

# Contents

<b>1. Background .....</b>	<b>3</b>
1.1. Graph Theory Theoretical Background .....	3
1.2. Maze Generation Algorithms .....	8
1.2.1. Binary Tree .....	8
1.2.2. Aldous-Broder.....	8
1.2.3. Recursive Backtracker .....	9
1.3. Maze Solving Algorithms.....	10
1.3.1. Breadth-First Search Algorithm - BFS .....	10
1.3.2. Dijkstra Algorithm .....	10
1.3.3. A* Algorithm.....	11
<b>2. Maze complexity problems.....</b>	<b>13</b>
2.1. Complexity measures in Graph Theory .....	13
2.1.1. Outlining Graph Parameters .....	13
2.1.2. Shannon Entropy .....	14
2.2. Maze Complexity Measures .....	14
2.2.1. Independant Maze Parameters .....	15
2.2.2. McClendon Measure.....	16
2.2.3. Other approaches to delineate maze complexity .....	17



# 1. Background

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, I provide a 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 paper. I will also present a background for determining the difficulty of a maze.

## 1.1. Graph Theory Theoretical Background

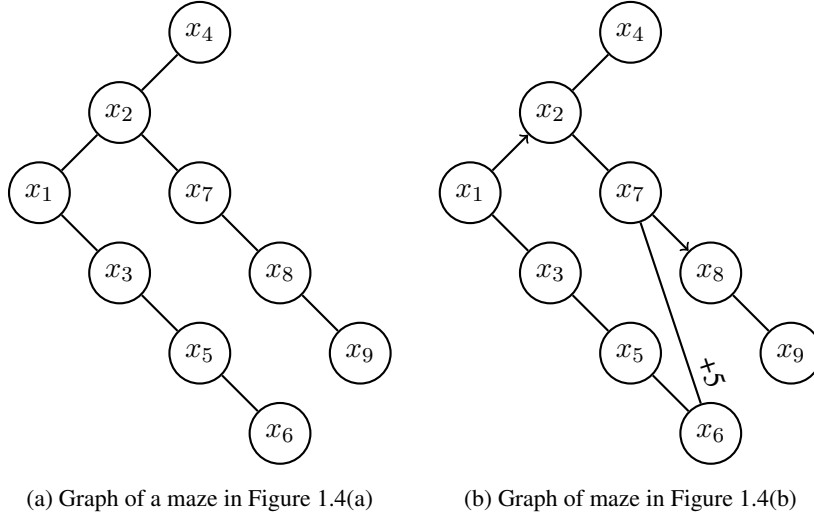
In the following section, I will discuss graph theory's most important mathematical concepts and establish a naming convention to follow in this paper.

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

**Definition 2.** A *Graph* is an object comprising two sets called vertex set and edge set. The vertex set is a finite, nonempty set 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)..

In this paper, I will consider three interesting subgroups of graphs:

- weighted graph,
- directed graph,
- cyclic graph.



**Rys. 1.1.** A different types of graphs

By applying *weight* or *direction* to edges, we are receiving a *weighted graph* or *directed graph*. A cyclic graph consists of at least one single *cycle*, which means at least 3 vertices connect in a closed chain.

**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.  $1, 2, 3, \dots, |V|$ . The representation of a Matrix of consisting  $|V| \times |V|$  such that:

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

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

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

(a) An adjacency matrix of an undirected, unweighted, acyclic maze

(b) An adjacency matrix of a weighted, directed, cyclic maze

**Rys. 1.2.** Examples of different adjacency matrices:

**Definition 4.** A **Graph** in which all vertices are adjacent to all others is said to be complete.

**Definition 5.** **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:

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

**Where:**

$n$  is the number of vertices in a graph.

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

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

**Where:**

$n$  is the number of vertices in a graph.

**Definition 6.** A **Free Tree** is an undirected, acyclic, connected graph. Let  $G = (V, E)$  be an undirected graph. The following properties of a tree apply

- $G$  is a free tree,
- every two nodes 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 7.** A **Binary Tree** is a tree in which each node has no more than two subordinate nodes. Is composed of three disjoint sets of nodes: a root node, a binary tree called its left subtree, and a binary tree called its right subtree.

**Definition 8.** 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.

**Definition 9.** A **Path** in a graph  $G$  is a sequence of nodes  $v_1, v_2, \dots, v_k$ . The shortest path is a path with the lowest cost between any two given nodes.

**Definition 10.** 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. [RTrud]

**Definition 11.** A *Cell* is a single node 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"; 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 12.** A *Degree* of a vertex is denoted as  $d(v)$  and it describes the number of adjacent cells.[ReHofs]

**Definition 13.** An *Average Degree*  $\bar{d}$  for a given graph is given by [ReHofs]:

$$\bar{d} = \frac{\text{density}}{n - 1} \quad (1.3)$$

**Where:**

$n$  is a number of nodes in the graph

**Definition 14.** A *Dead End* is defined as a node with a degree  $d(v) = 1$ . In the maze it's a cell that is linked to only one adjacent node.

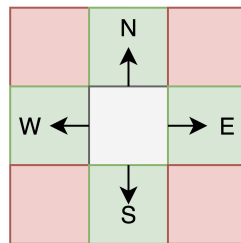
**Definition 15.** A *Fork* is defined as a node with a degree  $d(v) = 2$ . In the maze it's a cell that is linked to two adjacent nodes.

**Definition 16.** An *Intersection* is defined as a node with a degree  $d(v) = 3$ . In the maze it's a cell that is linked to three adjacent nodes.

**Definition 17.** A *Cross* is defined as a node with a degree  $d(v) = 4$ . In the maze it's a cell that is linked to four adjacent nodes.

**Definition 18.** A *Grid* in this paper 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 19.** A *Move* is considered as a transition from one cell to one of its closest neighbours. In this paper, we are using only NSWE moves presented in Figure 1.3. Diagonal moves are forbidden.



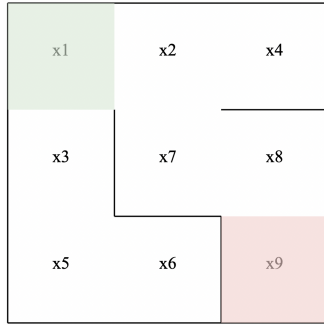
**Rys. 1.3.** Allowed moves

**Definition 20.** A *Maze* can be considered as a graph, where each intersection is a vertex, and the path between them is an edge.

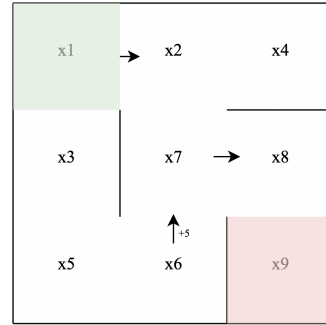
In this paper, I will be considering only 2D mazes on a rectangular grid Figure 1.1. The grid during the implementation process will be treated as a graph Figure 1.1. I will consider a few types of mazes:

- perfect maze,
- directed maze,
- cyclic maze.

The perfect maze will be a maze with only one path between any two given nodes, a directed maze will be a maze with some paths directed in a certain direction, and a cyclic maze will be a maze with at least one cycle.



(a) An undirected, unweighted, acyclic maze



(b) A weighted, directed, cyclic maze

**Rys. 1.4.** Examples of different mazes

If not mentioned otherwise, we assume the start and goal position to always be in the upper-left and lower-right, respectively Figure 1.4.

**Definition 21.** 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.[mazes]

Binary Tree, for example, will always produce mazes with those two unbroken corridors on the north and east.

**Definition 22.** 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 23.** A *Travelling Salesman Problem (TSP)* is a problem of finding the shortest path between a given list of nodes in the graph.

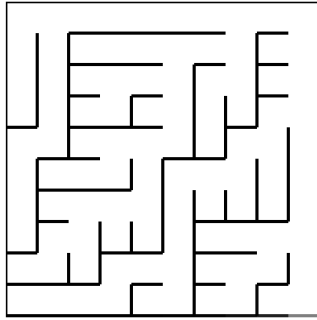
## 1.2. Maze Generation Algorithms

### 1.2.1. Binary Tree

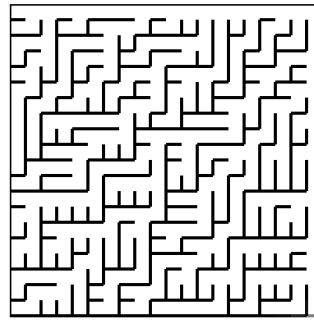
The Binary Tree algorithm is the simplest algorithm for generating a maze. In a given grid, for each cell algorithm decides whether to carve a passage north or east (or any two other directions south/west, south/east etc. ) between two adjacent cells. The algorithm produces a diagonally biased perfect maze which, in other words, is a random binary tree. For building the whole maze, the algorithm doesn't 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  $O(|V|)$ . Below pseudocode for a Binary Tree algorithm.

**Listing 1.1.** Pseudocode for a Binary Tree Algorithm

```
\begin{algorithm}
\FOR EACH 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);
\ENDFOR EACH
\end{algorithm}
```



(a) Binary Tree maze  $10 \times 10$



(b) Binary Tree maze  $20 \times 20$

**Rys. 1.5.** Examples of different binary tree mazes

### 1.2.2. Aldous-Broder

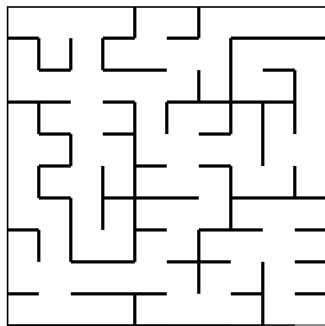
The Aldous-Broder is a well-known algorithm for generating uniform spanning trees (USTs) based on random walks. This means that the maze is perfect and unbiased [INune]. In a given grid, the algorithm randomly chooses any cell, and for this cell randomly chooses a neighbour and if this neighbour wasn't 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  $O(|V|^3)$ . Below pseudocode for an



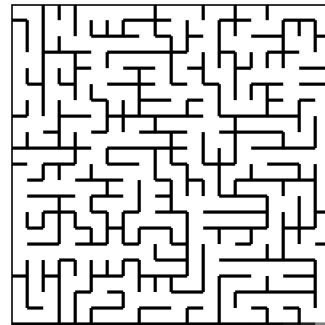
Aldous- Broder algorithm.

**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) Binary Tree maze  $10 \times 10$



(b) Binary Tree maze  $20 \times 20$

**Rys. 1.6.** Examples of different aldous-broder mazes

### 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. 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 we have discovered all the vertices that are reachable from the source vertex. The time complexity for the Recursive-Backtracker generator is  $O(|V| + |E|)$ . Below pseudocode for a Recursive-Backtracker algorithm.

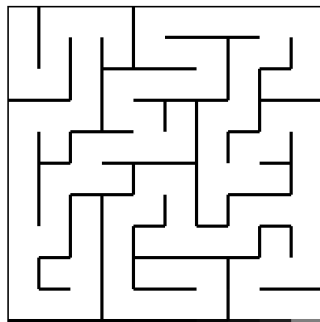
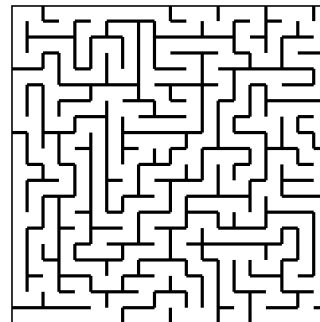
**Listing 1.3.** Pseudocode for a Recursive-Backtracker algorithm

```
\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) Binary Tree maze  $10 \times 10$ (b) Binary Tree maze  $20 \times 20$ **Rys. 1.7.** Examples of different recursive-Backtracker mazes

## 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, we can consider each maze 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  $s$ , BFS explores the edges of  $G$  to „visit“ each vertex directly connected with  $s$ . The algorithm also produces a BFS tree with  $s$  root that contains all reachable vertexes. The shortest path between  $s$  and any vertex  $v$  in  $G$  is a simple path in the BFS tree, that is, a path containing the smallest number of edges [TCorm].

### 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  $s$  have already been determined. As the algorithm floods the graph, we say it uses a greedy strategy.

**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 algorithm. It uses two functions derived from the previously described algorithms. A\* combines the information that Dijkstra's Algorithm uses, meaning choosing the nodes which are close to the starting point and implementing a new type of information which is heuristic meaning choosing nodes which are estimated to be close to the ending point. In the standard terminology used when considering A\*,  $g(n)$  represents the exact cost of the path from the starting point to any vertex  $n$ , and  $h(n)$  describes the heuristic estimated cost from vertex  $n$  to the goal. In each loop, the algorithm minimizes the following function:

$$neighbour.f = neighbour.g + neighbour.h \quad (1.4)$$

**Where:**  $neighbour.g = q.g + \text{distance between } q \text{ and } neighbour$

is a sum of distances from the starting point to the current node, and the distance from the current node to a neighbour.

Maze problems usually have a quick access to basic heuristic functions because of a graph implemented as a grid. In this paper, we will use a heuristic method: a Manhattan Distance. Because we are using only the NSWE moves in discussed mazes, the Manhattan Distance is the best solution. Manhattan Distance from neighbour cell to end cell is given as a:

$$neighbour.h = |end\_cell.x - neighbour.x| + |end\_cell.y - neighbour.y| \quad (1.5)$$

**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 = calculate_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 paper is to discuss complexity of a maze. This chapter will provide a variety of maze and graph complexity definitions. I will start by reviewing complexity measurement methods derived from graph theory. Then I will try to review some existing concept of measuring the complexity of a maze. I will also try to provide an answer to the question of why is it important to define what complexity means and the implication of such a definition. Thinking about a maze and its complexity, we may think about all different features. 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 the below part, 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 degree distribution, clustering coefficient, edge density, 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* or *maxent* 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. [HeZeni]. 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 24.** A *Degree Distribution* is defined as a proportion of vertex with a degree  $k$  to all vertices in a graph.

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

**Definition 25.** A **Clustering Coefficient**  $C_i$  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

$$\bar{C} = \frac{1}{n} \sum_{i=1}^n C_i \quad (2.2)$$

**Definition 26.** A **Community** is a subset of vertices densely connected respectively, and loosely connected to vertices in other communities in the same graph.

**Definition 27.** A **Graph Entropy** “Graph entropy represents information-theoretic measures for characterizing networks quantitatively” [MaDehm]. It is the most important and difficult method to determine graph complexity. There is no “good enough” definition of graph entropy which could be applied to all different kinds of problems. Searching new ways of calculating the entropy of the graph systems is a huge challenge for a scientist from mathematics, physics, biology, chemistry, computer and sociology sciences. 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. As a result, I will not try to make here a general definition of graph entropy, instead in the next subsection I will provide some more details about Shannon’s entropy measure which is one of the simplest method and I will use it later to calculate maze complexity.

### 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.” [HeZeni]. 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 of the object  $M(R, p(x_i))$  is given by (1.3) [HeZeni]:

$$H(M) = - \sum_{i=1}^n p(x_i) \log_2 p(x_i) \quad (2.3)$$

**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  $R$ ,

$n = |R|$

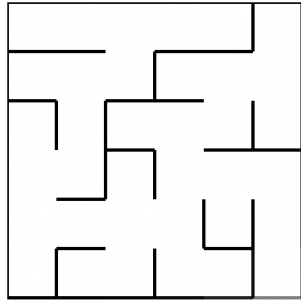
## 2.2. Maze Complexity Measures

In this section, I will discuss the characteristics which impacts the complexity of a maze. I will try to make an overview of different approaches and try to compare them. Since the problem of maze

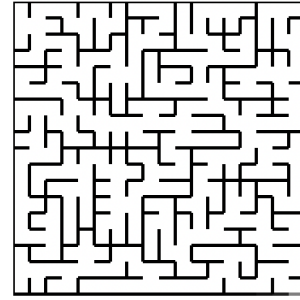
complexity isn't well described in the literature, one of the main aim of this work is to provide a complex description of different measures and approaches while exploring a maze complexity problem.

### 2.2.1. Independant Maze Parameters

**Size** One of the most indisputable complexity factors is the size of a maze. In this work, we use a definition of a maze, which is represented as a square grid, which size is denoted by  $s_m = n \times m$ . It is almost too easy to think that a small maze is a simple maze, and a huge maze is a difficult one.



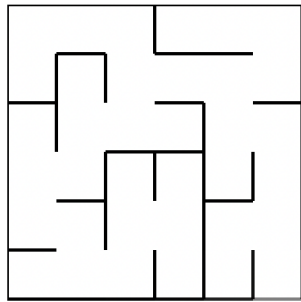
(a) An Aldous-Broder maze size  $6 \times 6$



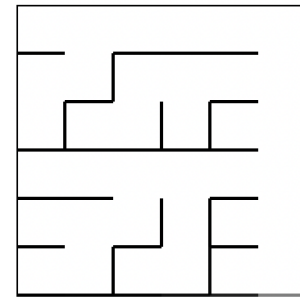
(b) An Aldous-Broder maze size  $18 \times 18$

**Rys. 2.1.** Examples of different size mazes

**Path length** Another key characteristic determining the complexity of a maze is the average length  $\bar{p}_l$  of the paths. The longer the path, the bigger the risk of following a faulty road to a solution. I consider a path in this case as a sequence of moves from the start to each dead-end in acyclic mazes. In cyclic mazes, paths can be infinite.



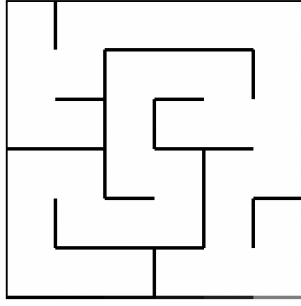
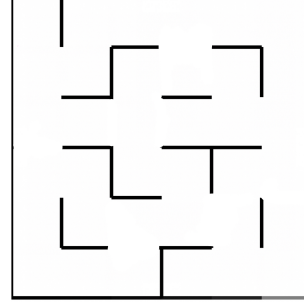
(a) An Aldous-Broder maze with  $\bar{p}_l = 9.42$



(b) A Binary Tree maze with  $\bar{p}_l = 10.8$

**Rys. 2.2.** Examples of different average path length mazes

**Density** Density for an acyclic graph is given by (1.1), and density for cyclic maze is given by (1.2)[SBorg]. It describes the ratio between the number of all possible connections and the existing number of connections ( edges)

(a) A recursive-backtracker maze with  $density = 0.40$ (b) A Binary Tree maze with  $density = 0.50$ **Rys. 2.3.** Examples of different density mazes

### 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 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  $degree = 2$ . A subset  $W_h = w_1, w_2, \dots, w_n$  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  $T$  of a maze. Each branch in  $M$  is connected to  $T$  by a point  $v_i$  in  $I$  which is an intersections set  $I = v_1, v_2, \dots, v_n$ . The McClendon's complexity of a hallway  $h$  is given by:

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

**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  $V(t_{i+1})$

$d(w_i)$  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:

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

**Where:**

$\gamma(T)$  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  $B_i$ .

In the method above to calculate the complexity of a maze, we must know the solution of the maze. To avoid this, we should use the extrinsic approximation of the above method which is given by:



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

**Where:**

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

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

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

**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  $90^\circ$ .

### 2.2.3. Other approaches to delineate maze complexity

In sections above we discussed different quantitative measures to define maze complexity. But we can also list at least two descriptive methods determining the difficulty of the maze.

#### 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.

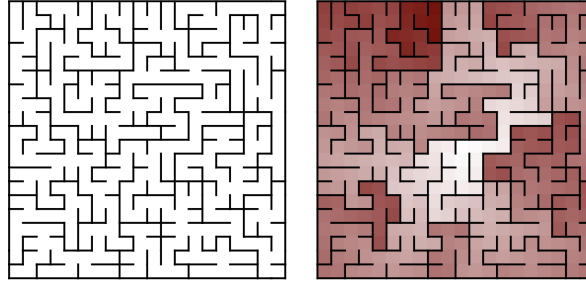
Maze Generator Algorithm Name	Time Complexity
Binary Tree	$O( V )$
Aldous-Broder	$O( V ^3)$
Recursive- Backtracker	$O( V  +  E )$

**Tabela 2.1.** Time complexity of maze generating algorithms.

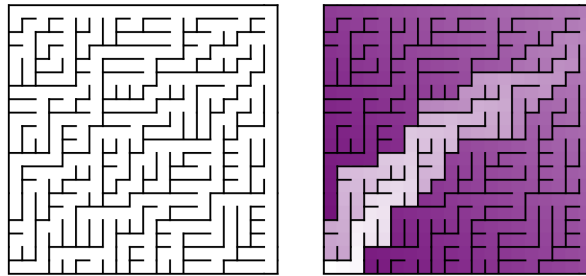
#### Uniqueness and Distinctiveness of a Maze

Another approach of 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 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, and two perpendicular corridors at the edges of the maze. The texture of maze is visible after applying coloring see Figure (1.4). The colors represent the distance from starting point.



(a) An example of non-biased maze generated by Aldous-Broder algorithm. Figure comes from [JaBuck].



(b) An example of biased maze generated by Binary Tree algorithm. Figure comes from [JaBuck].

**Rys. 2.4.** Examples of different maze textures.