Contents

List of Symbols

G	graph
V	set of vertices
v	vertex
n	vertex neighbour
E	set of edges
$A(i,j)$	adjacency matrix
w	edge weight
B	binary tree
T	spanning tree
$d(v)$	vertex degree
$P(k)$	degree distribution
n_k	k-degree vertex
$H(M)$	Shannon entropy
p_l	path length
M	maze
h	hallway
W_h	subset of all v in h
S	maze solution
p	maze starting vertex
q	maze goal vertex

1. Work Motivation

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, 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 problem has the same laws as the maze. And so we discover 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, we can state 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 amateurs, specialists, and society. In this chapter, following subjects will be covered: theoretical background of maze-generating algorithms, maze-solving algorithms, other theoretical concepts from the graph theory required to better understand the problems included in this paper. Moreover the concept of determining the difficulty of a maze will be introduced.

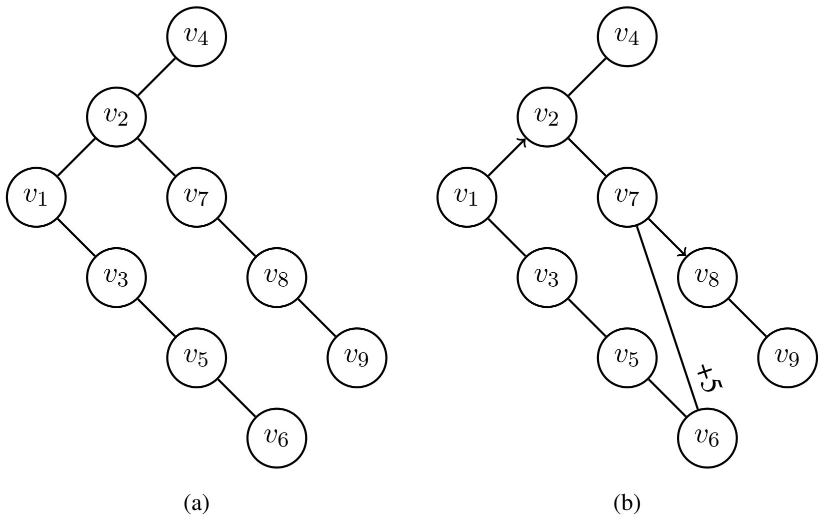
1.1. Graph Theory

In the following section, it will be discussed the graph theory's 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.[**Trudeau**, **2017**]

Definition 2. ***A Graph*** is an object comprising two sets called vertex set and edge set. The vertex set is a finite, nonempty and the edge set may be empty. A graph usually denoted as $G = (V, E)$ is a pair of a V set of nodes (*vertices*), and E set of (*edges*).[**Trudeau**, **2017**]

In this work, three interesting subgroups of graphs will be discussed: weighted graph, directed graph, cyclic graph. By applying *weight* or *direction* to edges of a *undirected*, *unweighted*, *acyclic* graph presented on Figure 1.1(a), we are receiving a *weighted graph* or *directed graph* prezented on Figure 1.1(b). A cyclic graph consists of at least one single *cycle*, which means at least 3 vertices connect in a closed chain Figure 1.1.



Rys. 1.1. In this picture different types of graphs are presented. Graph (a) is an undirected, unweighted, acyclic graph, where *v* denotes a vertex. Graph (b) has two directed edges marked by an arrow and one weighted edge with weight $+5$, and contains one cycle $x_1 \rightarrow x_2 \rightarrow x_7 \rightarrow x_6 \rightarrow x_5 \rightarrow x_3$.

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:

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

Where:
w is an edges weight

Figures 1.2(a) and 1.2(b) are the adjacency matrices of graphs presented on Figure 1.1(a) and 1.1(b) respectivly. The adjacency matrix of a graph requires $\Theta(V^2)$ memory, independent of the number of edges in the graph.

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 5 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

(a) (b)

Rys. 1.2. Examples of different adjacency matrices: adjacency matrix (a) is a matrix of graph presented in Figure 1.1(a), adjacency matrix (b) is a matrix of graph in Figure 1.1(b)

Definition 4. ***Density*** 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:

$$\frac{n(n-1)}{2} \tag{1.1}$$

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

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

$$\frac{n(n+1)}{2} \tag{1.2}$$

Definition 5. ***A Free Tree*** is an undirected, acyclic, connected graph. Let $G = (V, E)$ be an undirected graph. Properties of a tree [**Needham**]
– 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.[**La Rocca**]

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.*[**Jarai**]

Definition 8. ***A Path*** in a graph G is a sequence of vertices $v_1, v_2, \ldots, v_k$. The shortest path is a path with the lowest cost between any two given vertices.[**Erickson**]

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 a shortest path between two vertices contain other shortest paths within it. The shortest path cannot contain any cycles.[**Trudeau**]

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.[**Hofstad**]

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

$$\bar{d} = \frac{density}{n-1} \tag{1.3}$$

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.

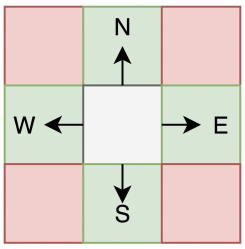
Definition 14. ***A Fork*** is defined as a node with a degree $d(v) = 2$. In the maze it is a cell that is linked to two adjacent nodes.

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 $n \times n$. The grid keeps the information about each cell and their 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 1.3. Diagonal moves are forbidden.

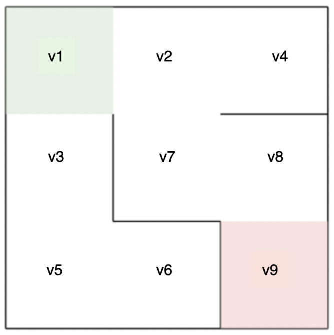


Rys. 1.3. Allowed moves

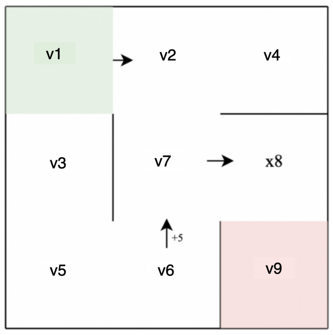
Definition 19. *A* ***Maze*** can be considered as a graph, where each intersection is a vertex, and the path between them is an edge.

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.



(a)



(b)

Rys. 1.4. Examples of different mazes. In subfigure (a) an undirected, unweighted, acyclic maze is shown. In subfigure (b) a maze with a cycle is presented. Maze in subfigure (a) coresponds to graph in Figure 1.1 (a), and maze in subfigure (b) corresponds to graph in Figure 1.1(b). Each cell in maze is considered as a vertex v

Definition 20. *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.[**Buck**]

Definition 21. *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 22. ***A Travelling Salesman Problem (TSP)*** is a problem of finding the shortest path between a given list of nodes in the graph.

1.2. Maze Generation Algorithms

This chapter describes the algorithms that were implemented for this work and were subjected to further comparative analysis. The Binary Tree algorithm [**Cormen**] is the simplest, fast and efficient algorithm for generating a maze. It doesn't require a lot of memory because it only needs to rememebr 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 1.1. Examples of mazes produced by this algorithm are presented in Figure 1.5.

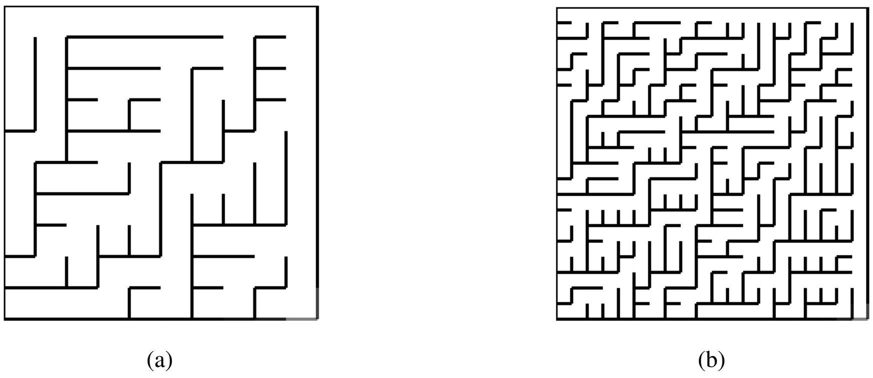
1.2.1. Binary Tree

Listing 1.1. Pseudocode for a Binary Tree Algorithm

```
\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}
```

1.2.2. Aldous-Broder

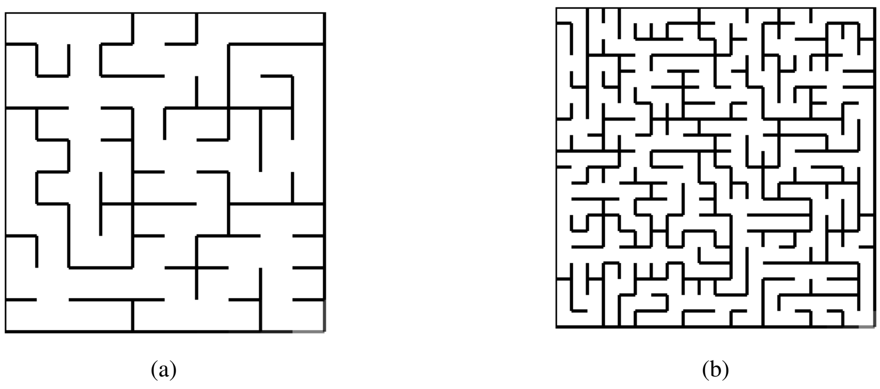
The Aldous-Broder is a well-known algorithm for generating uniform spanning trees (USTs) based on random walks. This means that the maze is perfect and unbiased [**Nunes**]. The algorithm is highly inefficent, but doesn't require a lot of memory. In a given grid, the algorithm randomly chooses any cell, 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 1.2 the pseudocode for an Aldous- Broder algorithm is described.Examples of mazes produced by this algorithm are presented in Figure 1.5


(a) (b)

Rys. 1.5. Examples of different mazes generated by Binary Tree algorithm implemended for this work. In subfigure (a) a maze of size 10×10, and in subfigure (b) of size 20×20

Listing 1.2. Pseudocode for an Aldous-Broder algorithm

```
\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;
\ENDFOREACH
\end{algorithm}
```


(a) (b)

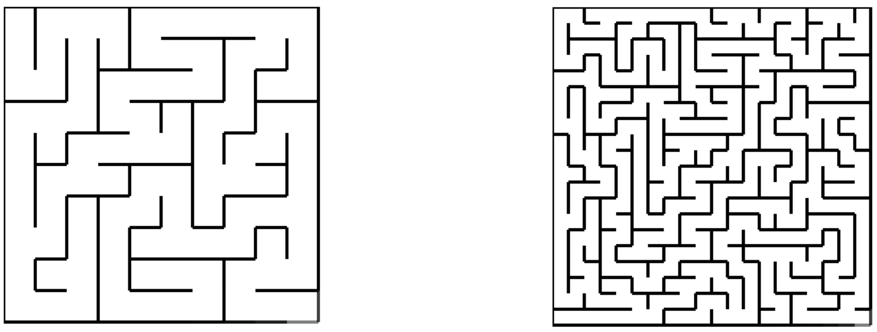
Rys. 1.6. Examples of different mazes generated by Aldous Broder algorithm implemended for this work. In subfigure (a) a maze of size 10×10, and in subfigure (b) of size 20×20

1.2.3. 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[**Puntambekar**]. 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 discover all the vertices that are reachable from the source vertex. The time complexity for the Recursive-Backtracker generator is $O(|V| + |E|)$. In Listing 1.3 the pseudocode for an Recursive Backtracker algorithm is described.Examples of mazes produced by this algorithm are presented in Figure 1.7

Listing 1.3. Pseudocode for a Recursive-Backtracker algorithm

```
\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)

Rys. 1.7. Examples of different mazes generated by Recursive Backtracker algorithm implemended for this work. In subfigure (a) a maze of size 10×10, and in subfigure (b) of size 20×20

1.3. Maze Solving Algorithms

1.3.1. Breadth-First Search Algorithm - BFS

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 [**Cormen**].

1.3.2. Dijkstra Algorithm

Dijkstra is a solving algorithm for single-source shortest-path problems. We can apply it on a weighted, 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 1.4.

Listing 1.4. Pseudocode for a Dijkstra's algorithm

```
\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}
```

1.3.3. 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 are close to the starting point and additionaly 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 1.5 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 1.4.The A^* Algorithm implemented in this work is described in Listing 1.5.

$$f(n) = g(v) + h(v) \tag{1.4}$$

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

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

$$h(v) = |q.x - n.x| + |q.y - n.y| \tag{1.5}$$

Where:
x and y are grid coordinates of vertices

Listing 1.5. Pseudocode for a A* algorithm

```
\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}
```

2. Maze complexity problems

One of the main purpose of this thesis is to discuss 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 complexity measurement methods derived from graph theory will be discussed, next some existing concept of measuring the complexity of a maze. Thinking about a maze and its complexity all different type 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 difficulty of generating a particular maze. This chapter will provide a theoretical background of studying the maze complexity and in chapter 5 I will provide a detailed analysis based on examples.

2.1. Complexity measures in Graph Theory

In this section I will present the approaches derived from graph theory, which describe and measure the complexity of a graph. Conventionally, graph complexity in graph theory is evaluated by a degree distribution, a clustering coefficient, an edge density, a community. Another approach which 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 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 maximum entropy concept is that the more statistically random graph is, the more typical. [**Zenil**]. 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.

2.1.1. Outlining Graph Parameters

Definition 23. ***A Degree Distribution*** is defined as a proportion of vertex with a degree k to all vertices in a graph.

$$P(k) = \frac{n_k}{n} \tag{2.1}$$

Definition 24. ***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

$$\bar{C} = \frac{1}{n}\sum_{n=1}^{n} C_i \tag{2.2}$$

Definition 25. ***A Graph Entropy*** *"Graph entropy represents information-theoretic measures for characterizing networks quantitatively"[***Dehmer***].* It combines graph informations and probabilistic distribution of vertices [**Changiz**]. We can distinguish 3 major fields of graph entropy: The Classical Entropy, The Deterministic Entropy and The Probabilistic Entropy. All three have different application, sometimes specific.

2.1.2. Shannon Entropy

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."[**Zenil**]. 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 (1.3)[**HeZeni**]:

$$H(M) = -\sum_{r=1}^{r} p(x_i)\log_2 p(x_i) \tag{2.3}$$

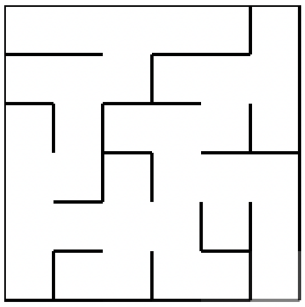
Where:
R is a set of possible outcomes, e.g. All possible adjacency matrix of size m
$p(x_i)$ is a probability of outcome R,
$r = |R|$

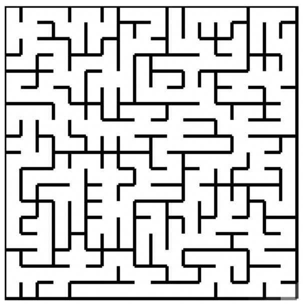
2.2. Maze Complexity Measures

This section, summarize the characteristics which impacts the complexity of a maze. It provides an overview of different approaches and try to compare them.

2.2.1. Independant Maze Parameters

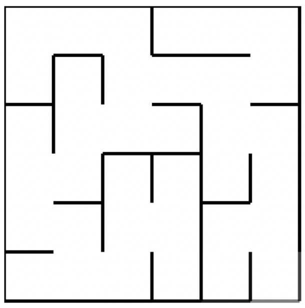
Size is one of the most indisputable complexity factors is of a maze. Maze is described as per Definition 20.In Picture 2.1 two mazes with different size are presented. Both are generated by the author of this work using the Aldous-Broder Algorithm described in Listing 1.2. It is almost too easy to think that a small maze is a simple maze, and a huge maze is a difficult one.
Another key characteristic determining the complexity of a maze is the average length $\bar{p_l}$ of the paths in

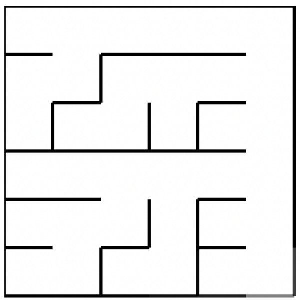

(a) An Aldous-Broder maze size 6×6


(b) An Aldous-Broder maze size 18×18

Rys. 2.1. Examples of different size mazes

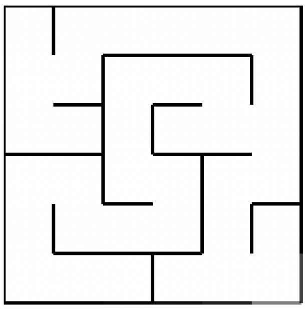
a maze. The longer the path, the bigger the risk of following a faulty road to a solution. Path is given by Definition 8. Its begining is always a start cell and it leads to each dead-end in acyclic mazes. In cyclic mazes, paths can be infinite. In Figure 2.2 two mazes of the same size but different average path lenghts are presented. Mazes in Figure 2.2 are generated by author of this work using algorithms described in Listings 1.1 and 1.2.


(a)

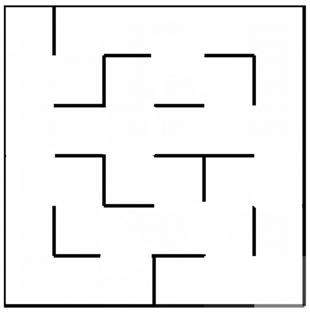

(b) A Binary Tree maze with

Rys. 2.2. Examples of different average path length mazes. In picture (a) it's an example of Aldous-Broder maze with $\bar{p}_l = 9.42$. In picture (b) it's an example of Binary-Tree maze with $\bar{p}_l = 10.8$

Density for an acyclic graph is given by the equation (1.1), and density for cyclic maze is given by the equation (1.2)[**SBorg**]. It describes the ratio between the number of all possible connections and the existing number of connections (edges).In the Picture 2.3 two mazes with different desity are presented. Mazes were generated by the author.



(a)



(b)

Rys. 2.3. Examples of different density mazes. In the Picture (a) maze with density $= 0.40$ is presented, and in Picture (b) maze with densisty $= 0.50$

2.2.2. McClendon Measure

There are few sources indicating a quantitative study of the measure of maze complexity. One of the most cited work in this field is a McClendon **[McClendon]** study of maze difficulty and complexity. McClendon's work treats about maze complexity and difficulty in continuous measure using continuum theory. 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 build 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:

$$\gamma(h) = D(h) \sum_{n=1}^{n} \frac{\theta(w_i)}{d(w_i) \cdot \pi} \tag{2.4}$$

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:

$$\gamma(M) = \log\left[\gamma(T) + \sum_{n=1}^{n} \gamma(B_i)\right] \tag{2.5}$$

Where:
$\gamma(S)$ is a complexity of a solution of the maze,
$\gamma(B_i) = \sum_{n=1}^{n} \gamma(h_i)$ is a complexity of a branch B_i.

Using the equation 2.5 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:

$$\gamma(M) \approx \log\left[\sum_{n=1}^{n} \gamma(h_i)\right] \tag{2.6}$$

Where:
$\gamma(h_i)$ is the complexity of 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:

$$\gamma(M) \approx \log\left[\sum_{n=1}^{n} \frac{h(i)_l \cdot \mathcal{T}}{2}\right] \tag{2.7}$$

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°.

2.2.3. Other approaches to delineate maze complexity

In sections 2.1 and 2.2 different quantitative measures to define maze complexity were disscused. In this section two descriptive methods determining the difficulty of the maze are presented.Baised mazes and time complexity of maze generators may be considered as premises to reduced algorithmic randomnes. This work will try to verify if those premises correlate with time required to solve a maze, or it's McClendon's complexity.

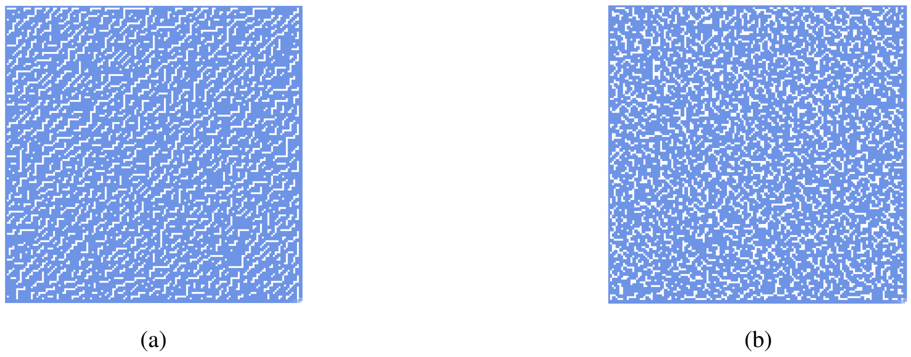
Time Complexity of maze generators
Looking at time complexity of maze generators we can compare them and analyze whether the solution time depends on the time complexity. Below in Table 1.1 time complexity of different maze generator is presented. The analysis of the relation between time complexity and solution time is presented in Chapter 5.

Maze Generator Algorithm Name	Time Complexity
Binary Tree[**Cormen**]	$O(\|V\|)$
Aldous-Broder[**Nunes**]	$O(\|V\|^3)$
Recursive- Backtracker[**\unskip**]	$O(\|V\| + \|E\|)$

Tabela 2.1. Time complexity of maze generating algorithms.

Uniquness and Distinctivness of a Maze

Another approach of describing a maze might be evaluating it uniqueness and distinctivens. 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 as density distribution or average path length and testing how often they appear in a specific configuration. Another method could be to look for biased features. Biased features are some peculiarities of the maze which are visible repeating structures. Good examples of biased maze might be a binary tree algorithm, which tends to create a visible diagonal intersections Figure 2.4(a) , and two perpendicular corridors at the edges of the maze. On the other hand there are mazes generated with Adlous- Broder Algorithm, where no specific bias are visible Figure 2.4(b).



Rys. 2.4. Examples of different tectures in mazes. In the Picture (a) maze generated with Binary Tree Algorithm with visible diagonal tecture, and in Picture (b) maze generated with Adlous-Broder Algorithm without any visible tecture. Mazes in both pictures were generated by the author using the algorithms described in Listings 1.1 and 1.2