L turn is 90°.

4.2.3. Other approaches to delineate maze complexity

In sections 4.1 and 4.2 different quantitative measures to define maze complexity were discussed. In this section, two descriptive methods determining the difficulty of the maze are presented. Biased mazes and the time complexity of maze generators may be considered as premises for reduced algorithmic randomness. This work will try to verify if those premises correlate with the time required to solve a maze, or it s McClendon's complexity.

Time complexity of maze generators

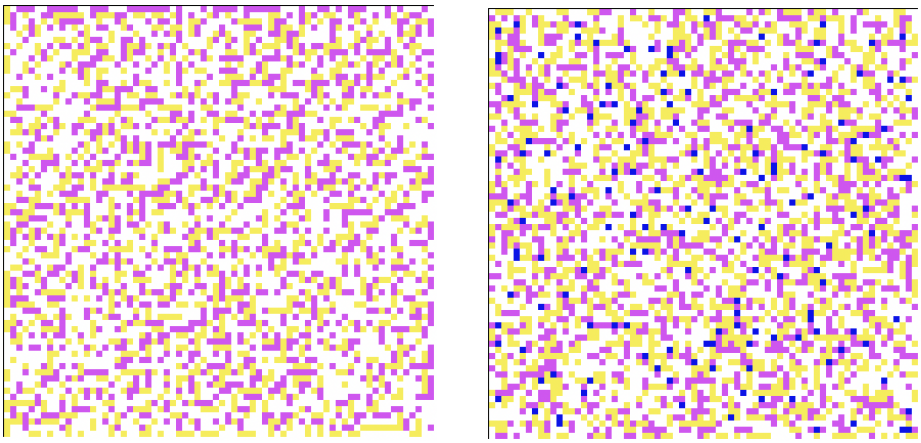
Looking at the time complexity of maze generators we can compare them and analyze whether the solution time depends on the time complexity. In Table 4.1 time complexity of different maze generators is presented. The analysis of the relation between time complexity and solution time is presented in Chapter 5.

Maze Generator	Time Complexity
Binary Tree[**16**]	$O(\|V\|)$
Aldous-Broder[**17**]	$O(\|V\|^3)$
Recursive Backtracker[**18**]	$O(\|V\| + \|E\|)$

Table 4.1. Time complexity of maze generating algorithms, represented in the Big O notation.
Source: created by the author based on [16,17,18].

Uniquness and distinctiveness of a maze

Another approach to describing a maze might be evaluating its uniqueness and distinctiveness. For this purpose, it is possible to study how complicated the algorithm generating a given maze must be and how statistically often a specific set of features occurs. One of the methods may be the parameterization of the maze by classic measures such as density distribution or average path length and testing how often they appear in a specific configuration. Another the method could be to look for biased features. Biased features are some peculiarities of the maze which are visible repeating structures. Good examples of a biased maze might be a Binary Tree Algorithm, which tends to create visible diagonal intersections and two perpendicular corridors at the edges of the maze presented in Figure 4.4(a). On the other hand, there are mazes generated with Aldous-Broder Algorithm, where no specific biases occurs. The unbiased texture of Aldous-Broder maze is visible in Figure 4.4(b).



(a) (b)

Figure 4.4. Examples of different textures in mazes. Figure (a) and (b) present a degree distribution applied directly on a maze where all edges were removed. Each colour defines other cell degrees (Definition 11): yellow - a dead-end, white - a fork, pink - an intersection and blue - a cross. In Figure (a) maze with a visible the diagonal texture was generated with Binary Tree Algorithm. In Figure (b) maze was generated with the Aldous-Broder Algorithm without any visible texture.
Source: Mazes were generated by the author uses the algorithms described in Listings 2.1 and 2.2.

5. Results, Analysis and Discussion

This chapter is a combined presentation of collected results, their analysis and a discussion. The first section is a presentation of the gathered results. The second section, divided into three subsections, introduces the evaluation of questions Q1, Q2 and Q3 respectively. The last section presents the conclusions of gathered data and performed analysis.

5.1. Results

All results presented in this section are derived from two data sets. Each data set was collected separately. The first data set is called variant 1 and the second is called variant 2. Both variants contain 2700 different maze problems, generated by Binary Tree, Aldous-Broder and Recursive-Backtracker algorithms. The first variant is a data set of mazes with only one solution, variant 2 on the other hand contains mazes with more than one solution. Each maze was a separate instance of a class Grid(), and the execution time of the solution was measured in the separated solver's method without the interference of any additional process. For data collecting all visualisation features were disabled and all data were saved during each iteration to the .csv file. Variants produced 900 randomly applied sizes. However 27% mazes from variant 2 do not have a solution. To avoid a situation of a higher proportion of unsolvable mazes, there was applied a small, 3%, ratio of directed cells, and a big, 50% ratio of added connections between cells. All data were collected on MacBook Pro with an Apple M1 microchip, 8GB RAM and 11.6 macOS Big Sur operating system. All the figures used for analysis were made using the Python GUI application Orange v.3.33.

Each maze must fulfil the following presumptions:
− two dimensional, minimum size 5×5, maximum size 80×80,
− staring point coordinates [0,0], goal point [mazeSize.x - 1, mazeSize.y - 1],
− weight of each edge equals 1.

Variant 1 specific presumptions:
− perfect maze (Definition 20),
− no directed edges,
− each maze is solvable, there is a direct path from starting point to the goal point.

Variant 2 specific presumptions:
− Unperfect maze (Definition 21),

– 50% of randomly added links to the randomly selected cells in a maze grid,
– 3% of south direction added to randomly selected cells in a maze grid,
– mazes do not have to be solvable.

Majority of the presented results are characterized by high distribution skewness. For skewed distributions, the following values are given in the tables: average, SD, Median, Skewness, W (Shapiro-Wilk test result)[**34**]. For the presented scatter plots, different maze generators are distinguished by a different colour: blue for the Aldous-Broder algorithm, red for the Recursive-Backtracker algorithm and green for the Binary Tree algorithm. The adopted confidence level is $\alpha = 0.05$ and the critical level of the test is $p < 0.001$. A skewness lower than 1.5 is considered small. This indicates that the magnitude of the difference between the sample distribution and the normal distribution is also small. A skewness higher than 1.5 is considered as medium or large, indicating a medium or large difference.

5.1.1. Paths Length

Table 5.1 presents the results of the average and median of path length for generated mazes in variant 1. According to Definition 8, paths in cyclic mazes may be infinite, therefore there is no data presented for variant 2. For Aldous-Broder and Binary Tree generators, the average and SD may be considered as reliable measures due to the small skewness. However, for the Recursive-Backtracker the median is a more reliable measure due to the big skewness.

Table 5.1. Descriptive statistics of path length for different maze solvers of different maze generators. Source: developed by the author.

	Calculated Parameters				
Maze Generator	Average	SD	Median	Skewness	W
Aldous-Broder	103	52	99	0.0775	0.960
Binary Tree	59	25	57	0.329	0.982
Recursive-Backtracker	311	227	263	1.469	0.880

5.1.2. Steps to solve

Tables 5.2 and 5.3 present descriptive statistics of steps needed to solve a maze. Data is split by the maze generators and maze solvers.

Table 5.2. Variant 1: descriptive statistics of number of steps needed to solve different maze generators by different maze generators. Source: developped by the author.

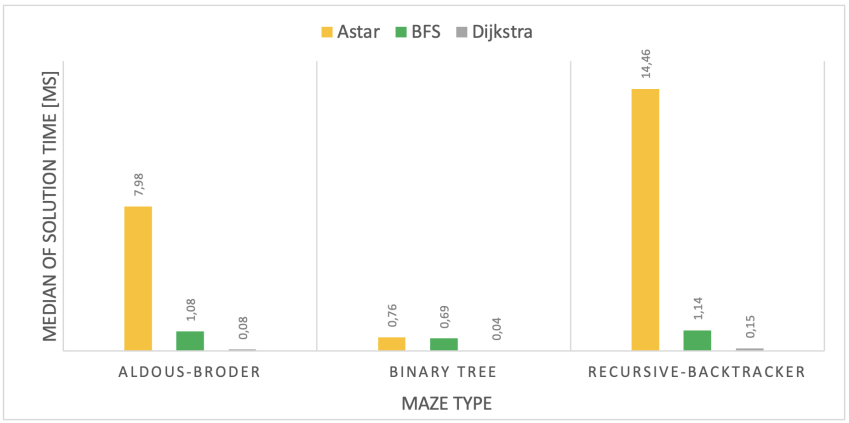
		Calculated Parameters				
Maze Solver	Maze Generator	Average	SD	Median	Skewness	W
Astar	Aldous-Broder	592	557	434.5	2.155	0.776
	Binary Tre	77	30	80	0.066	0.989
	Recursive-Backtracker	972	893	711	1.670	0.832
BFS	Aldous-Broder	1170	949	956	1.261	0.890
	Binary Tre	807	753	574	1.131	0.864
	Recursive-Backtracker	1227	1037	958	1.264	0.874
Dijkstra	Aldous-Broder	1773	1366	1361	0.862	0.916
	Binary Tre	1546	1254	1181	1.144	0.894
	Recursive-Backtracker	1802	1343	1455	0.811	0.920

Table 5.3. Variant 2: descriptive statistics of number of steps needed to solve different maze generators by different maze generators. Source: developped by the author.

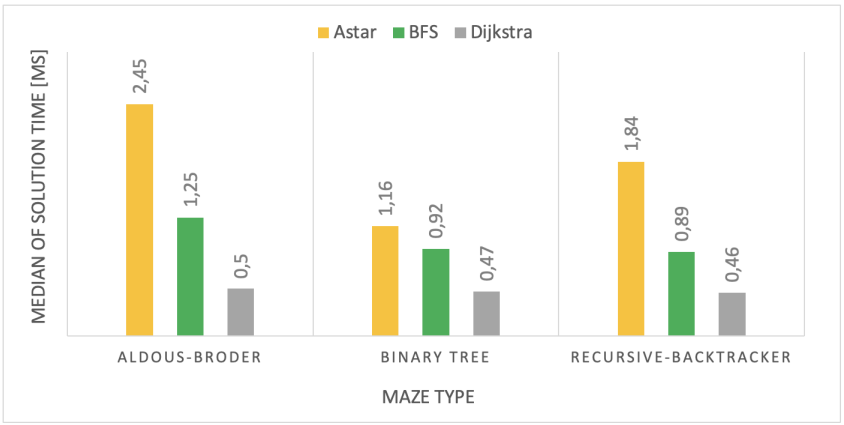
		Calculated Parameters				
Maze Solver	Maze Generator	Average	SD	Median	Skewness	W
Astar	Aldous-Broder	155	67	153	0.359	0.989
	Binary Tre	94	43	91	0.666	0.973
	Recursive-Backtracker	163	74	156	0.540	0.982
BFS	Aldous-Broder	973	875	689	1.500	0.848
	Binary Tre	709	810	428	2.366	0.717
	Recursive-Backtracker	979	890	715	1.647	0.840
Dijkstra	Aldous-Broder	1242	1118	800	1.268	0.865
	Binary Tre	1183	1066	840	1.418	0.852
	Recursive-Backtracker	1148	1065	826	1.713	0.832

5.1.3. Solution Time

Figure 5.1, Table 5.4 and Table 5.5 presents an overview of the descriptive statistics of measured solution time.



(a)



(b)

Figure 5.1. (a) presents Variant 1 and (b) presents Variant 2: median of solution time for different maze generators solved by different maze solvers.
Source: developed by the author.

Table 5.4. Variant 1: Descriptive statistics of solution time for different maze solvers of different maze generators. Source: developped by the author.

		Calculated Parameters				
Maze Solver	Maze Generator	Average	SD	Median	Skewness	W
Astar	Aldous-Broder	20.74	40.08	7.98	3.96	0.493
	Binary Tree	1.33	1.24	0.76	1.57	0.817
	Recursive-Backtracker	47.10	83.67	14.46	3.40	0.575
BFS	Aldous-Broder	1.29	1.05	1.08	1.57	0.886
	Binary Tree	0.92	1.13	0.69	7.07	0.578
	Recursive-Backtracker	1.38	1.18	1.14	2.03	0.853
Dijkstra	Aldous-Broder	0.11	0.10	0.08	1.99	0.743
	Binary Tree	0.08	0.09	0.04	1.80	0.663
	Recursive-Backtracker	0.20	0.24	0.15	6.70	0.533

Table 5.5. Variant 2: Descriptive statistics of solution time for different maze solvers of different maze generators. Source: developped by the author.

		Calculated Parameters				
Maze Solver	Maze Generator	Average	SD	Median	Skewness	W
Astar	Aldous-Broder	2.52	1.68	2.45	0.73	0.955
	Binary Tree	1.41	1.08	1.16	0.79	0.917
	Recursive-Backtracker	2.20	1.67	1.84	1.25	0.908
BFS	Aldous-Broder	1.35	1.07	1.25	0.87	0.923
	Binary Tree	1.07	0.99	0.92	1.87	0.845
	Recursive-Backtracker	1.19	1.06	0.89	1.29	0.876
Dijkstra	Aldous-Broder	0.72	0.64	0.50	1.53	0.846
	Binary Tree	0.71	0.71	0.47	2.77	0.759
	Recursive-Backtracker	0.64	0.58	0.46	1.91	0.815

5.1.4. McClendon's complexity

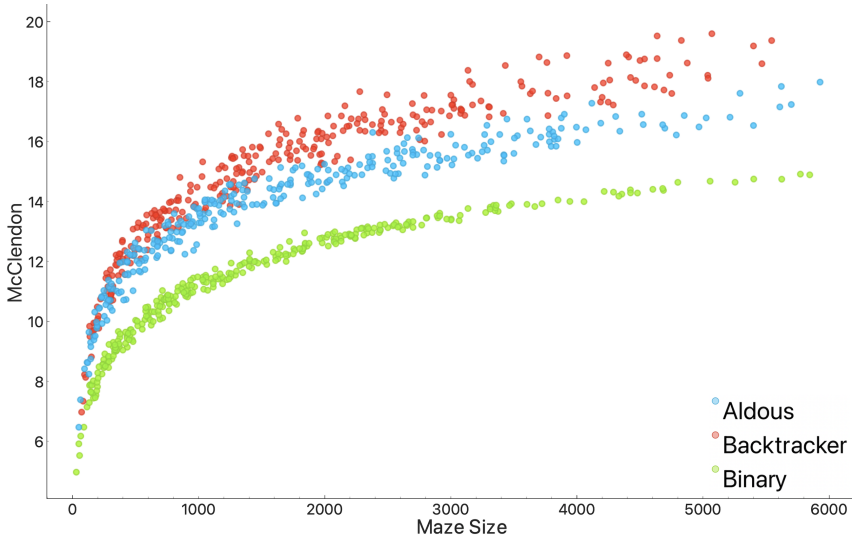
Tables 5.6 and 5.7 present the average and median of the complexity measure for different maze generators. Modulo of skewness is lower than 1.5 which states the set distribution is very close to the normal distribution, therefore, the average and SD may be considered the reliable measure. Figures 5.2(a) and 5.3(a) present scatter plots of McClendon's complexity versus maze size. In Figures, 5.2(a) and 5.2(b), a box plot of McClendon's complexity distributions are showing mean complexity with standard deviation and maximal and minimal values.

Table 5.6. Variant 1: Descriptive statistics of McClendon's complexity for different maze generators. Source: developped by the author.

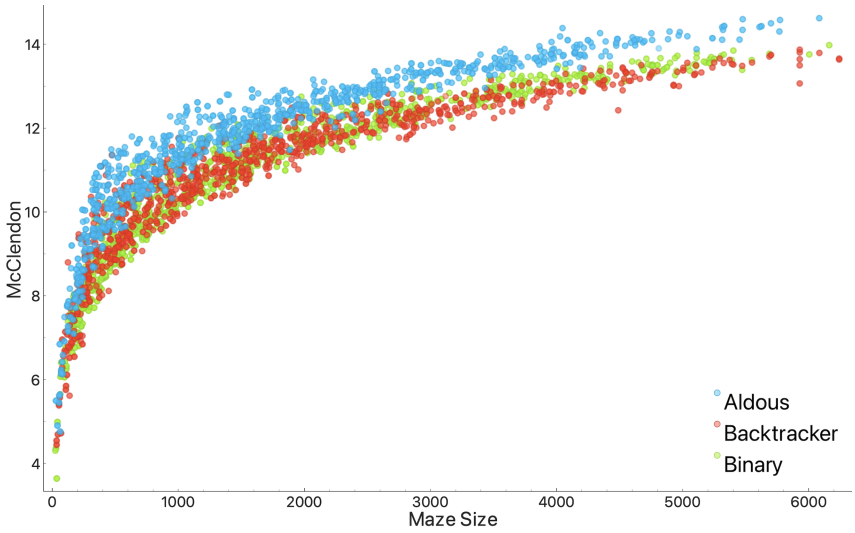
	Calculated Parameters				
Maze Generator	Average	SD	Median	Skewness	W
Aldous-Broder	13.83	2.12	14.13	-0.67	0.965
Binary Tree	11.36	1.98	11.49	-0.51	0.972
Recursive-Backtracker	14.94	2.48	15.39	-0.58	0.971

Table 5.7. Variant 2: Descriptive statistics of McClendon's complexity for different maze generators. Source: developped by the author.

	Calculated Parameters				
Maze Generator	Average	SD	Median	Skewness	W
Aldous-Broder	11.61	1.78	11.92	-0.96	0.944
Binary Tree	10.72	1.91	11.00	-0.55	0.964
Recursive-Backtracker	10.78	1.77	11.05	-0.82	0.956



(a)



(b)

Figure 5.2. (a) presents Variant 1 and (b) presents Variant 2: a scatter plot of McClendon's complexity versus maze size.
Source: developped by the author.

5.1.5. Shanonn's Entropy

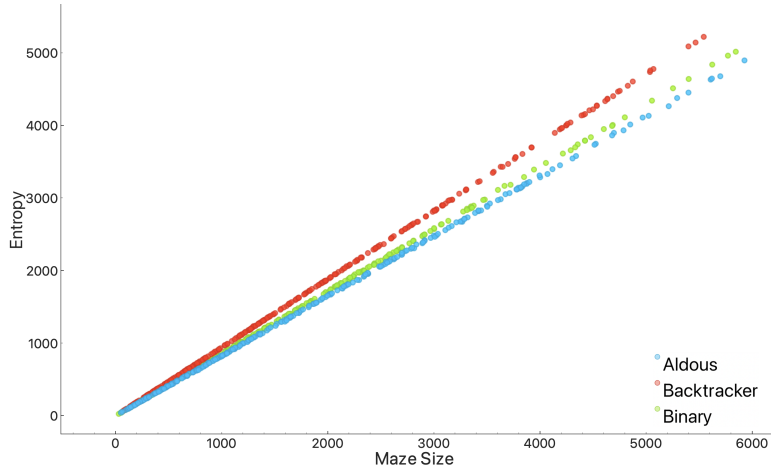
Tables 5.8 and 5.9 present the average and median of the complexity measure for different maze generators. Despite the skewness is smaller than 1.5 and stating that the set distribution is close to the normal distribution, due to huge SD the median may be considered a more reliable measure. In Figures 5.3(a) and 5.3(b), scatter plots of Shannon's entropy versus maze size are presented.

Table 5.8. Variant 1: Descriptive statistics of Shannon's entropy for different maze generators. Source: developped by the author.

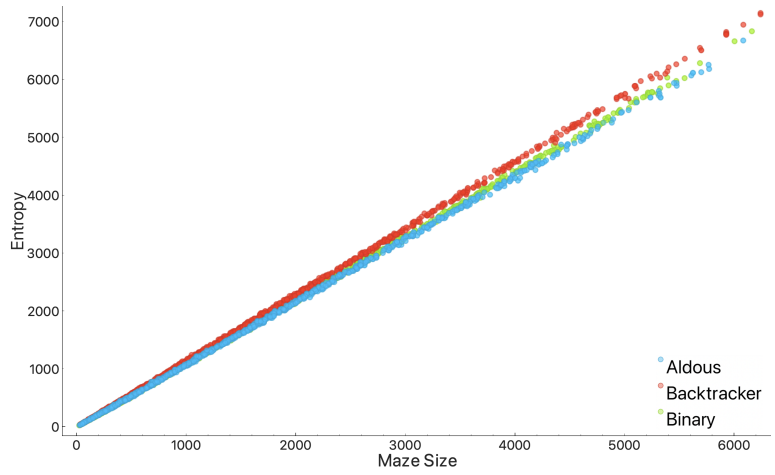
	Calculated Parameters				
Maze Generator	Average	SD	Median	Skewness	W
Aldous-Broder	1462	1125	1125	0.86	0.916
Binary Tree	1324	1074	1015	1.14	0.894
Recursive-Backtracker	1696	1264	1369	0.809	0.920

Table 5.9. Variant 2: Descriptive statistics of Shannon's entropy for different maze generators. Source: developped by the author.

	Calculated Parameters				
Maze Generator	Average	SD	Median	Skewness	W
Aldous-Broder	1802	1429	1466	0.926	0.909
Binary Tree	1845	1506	1468	0.931	0.901
Recursive-Backtracker	1913	1562	1512	1.061	0.897



(a)



(b)

Figure 5.3. (a) presents Variant 1 and (b) presents Variant 2: a scatter plot of Shannon's entropy versus maze size.
Source: developped by the author.

5.1.6. Degree Distribution

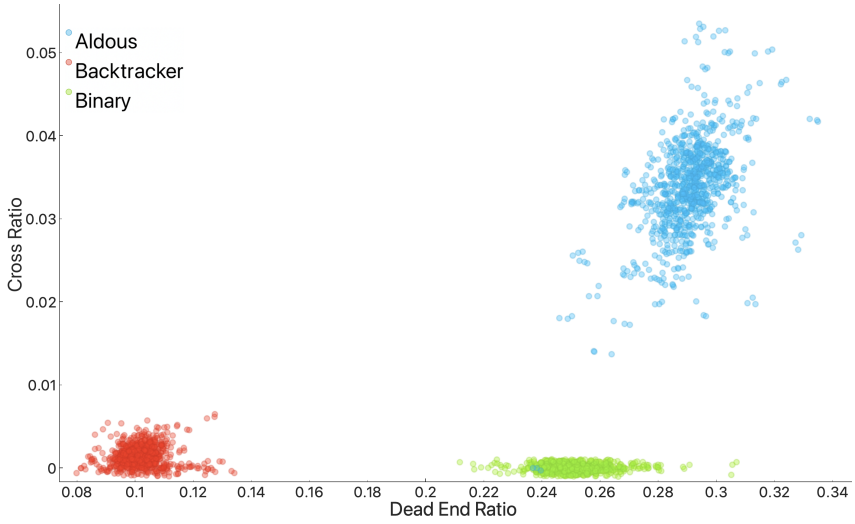
In Tables 5.10 and 5.11 statistical information of degree distribution for different maze generators. Degree distribution for both, variant 1 and variant 2 is defined by the normal distribution so the average and SD may be considered reliable measures. Figures 5.4(a) and 5.4(b) present a scatter plot of a cross ratio versus dead-end ratio.

Table 5.10. Variant 1: Degree distribution for different maze generators: Binary Tree, Aldous-Broder and Recursive-Backtracker. Source: developped by the author.

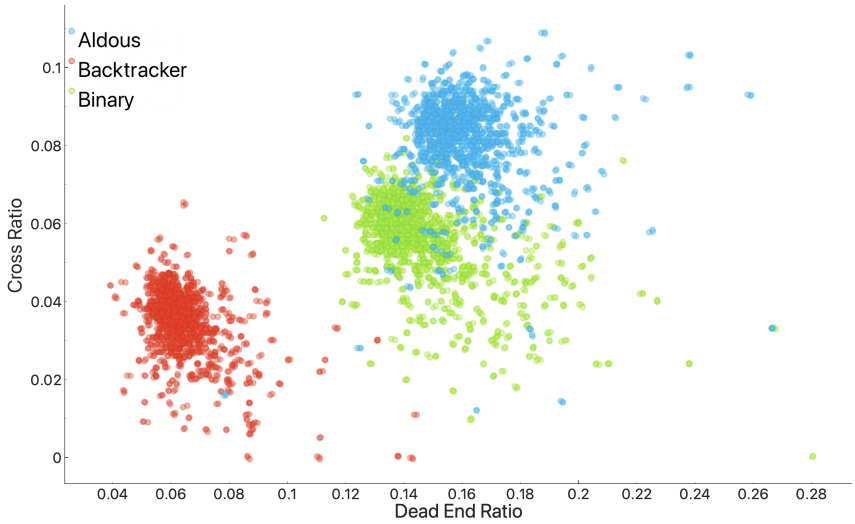
	Maze Generator		
Degree Distribution	Aldous-Broder	Recursive-Backtracker	Binary Tree
Dead-end Ratio	0.291 ± 0.011	0.1021 ± 0.0065	0.2512 ± 0.0097
Fork Ratio	0.455 ± 0.020	0.800 ± 0.012	0.501 ± 0.019
Intersection Ratio	0.220 ± 0.011	0.096 ± 0.011	0.248 ± 0.010
Cross Ratio	0.034 ± 0.006	0.001 ± 0.001	0.000 ± 0.000

Table 5.11. Variant 2: Degree distribution for different maze generators: Binary Tree, Aldous-Broder and Recursive-Backtracker. Source: developped by the author.

	Maze Generator		
Degree Distribution	Aldous-Broder	Recursive-Backtracker	Binary Tree
Dead-end Ratio	0.169 ± 0.025	0.067 ± 0.013	0.146 ± 0.017
Fork Ratio	0.427 ± 0.034	0.575 ± 0.026	0.436 ± 0.019
Intersection Ratio	0.328 ± 0.025	0.325 ± 0.024	0.361 ± 0.021
Cross Ratio	0.077 ± 0.021	0.034 ± 0.010	0.057 ± 0.010



(a)

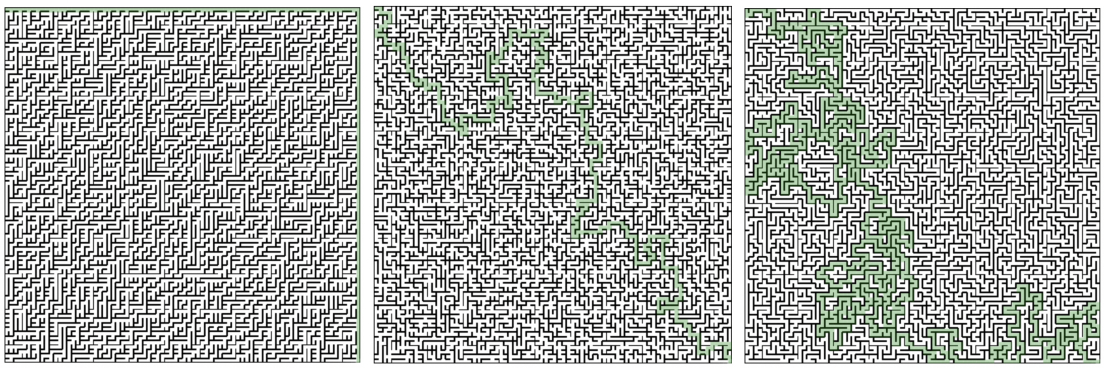


(b)

Figure 5.4. (a) presents Variant 1 and (b) presents Variant 2: A scatter plot of cross ratio versus dead-end ratio. Source: developped by the author.

5.2. Practical analysis of maze generators and maze solvers

In this section, all algorithms described in Chapter 4 are assessed in terms of their runtime and parameters of generated solutions. Three algorithms described in Chapter 4 were evaluated: Dijkstra, A^* and BFS, each algorithm was tested in the same way. The runtime measurement program worked as follows, the program generated a random-sized maze using one of the three algorithms: Binary Tree, Recursive Backtracker and Aldous-Broder. Then each solving algorithm one by one was applied to solve the same problem. Mazes were generated with randomly assigned sizes ranging from 5×5 to 80×80. The main assumption was to create a rectangular maze with the source cell p at the left top corner, and the goal cell q at the right bottom corner of the maze grid. All solvers could only use the NSWE moves described in Chapter 4. Maze problems usually have quick access to basic heuristic functions because of a graph implemented as a grid. Because of the assumption that only the NSWE moves are allowed the heuristic method $h(v)$ applied in algorithm A^* was the Manhattan Distance. The purpose is to compare both the maze generators and the solvers and build a framework which could classify which algorithms comply best with each other. Although the runtime of both, the generating and solving algorithms were measured, it was not the purpose of this work to minimize it. Therefore, the algorithms were implemented in JavaScript. It is beyond discussion that the implementation in a more low-level language would be more efficient, and could lead to building and solving bigger mazes. However, the choice of using Java Script for algorithm implementation was dictated by the eagerness of building a web application depending on and harnessing the results of this work. The reprezentative set of solved mazes is presented in Figure 5.5.



(a) (b) (c)

Figure 5.5. Examples of different, perfect, 80×80 mazes with applied Dijkstra solution. In subfigure (a) there is a Binary Tree maze, in (b) an Aldous-Broder maze, and in (c) Recursive-Backtracker
Source: developed by the author

5.2.1. Path and time comparison for different maze solvers

Variant 1

The algorithm which generates mazes with the longest path median is Recursive-Backtracker, which

is 227. The shortest median of path is generated by Binary Tree algorithm, which is 57. A significant value of the average path length might indicate a higher number of longer paths in the maze. That may results in a longer solution time due to the greater number of long side paths, i.e. paths that are not a solution. This is also confirmed by the results of the measured number of steps and time, which are also the biggest for Recursive-Backtracker, and the smallest for Binary Tree for every solver as shown in Tables 5.2, 5.3, 5.4 and 5.5. What is noticeable for this data set is a very big standard deviation, which might be caused by a big maze size range. The results for Aldous-Broder mazes fall between Recursive-Backtracker and Binary, as well as for solution time, path length and steps needed to solve.

Solution time for the Dijkstra solver is the shortest regardless of the size and maze type. Dijkstra's median of solution time does not exceed 0.2 ms for each type of maze. The longest solution time was measured for Astar solver for which the worst-case is 14.46 ms for solving a Recursive-Backtracker maze. However, the Astar solution time varies the most depending on the maze generator which is visible in Figure 5.1(a) and Table 5.4

Variant 2

There is an observable drop in the median of the number of steps and solution time for Astar and BFS solvers in mazes with cycles. Astar, worst-case solution time, for a Recursive-Backtracker maze, is only 1.84 ms in comparison to 14.46 ms for a perfect maze. The BFS solution time is also lower than for the perfect mazes, but not that significant. BFS worst-case solution time for a Recursive-Backtracker maze is 0.89 ms in comparison to 1.14 ms for a perfect maze. The Dijkstra algorithm was the fastest solver for all maze generators.

The presented results in some way contradict the conclusions contained in the paper [**31**], where the authors examined the same solvers but only small and medium-sized grids, with a maximum size of 32x32 and only square ones. The popular opinion [**32**] that the A^* algorithm must always be the fastest solver wasn't confirmed in this work. The additional calculations which Astar is performing while searching the solutions in the case of this work, significantly increased the solution time. It is true that the number of steps performed is lower compared to other solvers in the case of mazes with cycles and directed, but this does not affect the time of the solution.

5.2.2. Analysis of parameters affecting maze complexity

Variant 1

The evaluation of McClendon's complexity shows that the three types of maze generators can be characterized by this measure. The results confirm the conclusions of path and solution time evaluation. The most complex maze type is a Recursive-Backtracker with average complexity equal to 14.94 ± 2.48, the easiest type is Binary-Tree with average complexity equals 11.36 ± 1.98. The McClendon's complexity is based on hallway length so the results are consistent with what was said earlier about Recursive-Backtracker mazes.

Another complexity measure which has the potential to assess maze complexity is Shannon's entropy. There are few sources applying entropy to the mazes problems, as entropy is rather considered for bigger

networks. However, besides the big standard deviation, Shannon's entropy seems to correctly evaluate maze complexity in terms of its solution time. Figure 5.3(a) presents the data for maze entropy versus maze size. The relation is linear, however, each maze type can be easily distinguished. Moreover, the hierarchy of entropy corresponds to solution time. The bigger the entropy the longer the solution time. The most difficult mazes are Recursive-Backtracker and the easiest are Binary Trees. The implemented entropy is based not on the length path but rather a degree distribution, which allows concluding that it is the most reliable maze complexity measure that McClendon's.

Variant 2

On the other hand, McClendon's complexity measure does not occur accurately for mazes with cycles. Figure 5.2(b) and Table 5.7 state that the most complicated mazes are Aldous-Broder, Binary Tree and the easiest Recursive-Backtracker. But the solution times for each solver are the lowest, particularly for Aldous-Broder mazes. It seems that the McClendon complexity measure does not reflect the actual complexity but rather the hallway lengths and their twistiness. So it is tightly connected only to maze size, without taking into consideration the complexity itself. Which was also proven in [**4**], and confirmed by this work for a cyclic mazes.

Shannon's entropy was also measured for cycled and directed mazes, and the results are also very satisfying. The entropy level corresponds firmly with a solution time. The results are even better than for the perfect mazes. Different types of mazes can be even better predicted by this entropy measure. Furthermore, this is the only measure which manifests higher complexity to mazes with added cycles and direction. It s meaningful because it represents, the more complex state of the maze both statistically but also algorithmically.

5.2.3. Parametrizing the maze problem for choosing the best solver

The parameters described so far did not allow for a clear distinction between different types of mazes. Taking into account the data collected in Tables 5.10, 5.11 and Figures 5.4a and 5.4b, we can conclude that the most accurate measure of distinguishing between mazes is the cross number to a dead-end number ratio. Figure 5.4(a) shows how well different types of mazes can be distinguished on this basis. Despite, that the split between Aldous-Broder and Binary Tree mazes with cycles presented in Figure 5.4(b) is not that well separated, the mazes can still be distinguished. The separation classes could be defined by functions as follows: for variant 1 cross ratio < 0.01 and dead-end ratio < 0.14 denotes BackTracker maze, cross ratio $= 0$ denotes Binary tree maze in this configuration, dead-end ratio > 0.25 and cross ratio > 0 denotes Aldous-Broder maze. For variant 2 cross ratio< -6.67·dead-end ratio $+0.8$ denotes Backtracker maze, cross-ratio < -0.48·dead-end ratio $+0.14$ denotes Binary Tree maze, and cross ratio> -0.4·dead-end ratio +0.14 indicates the Aldous-Broder maze. Despite a lot of work dedicated to studying the degree distribution, non of them tries to apply such measures to introduce a framework to distinguish mazes by it. As was already proven the best maze solver for the perfect mazes generated by Binary Tree, Aldous-Broder and Recursive-Backtracker is the Dijkstra algorithm. The best solver for studied mazes with cycles and directed edges was Dijkstra for all maze generators.

5.3. Conclusions

This section presents the conclusions of the presented results and analysis. This thesis was an attempt to create a basic framework which could be helpful for researchers, engineers and developers to have a better understanding of maze problems. Three generating and solving algorithms were tested. Seven maze parameters were evaluated along with two different complexity measures. It was possible to compare obtained data with other works. After the results of analysis and discussion the answers to the research question are submitted:

Q1. What is the relation between the maze features, generated by Binary Tree, Aldous-Broder and Recursive-Backtracker algorithms, and their completion time obtained, when solved using a BFS, Dijkstra and A^* algorithms?
Solution time for perfect mazes with longer subsidiary paths, and cycled-directed mazes with higher fork and intersection ratios, such as Binary Tree, identify to have longer solution time.

Q2. Which maze parameters best describe the complexity of a problem in terms of time completion?
Shanon's Entropy seems to be the most accurate predicate of maze complexity, independent of maze size.

Q3. Which maze features are the best to distinguish different types of mazes?
The best maze feature to distinguish Binary Tree, Aldous-Broder and Recursive-Backtracker is cross number to dead-end number, both for perfect mazes and mazes with added cycles and directions.

6. Web Application Implementation

This chapter describes the web application which was created for this thesis. The main purpose of the application was to present discussed algorithms and the results collected during this study. One of the main assumptions of this application was to make it available to others. The purpose was to easily share this application, therefore it was written as a web frontend application. The application and its repository are publicly available at [**26**, **27**].

6.1. Technologies used

The following sections describe the technological stack used while building the application. It also provides a guide to the user interface. The application was supposed to have a modern look and feel, and be interactive and responsive for its users. The source code [**27**] provides a full implementation of all algorithms, views and scripts needed to correctly run this application. There is no database connection provided because there was no logical reason to create a user authentication or save the results in a database. Each user can export data generated with this application to a PDF file. The project was developed using a free IDE Visual Studio Code.

6.1.1. HTML, CSS and Bootstrap

The main task of this application was to create a presentation layer. Each application view was written in HTML files. HTML is the markup language which gives structure and meaning to web content and is a cornerstone of each web page or application. The second layer of the standard web stack is styling with CSS.
CSS is a language of style rules which apply styling to structured HTML content. As the design of the UI consists of many different components, uniform bootstrap styling was used. Bootstrap [**28**] is a free, open-source front-end development framework for the creation of websites and web apps. It utilizes a prebuilt grid system and components. It also provides dozens of CSS variables. Bootstrap was chosen also because JavaScript in Bootstrap is HTML-first. The implemented application does not contain any convoluted or many dynamic features so it was developed using only JavaScript. Therefore there was no reason to overcomplicate it with JQuery which is a popular JavaScript library which makes HTML document traversal, manipulation and event handling much simpler with an easy-to-use multi-browser API.

6.1.2. JavaScript

The secondary part of this thesis was to implement and compare algorithms. As the web application was arbitrarily chosen to be developed, hence JavaScript was the clear choice as it is third, after HTML and CSS, key layer component of modern web development. All maze-related algorithms were implemented in JavaScript. It may be noticed that from the performance perspective JavaScript should not be the apparent choice. However, the objective of this study was not to maximize the performance, a more low-level language would be better for such a case. JavaScript is a lightweight interpreted programming language. It combines two important features. On the one hand, it allows building object-oriented logic for the implemented algorithms due to some common programming features. On the other hand, it provides tools for dynamically changing the content of web applications [**29**]. Furthermore, the most useful functionality built on top of the client side is a Browser's API, an Application Programming Interface which can expose data from the surrounding computer environment. For this work, it was necessary to use two browser APIs, i.e. DOM and canvas. DOM is a Document Object Model that allows dynamically manipulate with HTML and CSS. Canvas is for creating 2D graphics which was used in the maze generation component to visualise the generated mazes and their solution.

6.1.3. Charts.js

Chart.js [**30**] is a free, open-source JavaScript library used for data visualisation. It allows to easy create different types of dynamic charts. Chart.js renders in HTML5 canvas component. It is a well-known library, which is easy to use. In the application, two scatter plots are implemented with three category classes for each solver. It also provides an animated look for generated charts out of the box.

6.1.4. GitHub Pages

The created app is publicly available through hosting by GitHub Pages [**28**]. GitHub Pages is a static site hosting service for client-side-only applications, it does not support server-side languages. It builds a web page directly from the GitHub Repository of the Project. Service takes HTML, CSS and JavaScript, and runs the files through the Jekyll building process and publishes a website on the github.io domain or custom domain for free. Jekyll is a static site generator with built-in support for GitHub Pages and a simplified build process. Jekyll takes Markdown and HTML files and creates a completely static website. After each push to the repository a GitHub Action is initialised. GitHub Actions is a continuous integration and continuous delivery (CI/CD) platform that allows automating the build, test, and deployment pipeline. However, published GitHub Pages have some restrictions: sites may be no larger than 1 GB, and have a soft bandwidth limit of 100 GB per month. For security purposes, when a GitHub Pages site is visited, the visitor's IP address is logged and stored by GitHub.

6.2. User Interface

The user interface was created using the components included in the Bootstrap and Charts.js libraries, described in subsections 6.1.2 and 6.1.4. The home page is the most important view of the application. It is a dashboard of many components. It contains a form for a maze data generator, data visualisation on canvas components, a table for collected data and a form for choosing the solver.

6.2.1. UI template

In this section, all key features of the user interface are described. The starting page of the application is presented in Figure 6.1. It is a blank template which can be filled with data generated by a user utilising the user interface and implemented algorithms. The template is divided into 5 different parts to provide the optimum user experience. The application allows the user to generate different types of mazes. Besides three base algorithms, the user can add cycles or directions. After maze creation, the user can follow the path and parameters of solvers. All data is visualised in two plots, and all data is collected in a table which may be extracted to the pdf file. The user interface is responsive so it can be also used on mobile devices.

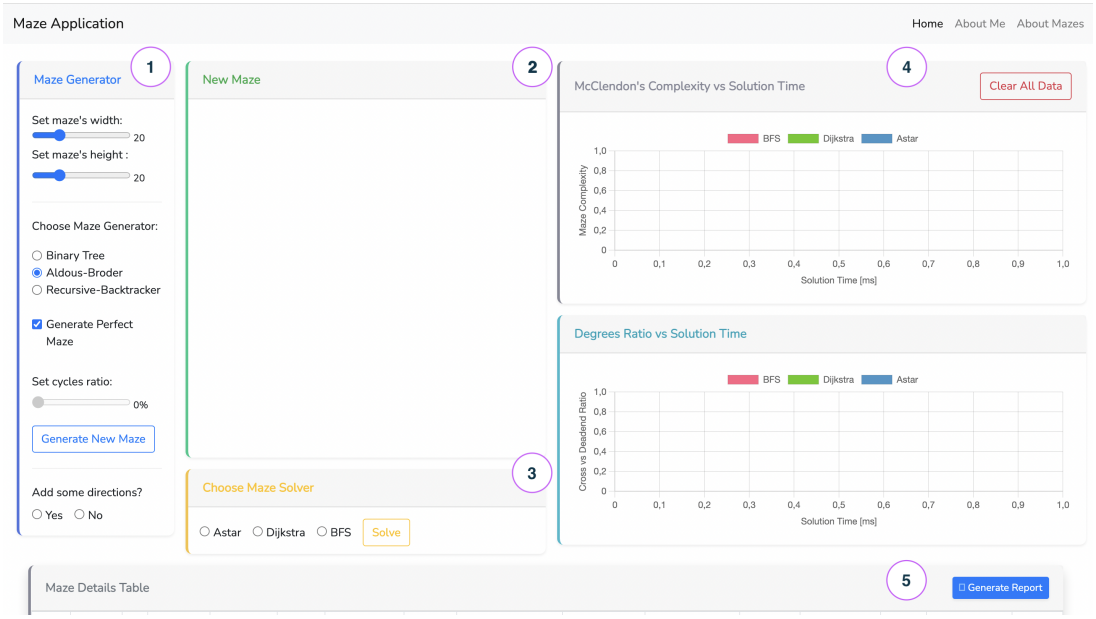


Figure 6.1. Main view of the most important part of the application. The figure presents an empty template before any user actions. All main parts are marked by a number in a pink circle. Maze Generator (1), Maze canvas (2), Solver Section (3), Charts Section (4), Data Table (5).
Source: created by the author.

Maze Generator

This section is a form which allows users to set all parameters before generating a maze. The form component is presented in Figure 6.2. There are a few options to choose. Users can set the rectangular maze size by setting the range. There are three implemented generating algorithms to choose from. Users can also check the Perfect Maze checkbox to generate a perfect maze, or can add some cycles to the maze, by setting the ratio range input. After clicking the Generate Button, the maze is populated into the canvas component.

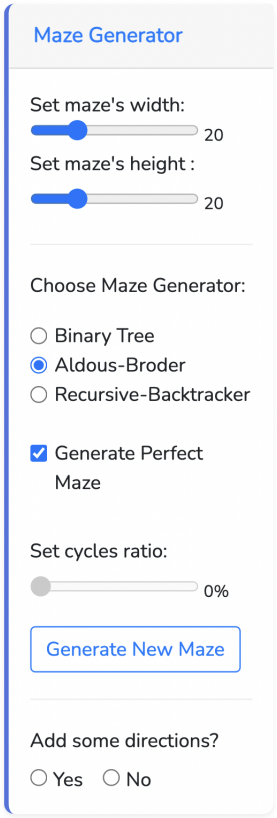


Figure 6.2. Maze Generator Form.
Source: created by the author.

Maze Canvas

When the maze is ready on canvas, the user can select, using radio buttons, if they want to add some directed edges to the maze before solving. If yes, the user can do it by clicking cells in canvas eg. in Figure 6.3. The direction implemented is, so the flow for cells marked green can happen only in one, north direction.

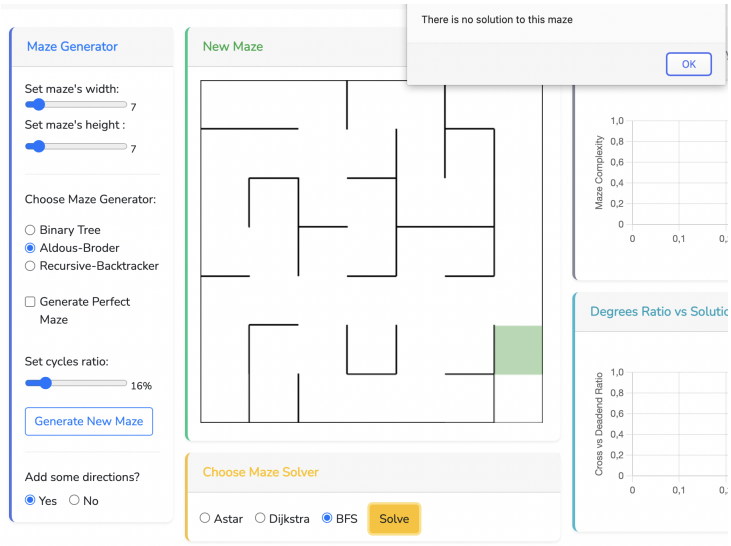


Figure 6.3. Prepared Maze on canvas with a cell chosen to be directed. The maze in the picture can not be solved due to a lack of trespass in the south direction.
Source: created by the author

Solver Section

After setting all maze parameters, and populating the maze on canvas, the user can choose out of three implemented maze solvers by selecting the radio button and clicking on solve button visible in Figure 6.4. One maze can be solved multiple times, by different solvers. Each solution path will be marked in a different colour: Astar - blue, BFS - red, Dijkstra - green.

Figure 6.4. A maze populated on canvas component, solved by three different solvers.
Source: created by the author

Charts Section

Each solution populates a dataset which is drawn into the charts canvas. The chart section contains two separate plots, which collect information about generated mazes and their solutions until the Clear All Data button is pressed. The top plot in Figure 6.5 presents a maze complexity versus solution time in milliseconds. Maze complexity represents a McClendon complexity measure defined in section 4.2.2 by the equation (4.7). Bottom plot in figure 6.5 shows $\frac{cross}{dead-end}$ ratio versus solution time in milliseconds. The ratio of dead-ends and crosses is given by the degree distribution given in section 4.1.1 by Definition 23 and the equation (4.1).

Figure 6.5. Plots generated by the data populated by the user using app maze generator and solvers.
Source: created by the author

Data Table

In the last table section, all generated data are collected in a data table presented in Figure 6.6. After generating and solving each maze the implemented functions calculate maze parameters such as average path length defined by Definition 8, degree distribution given by the equation (4.1) in Definition 23, McClendon's complexity is explained in section 4.2.2 by the equation (4.7), Shannon's entropy is described in section 4.1.2 by the equation (4.3) and time and steps needed to find a solution for each solver. All data is then inserted into a table. Information about the mazes and their solutions is collected until the Clear All Data button is pressed.

Maze Details Table

Generate Report

Maze Id	Generator	Size	Avg. Path Len.	Deadend Ratio	Fork Ratio	Intersection Ratio	Cross Ratio	McClendon's Complexity	Shannon's Entropy	Steps To Solve Dijkstra	Dijkstra Time	Steps To Solve BFS	BFS Time	Steps To Solve Astar	Astar Time
1	Recursive-Backtracker	2500	64.758	0.050	0.574	0.337	0.040	11.968	2902.121	118	1.300	1125	1.800	162	2.600
2	Aldous Broder	2500	62.153	0.149	0.411	0.354	0.086	13.002	2757.127	112	1.500	1011	1.500	151	2.000
3	Binary Tree	2500	55.719	0.134	0.420	0.383	0.063	12.172	2789.113	106	1.500	447	0.600	99	1.200

Figure 6.6. All generated data are stored in the table which may be exported to the PDF file. It gathers all maze and solvers parameters.
Source: created by the author

6.3. Conclusions

This section presents the conclusions of the described web application. The primary idea of the application was to present discussed algorithms and the results collected during this study. All main outlined objectives were realised. Application allows to verify how the described parameters affect the appearance of the maze, its solutions and the required solution time on a dedicated charts, tables and figures. The application is available publicly.

Each application user can:

– generate a maze using the following algorithms: Binary Tree, Aldous-Broder, Recursive-Backtracker,
– add cycles and directed cells to previously generated maze using the UI and interactive canvas,
– solve the maze using the following algorithms: Dijkstra, A^*, BFS,
– observe how maze complexity and cross to dead-end ratio changes for each maze,
– collect and download detailed informations of generated mazes and solutions.

7. Conclusions

This chapter presents the conclusions of the results obtained during this study. This thesis succesfuly created a framework which allows to parametrise three maze solvers: Binary Tree, Aldous-Broder and Recursive-Backtracker, and three maze generators: BFS, Dijkstra and Astar. Seven maze parameters were evaluated along with two different complexity measures. It was possible to compare obtained data with other works. After the results following answers to the research question are submitted:

Q1. What is the relation between the maze features, generated by Binary Tree, Aldous-Broder and Recursive-Backtracker algorithms, and their completion time obtained, when solved using a BFS, Dijkstra and A^* algorithms?
Solution time for perfect mazes with longer subsidiary paths, and cycled-directed mazes with higher fork and intersection ratios, such as Binary Tree, identify to have longer solution time.

Q2. Which maze parameters best describe the complexity of a problem in terms of time completion?
Shanon's Entropy seems to be the most accurate predicate of maze complexity, independent of maze size.

Q3. Which maze features are the best to distinguish different types of mazes?
The best maze feature to distinguish Binary Tree, Aldous-Broder and Recursive-Backtracker is cross number to dead-end number, both for perfect mazes and mazes with added cycles and directions.

For this work a lightweight web apllication in JavaScript was developed to present discussed algorithms and the results collected during this study. All main outlined objectives for this application were realised. Application allows verification of the described parameters affecting the appearance of the maze, its solutions and the required solution time on a dedicated charts, tables and figures. The application and its repository is available publicly.
Each application user can:
− generate a maze using the following algorithms: Binary Tree, Aldous-Broder, Recursive-Backtracker,
− add cycles and directed cells to previously generated maze using the UI and interactive canvas,
− solve the maze using the following algorithms: Dijkstra, A^*, BFS,
− observe how maze complexity and cross to dead-end ratio changes for each maze,
− collect and download detailed informations of generated mazes and solutions.

Bibliography

[1] A. Karlsson, *Evaluation of the Complexity of Procedurally Generated Maze Algorithms*, Bachelor Thesis, Blekinge Institute of Technology, 2018

[2] J. Kwiecień, *A Swarm-Based Approach to Generate Challenging Mazes*, Entropy 20, 762, 2018

[3] X. Liu, D. Gong, *A comparative study of A-star algorithms for search and rescue in perfect maze*, 2011 International Conference on Electric Information and Control Engineering, pp. 24-27, 2011

[4] V. Bellot, M. Cantres, J.M. Favreau, M. Gonzalez-Thauvin, P. Lafourcade, K. Le Cornec, B. Mosnier, S. Rivière-Wekstein, *How to Generate Perfect Mazes?*, Information Sciences, Elsevier, 2021

[5] R. J. Trudeau, *Introduction to Graph Theory*, Dover Publications Inc., New York, 1993

[6] M. Needham, A. E. Hodler,*Graph Algorithms, Practical Examples in Apache Spark and Neo4j*, O'Reilly Media, Inc.,Sebastopol USA, 2019

[7] M. La Rocca, *Advanced Algorithms and Data Structures*, Manning Publications Co., Shelter Islands, 2021

[8] A. A. Jarai, *The Uniform Spanning Tree and related models*, Infinite volume limit of the Abelian sandpile model in dimensions d 3., Probab. Theory Related Fields 141, pp. 181–212., 2009

[9] J. Erickson, *Algorithms*, Independently published, USA, 2019

[10] R. Hofstad, *Random Graphs and complex networks*, Cambridge University Press, The Netherlands, 2017

[11] H. Zenil, N.A. Kiani, J. Tegner, *A Review of Graph and Network Complexity from an Algorithmic Information Perspective*, Entropy 20, 551, 2018

[12] M. Dehmer, F. Emmert-Streib, Z. Chen, X. Li, Y. Shi,*Mathematical Foundations and Applications of Graph Entropy, Vol 6*, Wiley-VCH, 2017

[13] S. Changiz, *Entropy and Graphs*, Master's Thesis, University of Waterloo, Canada, 2013

[14] M.S. McClendon, *The Complexity and Difficulty of a Maze*, Bridges: Mathematical Connections in Art, Music, and Science Confrerence Proceedings 2001 p.213-220

[15] J. Buck, *Mazes for Programmers*, The Pragmatic Programmers LLC., Dallas USA, 2015

[16] T. Cormen, E. Leiserson, R. Rivest, C. Stein, *Introduction to Algorithms 3rd Edition*, The MIT Press, Cambridge, 2009

[17] I. Nunes, G. Iacobelli, D. Ratton Figueiredo, *A transient equivalence between Aldous-Broder and Wilson's algorithms and a two-stage framework for generating uniform spanning trees*, Cornell University, 2022

[18] A.A. Puntambekar, *Analysis and Design of Algorithms*, Technical Publications, Pune, India, 2020

[19] S. Bałdyga, K. Lichy, *Use of a-star algorithm in the design of water vessels*, Zeszyty Naukowe Wyższej Szkoły Informatyki, vol 16, pp. 5-24, 2017

[20] A. Montazeri, I. H. Imran, *Unmanned Aerial Systems:Autonomy, Cognition and Control*,Unmanned Aerial Systems, Elsevier, pp. 47-80, 2021

[21] H. Liu, *Robot Systems for Rail Transit Applications*, Elsevier, Amsterdam, 2020

[22] Autonomous robots, *https://www.starship.xyz*, [last verified access on November 2022]

[23] OSPF protocol, *https://networkencyclopedia.com/open-shortest-path-first-ospf-protocol/*, [last verified access on November 2022]

[24] Nurdin, Bustami, M. Hutomi, M. Elveny, R. Syah *Implementation of the BFS algorithm and web scraping techniques for online shop detection in indoensia* Journal of Theoretical and Applied Information Technology, vol 99, pp. 2878-2899, 2021

[25] A. Candra, M. Andri, R. I. Pohan, 2020 *Application of A-Star Algorithm on Pathfinding Game*, Journal of Physics: Conference Series, vol 1898, 2020

[26] Maze Application, *https://soniaorlikowska.github.io/maze/*, [last verified access on November 2022]

[27] Project Repository, *https://github.com/SoniaOrlikowska/maze*, [last verified access on November 2022]

[28] GitHub Pages Documentation, *https://docs.github.com/en/pages*, [last verified access on November 2022]

[28] JQuery Documentation, *https://api.jquery.com*, [last verified access on November 2022]

[29] Bootstrap Library Documentation, *https://bootstrap-table.com*, [last verified access on November 2022]

[30] JavaScript Documentation, *https://developer.mozilla.org/en-US/docs/Web/JavaScript*, [last verified access on November 2022]

[31] Chart.js Documentation, *https://www.chartjs.org/docs/latest/*, [last verified access on November 2022]

[32] F. Fahleraz *A comparison of BFS, Dijkstra and Astar algorithm for grid-based path- finding in mobile robots*, Institut Teknologi Bandung, Indonesia, 2019

[33] Stackoverflow thread, *https://stackoverflow.com/questions/44648149/a-star-algorithm-vs-dijkstra-algorithm*, [last verified access on November 2022]

[34] Online Statistics textbook, http://www.statsoft.pl/textbook/stathome.html, [last verified access on November 2022]