JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue

JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)

An International Scholarly Open Access, Peer-reviewed, Refereed Journal

Maze Theory

Pathfinding in a maze using A search algorithm*

Shaikh Mohd Ashfaque, Shaikh Hammad Sikandar Bakht, Khan Imran Amirullah

mhdashfaque@eng.rizvi.edu.in, shaikhhammad@eng.rizvi.edu.in, imran.khan@eng.rizvi.edu.in
B.E Computer Engineering,
Rizvi College of Engineering, Mumbai, India.

Abstract: A Maze is a 2-dimensional representation of intricate walls and passages. We can consider the smallest unit of a maze as a cell. Walls and passages are combination of cells. A passage is an empty space through a maze whereas, a wall is a closed passage (block) in a maze which is inaccessible. Mazes can be of different types like Perfect Maze - which consists of one solution, where Braid and Spiral Mazes have multiple solutions. A Perfect Maze does not have any loops or inaccessible areas just a connection of neighboring cells. In a Perfect Maze there is only one entry point and exit point. That means it has exactly one solution. The primary go to algorithms for generating a perfect maze are Kruskal's algorithm or Prim's algorithm. They both use a Minimum Spanning Tree over the set of cells. Solving a maze manually is a whole different scenario, it requires tremendous amount of time and effort, also as the mazes get more complex it becomes more tedious. There are many computer algorithms used to solve a maze like A* Search Algorithm, Breadth First Search (BFS), Depth First Search (DFS), Dijkstra's Algorithm. However A* is the best for solving a perfect maze due it two major advantages over other algorithms:

- ✓ Good Heuristic function resulting in lower time complexity.
- ✓ Less memory requirement.

Index Terms – **Maze Generation, Prim's Algorithm, Maze Solving, A* Search Algorithm.**

1. PROBLEM STATEMENT:

Pathfinding algorithms find a path from source to destination avoiding obstacles and minimizing the costs (time, distance, risks, fuel, price, etc.). They are mainly used in navigation like maps and in strategy games (with the use of artificial intelligence) and also in robots for tasks like cleaning, placing objects, picking trash, etc.

In this project, we will first generate a random maze of a specified size using Prim's Algorithm. One entry point and exit point are integrated into the maze. Since the maze is perfect it will have only possible solution. Then, for each case, using A* Search Algorithm we will find the solution of the given maze in the least possible amount of time.

2. LITERATURE REVIEW:

- ➢ Study of procedurally generated maze algorithms: Albin Karlsson (March 2018)

This study has attempted to provide level designers, developers, and game designers with information to evaluate and understand what makes some mazes more complex than others.

To do this a software implementation was developed to perform tests on three different maze generation algorithms across five different maze resolutions, which gathered their solving path length, traversed nodes, corridor distribution, branch distribution, and completion time data. This project aims to investigate the difference in complexity between the maze generation algorithms recursive backtracker (RecBack), Prim's algorithm (Prims), and recursive division (RecDiv), in terms completion time, when solved using a depth-first-search (DFS) algorithm. And in order to understand which parameters affect completion time/complexity, investigate possible connections between completion time, and the distribution of branching paths, distribution of corridors, and length of the path traversed by DFS.

Prims algorithm had the lowest complexity, RecDiv intermediate complexity, and RecBack the highest complexity. Increased solving path length, traversed nodes, and increased proportions of 2-branches, seem to correlate with increased complexity. However the corridor distribution results are too small and diverse to identify a pattern affecting completion time.

➢ Analysis of Maze Generating Algorithms: Gabrovˇsek & Peter (January 2019)

In this paper the main goal was to rank different maze generating algorithms according to the difficulty of the generate mazes. Following this goal the author implemented and analyzed six 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. The best performing algorithms are derived from algorithms for finding uniform spanning trees in graphs.

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

- Best results are achieved by Aldous-Broder 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.
- Next pair by ranking are Kruskal and Prim. They originate from algorithms for finding minimum spanning trees in graphs. In comparison to the first group the paths here are not so evenly distributed which makes the mazes less difficult.
- 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.

3. COMPARISON OF MAZE GENERATION ALGORITHMS:

Algorithm	Dead End %	Type	Focus	Bias Free?	Uniform?	Memory	Time	Solution %	Source
Unicursal	0	Tree	Wall	Yes	never	N^2	379	100.0	Walter Pullen
Hunt and Kill	11	Tree	Passage	Yes	never	0	100	9.5	Walter Pullen
Wilson's Algorithm	29	Tree	Either	Yes	Yes	N^2	48	4.5	David Wilson
Aldous-Broder Algorithm	29	Tree	Either	Yes	Yes	0	279	4.5	David Aldous & Andrei Broder
Kruskal's Algorithm	30	Set	Either	Yes	no	N^2	33	4.1	Joseph Kruskal
Prim's Algorithm (modified)	36	Tree	Either	Yes	no	N^2	30	2.3	Robert Prim

The above table compares various Maze Generation Algorithms on different criterion as follows:

- **Dead End:** This is the approximate percentage of cells that are dead ends in a Maze created with this algorithm, when applied to an orthogonal 2D Maze.
- **Type:** There are two types of perfect Maze creation algorithms: A tree based algorithm grows the Maze like a tree, always adding onto what is already present, having a valid perfect Maze at every step. A set based algorithm builds where it pleases, keeping track of which parts of the Maze are connected with each other.
- **Bias Free:** This is whether the algorithm treats all directions and sides of the Maze equally.
- **Uniform:** This is whether the algorithm generates all possible Mazes with equal probability. "Yes" means the algorithm is fully uniform. "No" means the algorithm can potentially generate all possible Mazes within whatever space, but not with equal probability. "Never" means there exists possible Mazes that the algorithm can't ever generate.
- **Memory:** This is how much extra memory or stack is required to implement the algorithm (that will be proportional to the no. of cells).
- **Time:** This gives an idea of how long it takes to create a Maze using this algorithm, lower numbers being faster.

- **Solution:** This is the percentage of cells in the Maze that the solution path passes through, for a typical Maze created by the algorithm.

- **Source:** This is the source of the algorithm, which means who it's named after, or who invented or popularized it.

From the above table we can conclude that Prim's algorithm is better in most cases.

4. COMPARISON OF MAZE SOLVING ALGORITHMS:

➢ **Breadth First Search v/s Depth First Search v/s A* Search Algorithm:**

- A* uses the least amount of memory among the three.
- For a good enough heuristic, A* is faster than the other two.
- If the maze (tree) is dense, A* outperforms BFS and DFS.

5. METHODOLOGY:

Graph

The graph is one of the pillars of graph theory. A graph consists of a vertex set, an edge set, and a mapping function called relation of incidence. This function states that each edge has two vertices. Another graph-attribute is connectivity which is used to distinguish between connected and disconnected graphs. In a connected graph all vertices are connected, which means that there is a path.

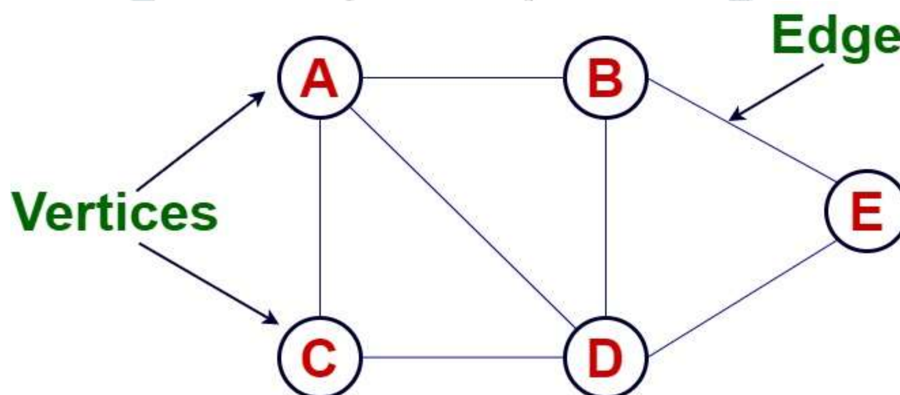

Figure 1: Example of a Graph.

Spanning Tree

A tree in graph theory is a connected graph with no cycles, a cycle being defined as a path that starts and ends at the same vertex and includes at least one edge. A tree is considered a spanning tree when it includes all the graph vertices (spans across all vertices).

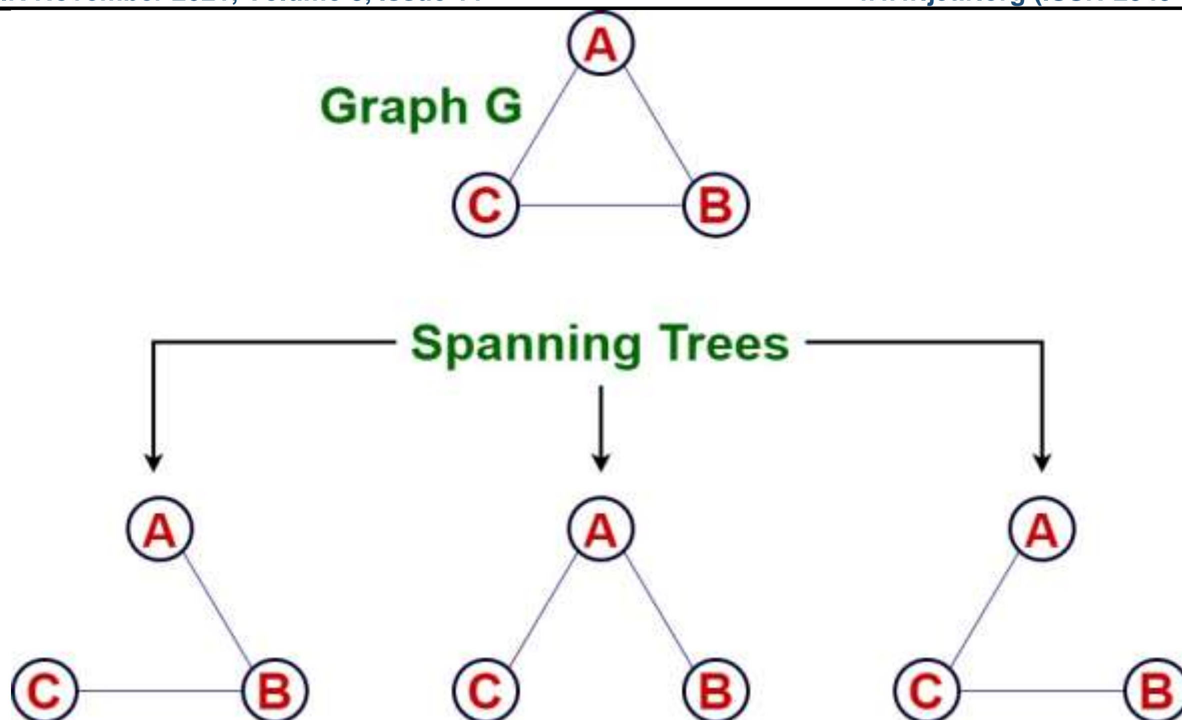

Figure 2: Forming a Spanning Tree from a given Graph.

Weighted Graph

A weighted graph has a weight associated with each edge. A common use of weight is having it represent the length of the edge. For instance classical Prim's algorithm searches for the minimum weight spanning tree in a weighed graph where weight represents edge length.

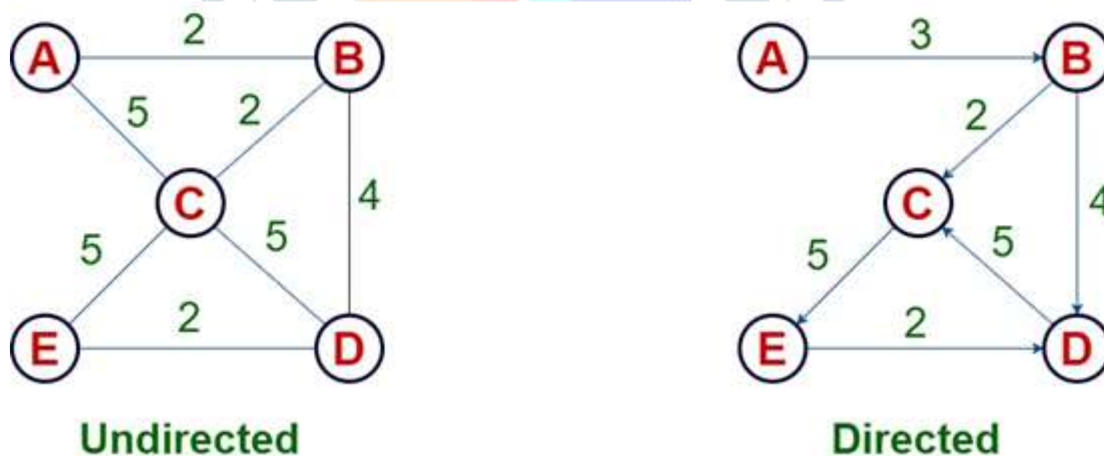

Figure 3: Examples of Weighted Undirected & Weighted Directed Graphs.

Based on types of passages, mazes can be classified into 4 different categories:

- **Perfect:** A "perfect" Maze means one without any loops or closed circuits, and without any inaccessible areas. Also called a simply-connected Maze. From each point, there is exactly one path to any other point. The Maze has exactly one solution.

- **Braid:** A "braid" Maze means one without any dead ends. Also called a purely multiply connected Maze. Such a Maze uses passages that coil around and run back into each other (hence the term "braid") and cause you to spend time going in circles instead of bumping into dead ends. A well-designed braid Maze can be much harder than a perfect Maze of the same size.

- **Unicursal:** A unicursal Maze means one without any junctions. Sometimes the term Labyrinth is used to refer to constructs of this type, while "Maze" means a puzzle where choices are involved. A unicursal Maze has just one long snake-like passage that coils throughout the extent of the Maze.
- **Sparseness:** Sparse Maze is one that doesn't carve passages through every cell, meaning some are left uncreated. This amounts to having inaccessible locations, making this somewhat the reverse of a braid Maze.

6. ALGORITHMS IMPLEMENTED:

Maze Generation Algorithm:

✓ Prim's Algorithm:

Prim's Algorithm unlike traditional approaches starts at a random point in the given graph and then grows outwards from that point. Rather than working edgewise across the entire graph, Prim's Algorithm starts at one point, and grows outward from that point. The algorithm in action works like this:

Algorithm:

1. Choose an arbitrary vertex from G (the graph), and add it to some (initially empty) set V.
2. Choose the edge with the smallest weight from G, that connects a vertex in V with another vertex not in V.
3. Add that edge to the minimal spanning tree, and the edge's other vertex to V.
4. Repeat steps 2 and 3 until V includes every vertex in G.

And the result is a minimal spanning tree of G.

Note that simply running classical Prim's on a graph with random edge weights would create mazes stylistically identical to Kruskal's, because they are both minimal spanning tree algorithms. Instead, this algorithm introduces stylistic variation because the edges closer to the starting point have a lower effective weight.

Modified Version

Although the classical Prim's algorithm keeps a list of edges, for maze generation we could instead maintain a list of adjacent cells. If the randomly chosen cell has multiple edges that connect it to the existing maze, select one of these edges at random. This will tend to branch slightly more than the edge-based version above.

Simplified Version

The algorithm can be simplified even further by randomly selecting cells that neighbor already-visited cells, rather than keeping track of the weights of all cells or edges. It will usually be relatively easy to find the way to the starting cell, but hard to find the way anywhere else.

Randomized Prim's Algorithm

Prim's algorithm is a greedy algorithm that finds a minimum-length spanning tree in a weighted graph. The randomized version of Prim's algorithm does not need a weighted graph, since instead of it choosing the node with least weight, it is made to instead choose a random node.

Maze Solving Algorithm:

✓ A* Search Algorithm:

A * algorithm is a searching algorithm that searches for the shortest path between the initial and the final points of the maze in least possible amount of time.
Imagine a square maze which possesses many walls, scattered randomly. The initial and the final cell is provided. The aim is to reach the final cell in the shortest amount of time.

A* algorithm has 3 parameters:

g: the cost of moving from the initial cell to the current cell. Basically, it is the sum of all the cells that have been visited since leaving the first cell.

h: also known as the heuristic value, it is the estimated cost of moving from the current cell to the final cell. The actual cost cannot be calculated until the final cell is reached. Hence, h is the estimated cost. We must make sure that there is never an over estimation of the cost.

f: it is the sum of g and h. So, $f = g + h$
The way that the algorithm makes its decisions is by taking the f-value into account. The algorithm selects the smallest f-valued cell and moves to that cell. This process continues until the algorithm reaches its goal cell.

Algorithm:

1. Initialize list1
2. Initialize list2 and
   put the starting node in list1

```
3.  while list1 is not empty
    a) find the node with the least f on
        list1, call it "q"

    b) pop q off list1

    c) generate q's 8 successors and set their
       parents to q

    d) for each successor
        i) if successor is the goal, stop search
          successor.g = q.g + distance between
                              successor and q
          successor.h = distance from goal to
          successor (Manhattan Distance for heuristics)

          successor.f = successor.g + successor.h

        ii) if a node with the same position as
            successor is in list1 which has a
           lower f than successor, skip this successor

        iii) if a node with the same position as
            successor  is in list2 which has
            a lower f than successor, skip this successor
            otherwise, add  the node to list1
     end (for loop)

    e) push q in list2
    end (while loop)
```

There are many ways to find h but we'll consider **Manhattan Distance** to calculate it, since we're only considering four directions in a maze (top, bottom, left and right),

It is nothing but the sum of absolute values of differences in the goal's (in **Green**) x and y coordinates and the current cell's (in **Blue**) x and y coordinates respectively, i.e.

```
h = abs (current_cell.x – goal.x) +
     abs (current_cell.y – goal.y)
```

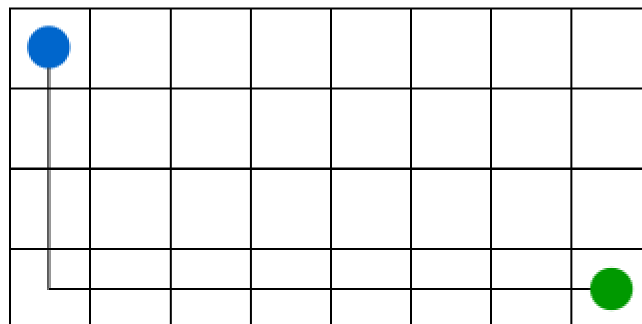

Figure 4: Example of Manhattan Distance calculation.

If we want to reach the target cell from the source cell, then the A* Search algorithm would follow path as shown below. Note that the below figure is made by considering **Manhattan Distance** as heuristics.

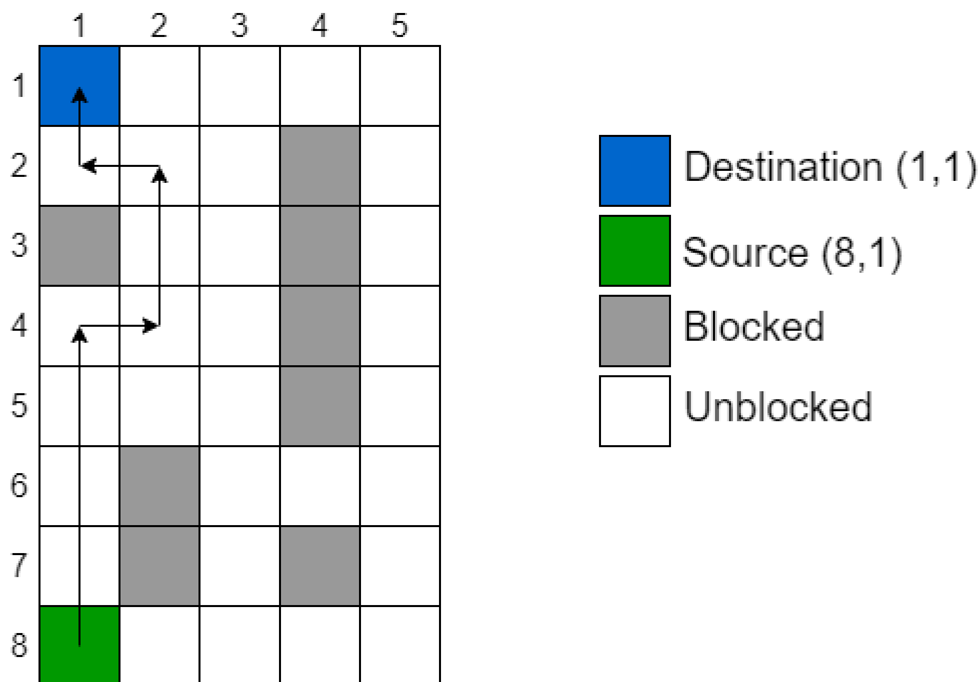

Figure 5: Example of how A* Search Algorithm carves a solution in a Maze.

In **Figure 5**, we can see that A* algorithm makes an **intelligent** choice to go from cell (8,1) to (7,1) instead of going to this path (8,1) to (8,2) to (8,3) to (7,3) to (6,3) and so on.

Also, from cell (2,2) we go to (2,1) then to (1,1) which is goal cell, we can also go from (2,2) to (1,2) then to (1,1) (goal cell) since the **Manhattan Distance** in both cases or paths is (2-1) + (2-1) = 2 which is same.

Complexity:

Considering a graph, it may take us to travel all the edge to reach the destination cell from the source cell the worst case time complexity is O (E), where E is the number of edges in the graph.

Auxiliary Space in the worst case we can have all the edges inside the first list, so required auxiliary space in worst case is O(V), where V is the total number of vertices.

7. CONCLUSION:

The goal of the project was to see how A* search algorithm can be used to solve mazes of varying dimensions. The initial maze was generated using Prim's algorithm due to its low time complexity. For the solving part, in A* algorithm we used **Manhattan Distance** heuristics to approximate the value of 'h'. This can be applied to more complex and intricate mazes such braid, spiral & unicursal and can also be utilized for navigation purposes.

8. REFERENCES:

[1] "Introduction to Algorithms" by Thomas H. Cormen, Charles E. Leiserson, Ronald Rivest, Clifford Stein, 3rd Edition, September 2009.

[2] "Algorithms" (4th Edition) by Robert Sedgewick & Kevin Wayne, April 2011.

[3] "Mazes for Programmers: Code Your Own Twisty Little Passages" by Jamis Buck, July 2015.

[4] "Study of procedurally generated maze algorithms" by Albin Karlsson, March 2018.

[5] "Analysis of Maze Generating Algorithms" by Gabrovˇsek & Peter, January 2019.