

Implementation and Analysis of Maze Generating Algorithms

-Shaikh Ubaid

-180001050

Project Overview

- **Goal**- The main goal is to rank different maze generating algorithms according to the difficulty of the generated mazes.
- Following our main goal we implement and analyse four algorithms.
- For the purpose of evaluating and ranking maze generating algorithms we devise four agents which solve mazes and report the results.
- To assess the level of difficulty of a maze we inspect several features such as number of visited intersections, dead ends, and overall steps of the agents.
- According to agents' performances we rank maze generating algorithms.



Table of Content

1. Introduction
2. Generating Mazes
3. Solving Agents
4. Analysis and Results
5. Discussion
6. Results
7. Proposed Modification
8. Conclusion
9. References



1. Introduction

History

- Mazes are closely related to labyrinths.
- Usually build with naturally occurring materials.
- Had spiritual connotation .
- Later purpose - mainly amusement.
- Wikipedia-In Greek mythology, the Labyrinth was an elaborate, confusing structure designed and built by the legendary artificer Daedalus for King Minos of Crete at Knossos.



Uses

Besides entertainment purposes, mazes are also used in psychology studies of human and animal behaviour to determine space awareness and also intelligence.

Among others, mazes can be used in physics, for example in study of crystal structures.

Project Flow

- We describe and analyse four maze generating algorithms. Our algorithms are originally used in graph theory and we implemented them to generate mazes.
- Mazes containing no loops are known as "simply connected", or "perfect" mazes, and are equivalent to a *tree* in graph theory. Intuitively, if one pulled and stretched out the paths in the maze in the proper way, the result could be made to resemble a tree.
- We generate mazes which are represented as trees/graph only. Our focus is on 2D and planar mazes that do not contain overpasses.
- We devise four different agents that walk across the mazes to help us analyse the difficulty of each maze type. They tell us the number of steps they make from the beginning to the end as well as the number of visited intersections and dead ends.

Project Flow contd...

- Finally, we analyse the relation between properties of mazes and attributes that agents provide us with.
- Final goal is to determine which algorithms are giving us **the most difficult mazes**.
- In the next section we describe four maze generating algorithms. In section 3 we establish the maze solving agents and their properties. In section 4 we analyse mazes and present the results. In section 5 we conclude the project.

2. Generating Mazes

- Data Structure used - square grid graph, grid edges represent walls
- The algorithms convert specific walls into passages.
- After the algorithms do their job, the subgraph made out of passages represent a tree-like structured maze.
- Every maze has properties which we use in the analysis, such as size, number of intersections, number of dead ends, etc.
- Note: All mazes generated and analysed are square.

2.1 Recursive Backtracking

- Firstly, we implement depth-first search (DFS) algorithm also called backtracking . **Main idea of DFS is to go forward as much as possible, then backtrack to the first branch that has unvisited paths and repeat until everything is searched.** We use randomised DFS to obtain random (non-trivial) mazes.
- I have implemented both DFS with explicit stack and without explicit stack(recursion).

Pseudo Code/Steps

1. Choose a starting point in the field.(used the 1st grid in implementation)
 2. Randomly choose a wall at that point and carve a passage through to the adjacent cell, but only if the adjacent cell has not been visited yet. This becomes the new current cell.
 3. If all adjacent cells have been visited, back up to the last cell that has uncarved walls and repeat.
 4. The algorithm ends when the process has backed all the way up to the starting point.
- As this algorithm visits every cell of the grid, its complexity turns out to be $O(n^2)$ where n is the size of the grid.

2.2 Aldous-Broder Algorithm (AB)

- Aldous-Broder **uses random walk** until all vertices are visited. During the walk, suitable connections between the vertices are created (under certain criteria).
- This algorithm is originally used to find uniform spanning tree in the graph.



Pseudo Code/Steps

1. Choose a vertex. Any vertex.
2. Choose a connected neighbour of the vertex and travel to it. If the neighbour has not yet been visited, add the travelled edge to the spanning tree.
3. Repeat step 2 until all vertexes have been visited.

2.3 Wilson's Algorithm (W)

- Majority of maze generation algorithms have biases of various sorts: depth-first search is biased toward long corridors, while Kruskal's/Prim's algorithms are biased toward many short dead ends. Wilson's algorithm, on the other hand, generates an *unbiased* sample from the uniform distribution over all mazes.
- Wilson's algorithm is originally used to find a uniform spanning tree in the graph .

Pseudo code/steps

1. Choose any vertex at random and add it to the UST.(Uniform Spanning Tree)
2. Select any vertex that is not already in the UST and perform a random walk until you encounter a vertex that is in the UST.
3. Add the vertices and edges touched in the random walk to the UST.
4. Repeat 2 and 3 until all vertices have been added to the UST.

2.4 Hunt and Kill (HK)

- Hunt and Kill algorithm **uses the idea of the recursive backtrack** but it starts from a random unvisited vertex whenever hits the dead end.
- There is other version of HK which backtracks to the first unvisited vertex with visited neighbour whenever hits the dead end. (I have implemented this version).
- It does not backtrack to the last vertex with unvisited neighbours.



Pseudo code/steps

1. Choose a starting location.
2. Perform a random walk, carving passages to unvisited neighbors, until the current cell has no unvisited neighbors.
3. Enter “hunt” mode, where you scan the grid looking for an unvisited cell that is adjacent to a visited cell. If found, carve a passage between the two and let the formerly unvisited cell be the new starting location.
4. Repeat steps 2 and 3 until the hunt mode scans the entire grid and finds no unvisited cells.

Again this algorithm visits each cell once, hence time complexity is $O(n^2)$



3. Solving Agents

Agents help us understand how difficult a particular maze is to solve.

Agents produce various attributes of mazes with which we will later analyse the difficulty of the mazes.

These attributes are:

- number of steps needed from the beginning to the end of the maze
- number of visited cells
- number of visited intersections
- number of visited dead ends




3.1 Random Walk (RW) Agent

- The agent walks randomly from a vertex to its random neighbour until it gets to the end of the maze.
- In particular, when located in a node, an agent selects a neighbouring node uniformly at random and moves into it. It repeats this procedure until it finds the end.



3.2 Depth First Search (DFS) Agent

- This agent walks as far as it can until it hits a dead end.
 - The agent then backtracks to the first node with unvisited neighbours.
 - It keeps repeating the walk, until it comes to the end of the maze.
 - The precedence of agent's turns at intersections are manually predefined: left, top, right, bottom
- 

3.3 Heuristic Depth First Search (HDFS) Agent

- Similar to the DFS agent, but selects the preferred directions with a simple heuristic.
- In particular neighbours with lower Manhattan distance to the end of the maze are preferred.
- Manhattan distance - $|x-x_0|+|y-y_0|$



3.4 Breadth First Search (BFS) Agent

- This agent uses the idea of BFS to solve the maze.
- This agent resembles a human solver which can freely jump from one path to another (at least when solving printed mazes).



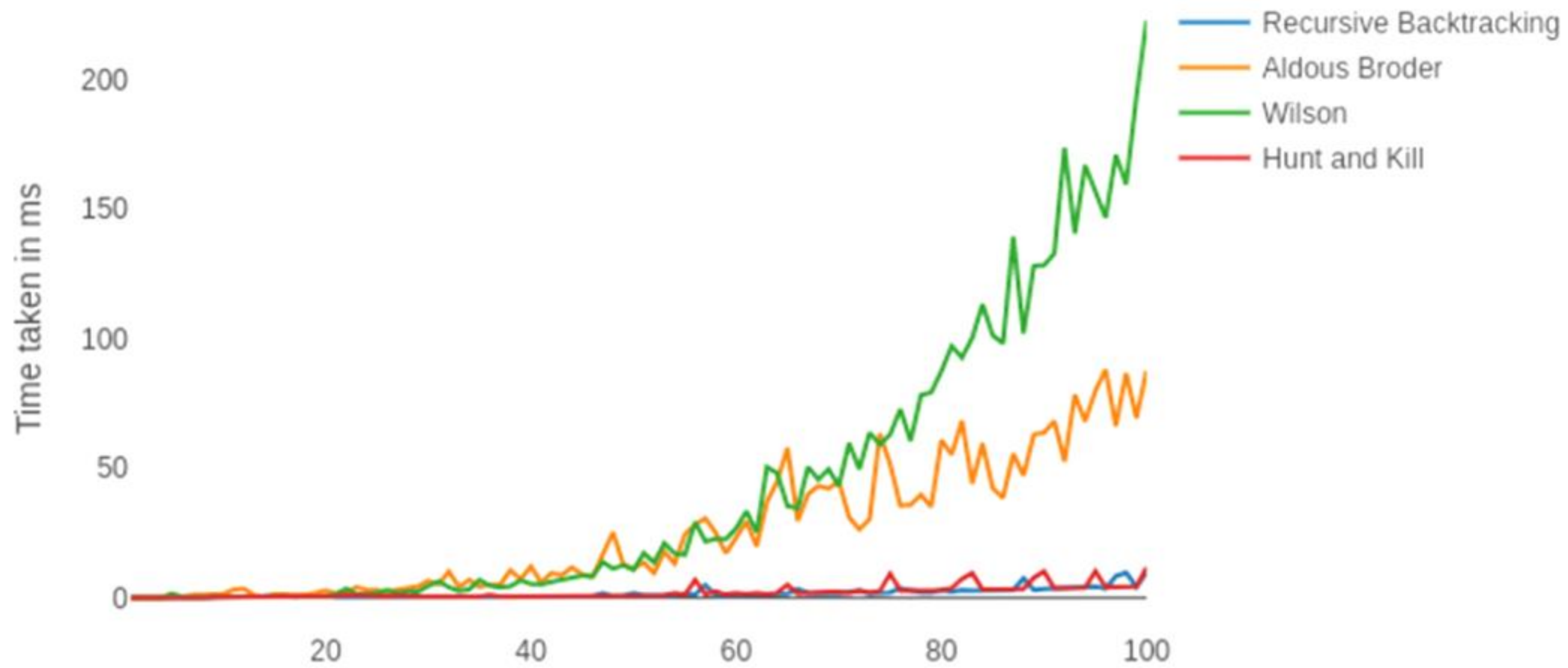
4. Analysis and Results

- In this section we analyse the difficulty of the mazes constructed by the above four generating algorithms.
- We analyse properties of mazes and results of solving agents to determine difficulty of mazes.
- We also experimentally analyse the time performance of algorithms.



4.1 Maze Generating Time

- **We analyse the execution time of maze generating algorithms with respect to the maze size.** We did not bother to consider the number of nodes of the maze which would be a true measure. We used only the length of the side of the square grid graph because we are interested in relations among performances of different algorithms. The result is shown in next slide.
-





Analysis of graph

- Some algorithms stand out performance wise. They are either exceptionally slower or faster than we would expect.
 - Wilson algorithm is very similar to Aldous-Broder with slightly better asymptotic time-complexity in theory. In practice Wilson turns out to be slower because it uses dictionaries unlike the Aldous-Broder, which uses only arrays
 - We have Aldous Broder algorithm which is surprisingly fast. It is simple and uses primitives instead of higher data structures.
 - As both Recursive Backtracking and Hunt and Kill algorithm basically use DFS, their time complexities curves are similar.
-

4.2 Maze Properties

To analyse the difficulty of a maze, we consider the following properties:

- size s
- number of intersections n_i ; intersections are vertices with more than two neighbours
- number of dead ends n_{de} ; dead ends are vertices with only one neighbour.

The bigger n_i the more difficult is the maze. The same goes for dead ends.

Data Set

- I have generated 100 mazes (of size 20×20) from each algorithm, and calculated the number of intersections and dead ends. The average results are shown in Table 1.

	No of Intersections Generated	No of Dead Ends Generated	Score	Rank
AldousBroder	10173	11709	2	1
Wilson	10147	11663	2	1
RecursiveBacktracking	3973	4216	0.753029337	2
HuntAndKill	3728	3938	0.7027827249	3



Table Analysis

- In particular Aldous-Broder and Wilson have practically the same number of intersections and dead ends. They both originate from algorithms for finding uniform spanning tree.
 - Hunt and Kill and Recursive backtrack have notably smaller values of n_i and n_{de} than other mazes. These algorithms originate from DFS.
 - Larger number of intersections and dead ends means more difficult maze, one can deviate from a correct path easily
-

4.3 Agent Performance

Maze solving agents give us another set of maze properties:

- number of steps s that agent needed to get from the beginning to the end
- number of visited intersections (vertices with more than two neighbours)
- number of visited dead ends (vertices with only one neighbour)

Ranking

- We ranked maze generating algorithms according to the performances of the maze solving agents. To rank maze generating algorithms we devised a simple method.
- For every algorithm i we calculated score s_i :

$$s_i = \sum (A_i \div \max(A))$$

- where A_i represents value of score of an agent A for algorithm i , and $\max(A)$ represents the maximum value that the agent scored among all generating algorithms.
- According to the score s_i we ranked generating algorithms. Algorithm with the biggest (best) score has rank 1, etc

4.3.1 Number of Steps

- The basic measure is the number of steps an agent makes from the start to the end. A step is defined as a transition from a node to the adjacent node.

	DFS	BFS	HeuristicsDFS	RandomWalk	Score	Rank
RecursiveBacktracking	33940	24038	16199	4819386	3.693617267	1
Wilson	29585	33568	13346	1715718	3.03706382	2
AldousBroder	29654	34062	12055	1994888	3.886127007	3
HuntAndKill	17055	34656	11427	1976270	2.617985094	4

4.3.2 Visited Intersections

- Next, we analyse how many intersections agents visit. The more intersections that an agent visits the better chance to miss the right path. Hence, the generating algorithm is more difficult.

	DFS	BFS	HeuristicsDFS	RandomWalk	Score	Rank
AldousBroder	8013	8899	3837	795273	3.940902403	1
Wilson	7981	8741	4078	692631	3.849186572	2
RecursiveBacktracking	3329	2134	1374	602349	3.515462844	3
HuntAndKill	2011	3440	1535	271467	1.355288177	4

4.3.3 Visited Dead Ends

- The last property that we analyse is the number of dead ends that agents visit on average.

	DFS	BFS	HeuristicsDFS	RandomWalk	Score	Rank
AldousBroder	8187	9718	2718	294181	3.892024943	1
Wilson	8059	9557	3047	253293	3.828809001	2
RecursiveBacktracking	3188	2120	627	219894	3.367494267	3
HuntAndKill	1213	3343	855	97047	1.102655158	4



5. Discussion

- Our goal was to rank the maze generating algorithms from those that generate the most difficult mazes to those that generate the least difficult mazes.
 - We did that with the help of several criteria: maze properties and solving agents performances.
-

6. Results

- We ranked the algorithms according to the criteria.
- The final ranking of difficulty level: For every measure we ranked algorithms. Finally we calculated average of all the ranks which gives us the final order

	Steps taken by different agents	Intersections visited by different agents	Dead Ends visited by different agents	Total Intersections and Dead Ends Generated	Average	Rank
AldousBroder	3	1	1	1	1.5	1
Wilson	2	2	2	1	1.75	2
RecursiveBacktracking	1	3	3	2	2.25	3
HuntAndKill	4	4	4	3	3.75	4



Results contd...

- Having established the ranking of the algorithms we can now find the properties that distinguish among the various levels of difficulty of the algorithms.
- The number of intersections is correlated to the difficulty of mazes. More intersections means that the maze is more difficult, and that there is more chance to miss the correct path.

Results contd...

The type of the algorithm contributes to the level of the difficulty:

1. Best results are achieved by AldousBroder and Wilson. They originate from algorithms for finding uniform spanning trees in graphs. We speculate that agents have difficulty navigating through the maze because the paths are unbiased in any direction.
2. The worst performing pair is Recursive Backtracking, and Hunt and Kill algorithms. They originate from the graph search algorithms. On the other hand most solving agents use the same approach which enables them to solve the maze easily. Therefore the mazes are generated in a way that suit the solving agents.

7.

Proposed Modification

- One issue with Wilson algorithm is maintaining list of unvisited vertices so as to pick random vertex from it.
- As soon as vertex is picked, we need to remove it from the list.
- So to support insertion, removal, searching as well as random picking we can use a combination of array and hash map.



Proposed Modification contd...

- The hash map stores array values as keys and array indexes as values.
 - Insert will insert at end and add entry into hash map.
 - Searching will be done in const time as we can just look up the hash table to find the given element.
 - For deletion, we first get position of element to be deleted(can be done in $O(1)$ time using hash table), then swap this element with last element and update the location of last element in the hash table.
 - For random element, generate a random number from 0 to last index. And return the array element at the randomly generated index.
-



8. Conclusion

- we studied and analysed two different approaches of generating mazes and were able to rank them by levels of difficulty. Nevertheless both considered types were somehow related to each other since they are used for finding trees in graphs.
 - In the future it would be useful to take into the consideration algorithms with completely different approach and then compare the results
-



9. References

1. Analysis of Maze Generating Algorithms Gabrovšek, Peter, Link to [paper](#).
 2. Maze Generation Algorithm, [Wikipedia](#)
 3. Web Blog, by Jamis Buck, [Link](#)
-

Thank You

