A Comprehensive and Comparative Study of Maze-Solving Techniques by Implementing Graph Theory

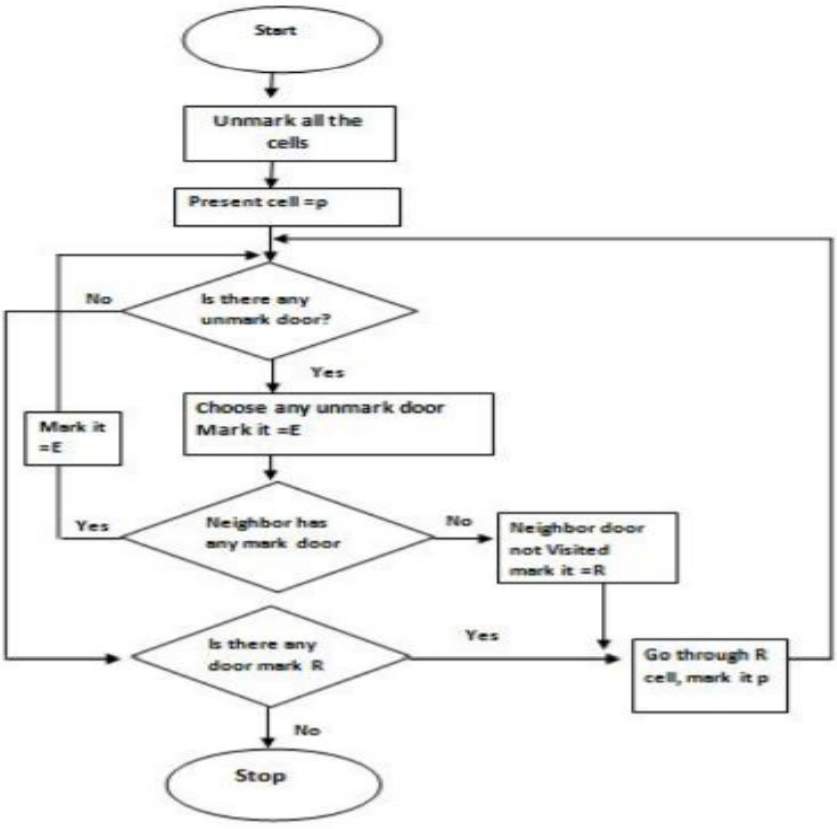
Link : https://ieeexplore.ieee.org/document/5656597/

Authors : Adil M. J. Sadik, Maruf A. Dhali, Hasib M. A. B. Farid, Tafhim U Rashid, A. Syeed

Graph theory appears as an efficient tool while designing proficient maze solving techniques. Graph is a representation or collection of sets of nodes and edges and graph theory is the mathematical structure used to model pair wise relations between these nodes within the graph. By proper interpretation and mathematical modeling, it is possible to figure out the shortest distance between any of the two nodes. This concept is deployed in solving unknown maze consisting of multiple cells. Depending on the number of cells, maze dimension may be 8x8, 16x16 or 32x32. Each cell can be considered as a node which is isolated by walls or edges. This is nothing but the interpretation of graphs onto a maze. This analogy between graph and maze provides the necessary foundation in developing maze solving algorithm

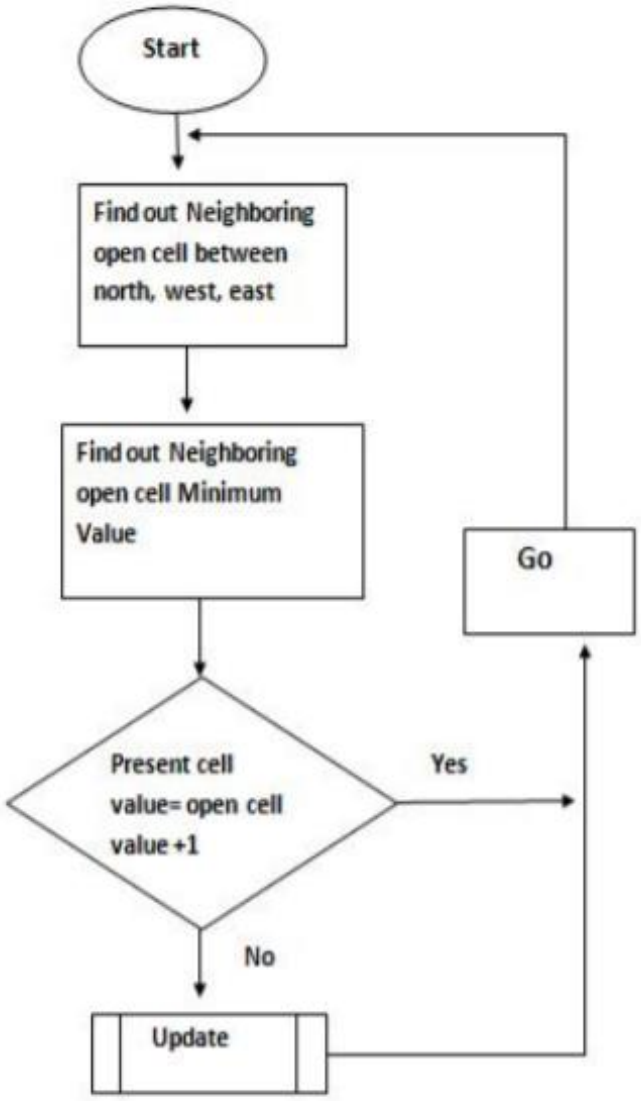
Depth First Search Algorithm

In DFS, starting from the root of the graph the exploration continues towards the deeper region of the tree or graph until it hits a wall and needs to back track. The whole maze is mapped as a graph where the nodes or vertexes are considered as maze cells. Terminating from the source or original cell, the mice visits all cells by visiting each door of that cell once in each direction. Mice continue searching the cells until it reach the destination cell. Instead of marking each cell, it keeps track of the cell walls. At the initial stage, all the doors of original cell kept unmarked. This is also known as Tremaux algorithm



Floodfill Algorithm or Bellman's Algorithm

Floodfill is a specialized DFS (Depth First Search). In floodfill, it looks for the all nodes within an array which is connected with the start and end vertexes. In case of maze, the start node is taken as the position of mice at the beginning of the run and the end node is the final destination, where the mice has to reach by solving the whole maze in shortest possible time. The whole maze is flooded with numerical values which are known as distance value



Confirm that the stack is empty and push the current cell onto the stack. The current cell is the one where mice is standing on. Repeat the instructions given bellow. Iterate it until stack is empty.

{ PULL A CELL FROM THE STACK.

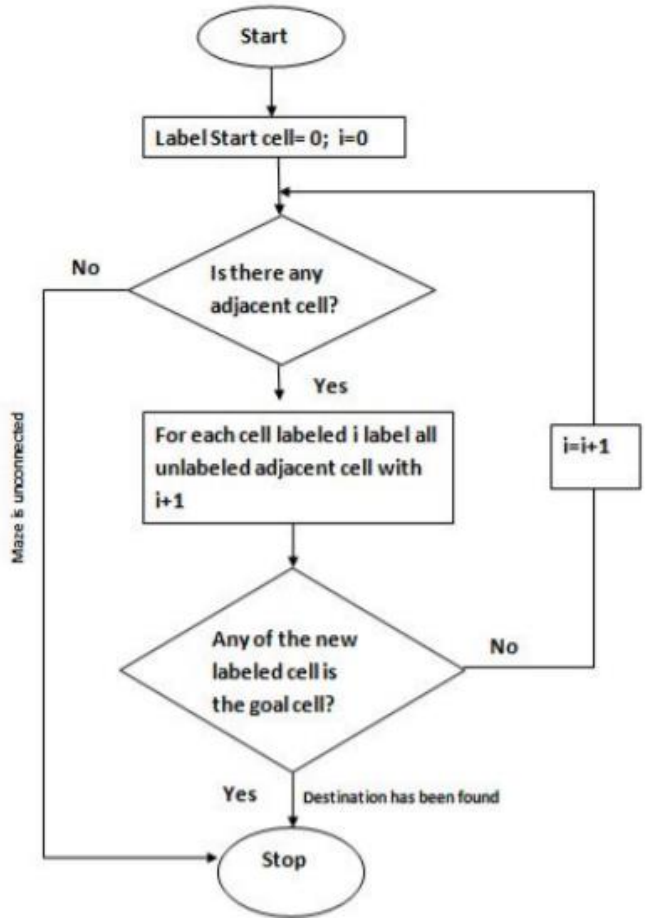
DISTANCE VALUE OF PRESENT CELL = 1 + THE MINIMUM VALUE OF ITS OPEN NEIGHBORS.

NO -> CHANGE THE CELL TO 1 + THE MINIMUM VALUE OF ITS OPEN NEIGHBORS AND PUSH ALL OF THE CELL'S OPEN NEIGHBORS ONTO THE STACK TO BE CHECKED

YES -> DO NOTHING }

Breadth First Search Algorithm

Breadth First Search is a special case of Uniform Cost Searches (UCS). UCS is a weighted graph search. It use cost storage to determine the order of nodes visited. Breadth First Search is a special case of Uniform Cost Searches (UCS). UCS is a weighted graph search. It use cost storage to determine the order of nodes visited



Conclusions:

Breadth first search and Depth first search are quite similar. The choice is dependent upon the maze structure. DFS may not provide the best solution as it tends to explore every possible path to reach the destination.

A vital problem with DFS is, it may stuck in a dead-end.

But BFS doesn't stuck in a blind lane within the maze and it will always find the shortest path.

The summary between DFS and BFS can be drawn as bellow:

- Memory requirement for coding is higher in DFS than BFS.
- In case of large mazes BFS is more appropriate than DFS.
- If the maze contains several possible paths to reach the destination then DFS may be time consuming. So it is suggested not to use DFS in those scenarios.
- If the consideration point is to find the shortest path then BFS is preferred then DFS.

Floodfill is also a potential GT algorithm. It always find the solution and the number of cell traversed will be lower compared to DFS. It also posses the potential to find shortest possible path -

The memory space required is also very high in Floodfill. Following statements can be drawn to compare BFS and Floodfill.

- If the maze is large then Floodfill should be avoided. BFS will give satisfactory performance in this scenario.
- For small mazes (i.e. 8x8) Floodfill is suggested to use. It will find the shortest path quickly than BFS.

GT algorithms are far more proficient compared to the NGT

Maze Solving Algorithms for Micro Mouse

Link : https://ieeexplore.ieee.org/document/4725791

Authors : Swati Mishra

The speed of robot to find its path, affected by the applied algorithm, acts the main part in the present project. The flood-fill algorithm involves assigning values to each of the cells in the maze where these values represent the distance from any cell on the maze to the destination cell. The destination cell, therefore, is assigned a value of 0. If the mouse is standing in a cell with a value of 1, it is 1 cell away from the goal. If the mouse is standing in a cell with a value of 3, it is 3 cells away from the goal. Assuming the robot cannot move diagonally

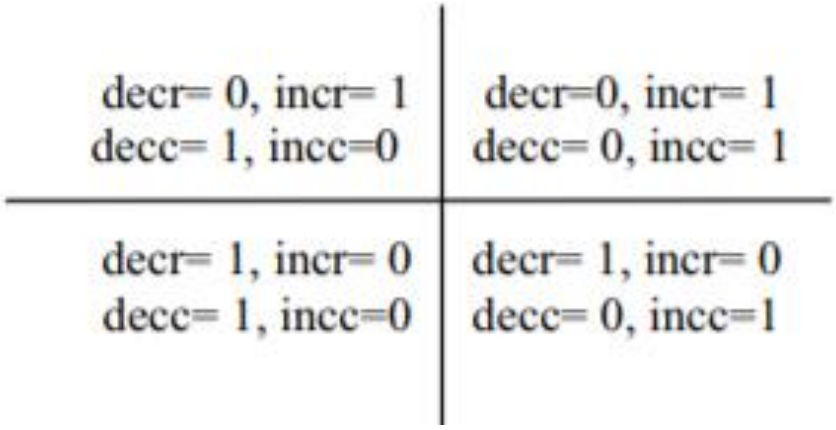
The maze is represented as a 16x16 array in the memory.

The centre is given the value (0, 0).

All cells in its immediate vicinity are assigned 1, the cells next to it as 2, and so on.

The array is divided into 4 symmetrical regions and then the assignment is done.

- Upper left quarter, loop decrements the column, increments the row: R= R+j, C=C-i, i, j vary from 0 to 8.
- Upper right quarter, loop increments the column, increments the row: R=R+j, C=C+i, i, j vary from 0 to 8.
- Lower left quarter, loop decrements the column, decrements the row: R=R-j, C=C-i, i, j vary from 0 to 8.
- Lower right quarter, loop increments the column, decrements the row: R=R-j, C=C+i, i, j vary from 0 to 8.



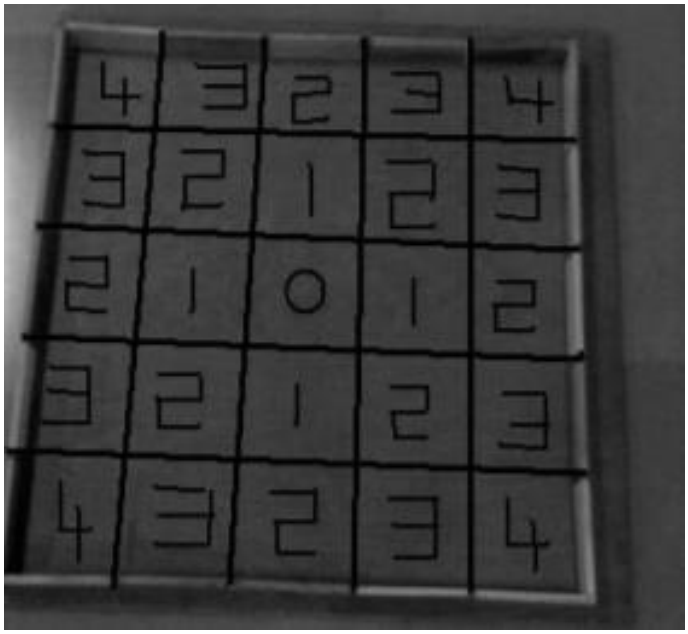
So it is combined into a MATHEMATICAL EQUATION as follows:

Row increment and decrement: R-(i*decr) + (i*incr)

Column increment and decrement: C-(j*decc) + (j*incc)

Now when the assignment is done, the entire equation is as follows: (where the variables i, j vary in the loop from 0 to 8)

```
MAZE[R-(i*decr) + (i*incr)][C-(j*decc)+j*incc)]
= i+j;
```



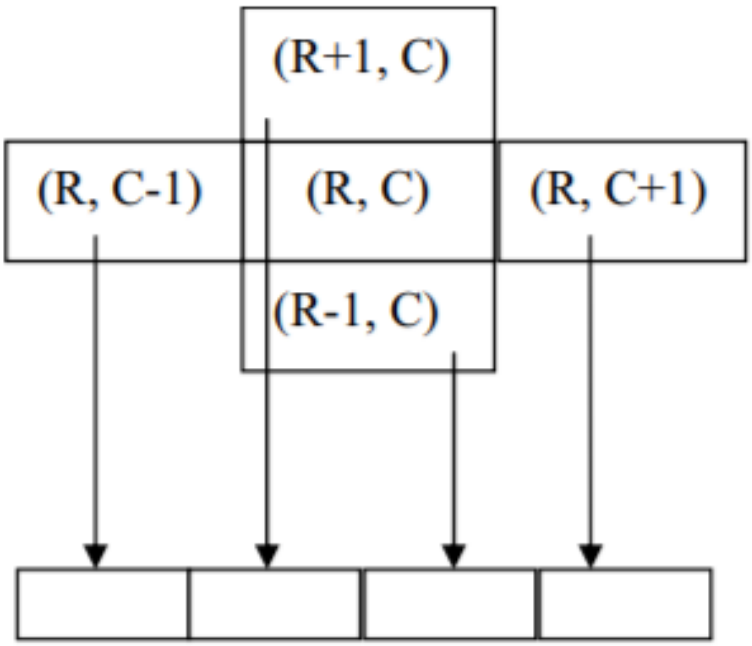


Figure 5. Storing elements in array; temp [4]

The values of the neighbors are stored in the above array. After the values are stored in the array, they are sorted using any kind of sorting. We have used here selection sort.

After this, the array is ready for further processing, which includes the deciding of which path to be taken and which values in the map to be changed.

The maze after being flooded is then traversed and the map of the maze is updated after every traversal. Every time a new cell is traversed, it creates the array described above and decides the lowest value nearby that can be

START: form array temp[4] for Maze[R][C].

STEP 1: From the array, select the ith element, (i=1 initially)

STEP 2: If the value temp [i]<= Maze [R][C] , then go to step 4. If temp [i]>=Maze[R][C], call check(R,C).

Step 3: if the value temp [1]=256, turn 180 deg, i=i+1,goto step 1

STEP 4: locate the cell of the value temp [i].

STEP 5: check if the wall is present in the way, if yes then, i++, go to step1

STEP 6: check if Maze[R1][C1]=temp[i+1], if yes, then use call Locate(R, C, temp(i+1)) algorithm defined below, to locate its cell.

STEP 7: store the result in (R2, C2)

STEP 8: call the function direction of move (R1, R2, C1, C2)

STEP 9: move to Maze (R', C')

STEP 10: update value, R=R', C=C'

STEP 11: check if Maze[R][C]=0, if yes, call return to start ( ), else go to START.

STEP 12: call follow ( ).

Similarly for other directions

Conclusion:

if we don't have any time and hardware constraint we can effectively use the Djikstra's algorithm, but if both are the constraints then Flood fill would be superior to others.

Further, if we do not wish to have any complex calculation to embed in the system, that means, if we have a memory constraint as well as the maze to be solved is pretty easy, we can stick to the left\right wall follower.

But for this we need to have a previous knowledge of the maze, whether it is rightwalled or left-walled.

Thus, the flood fill is by far the most effective of all, with fewer or almost no drawbacks apart from complex software which is difficult to code.

A Potential Maze Solving Algorithm for a Micromouse Robot

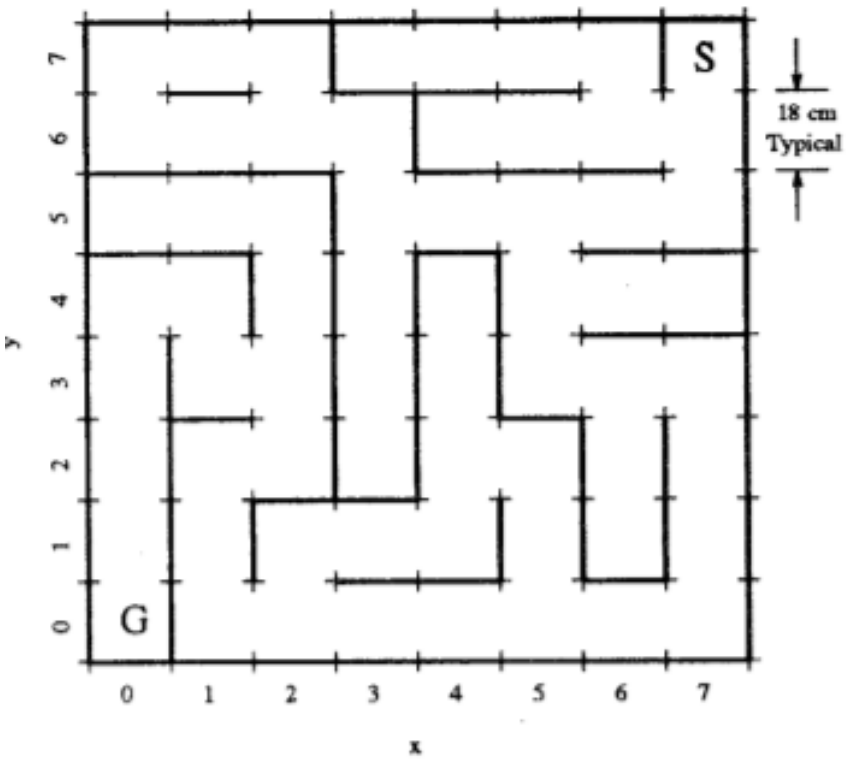
Link : https://ieeexplore.ieee.org/document/4725791

Authors : L. Wyard-Scott Q.-H. M. Meng

Discretely assigned potential levels can he effectively used in making autonomous route decisions for a mobile robot to reach a goal. This paper demonstrates methods of assigning and manipulating these artificial potentials to provide locally optimized path choices while maintaining the integrity of the potentials. The basic algorithm is improved by retaining information of the number of decisions that have been made. Results from implementation and simulation of the algorithm for a Micromouse maze-solving robot are presented. Consideration is given to implementation using a limited power microprocessor.

The goal is to provide locally optimized routing choices based upon odometry and limited sensory data. The starting position of the automaton is known, as is the goal location, but knowledge of the region between the start and finish points is limited or unobtainable.

The maze consists of 16 by 16 squares of size 18cm by 18cm. The starting point is one of the 4 corners at which the mobile robot begins with a clockwise orientation; the goal location is one of the four squares which lie in the centre of the maze. To prevent graphical demonstrations from becoming cluttered, only one quadrant of the regulation size maze is used. One-quarter of a typical maze structure is shown in Fig. 1,



The approach is to assign potentials in such a way that the initial assignment and manipulation of the potentials in the solving algorithm will not require extensive computational resources. The simplest manner of assignment for an environment such as the Micromouse maze is to define the potentials at the centre of the grid squares. The potential value will be related to the distance required to travel to the goal if no obstacles are present:

$$V(x, y) = \phi(\Delta x, \Delta y)$$

If the goal coordinates are specified as the origin of the Cartesian coordinate frame, a potential distribution for the micromouse maze problem is

$$V(x, y) = K(x + y)$$

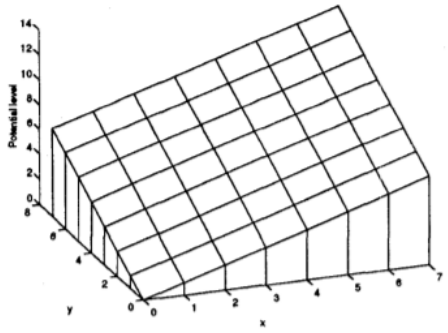


Figure 2: Assignment of initial potentials in a Cartesian environment. $V(x, y) = x + y$.

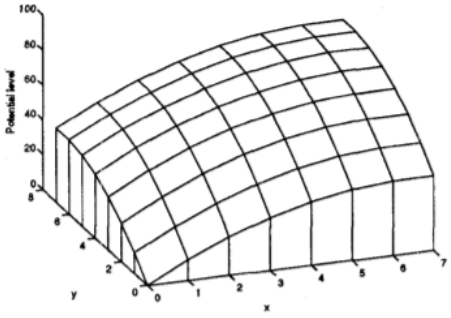
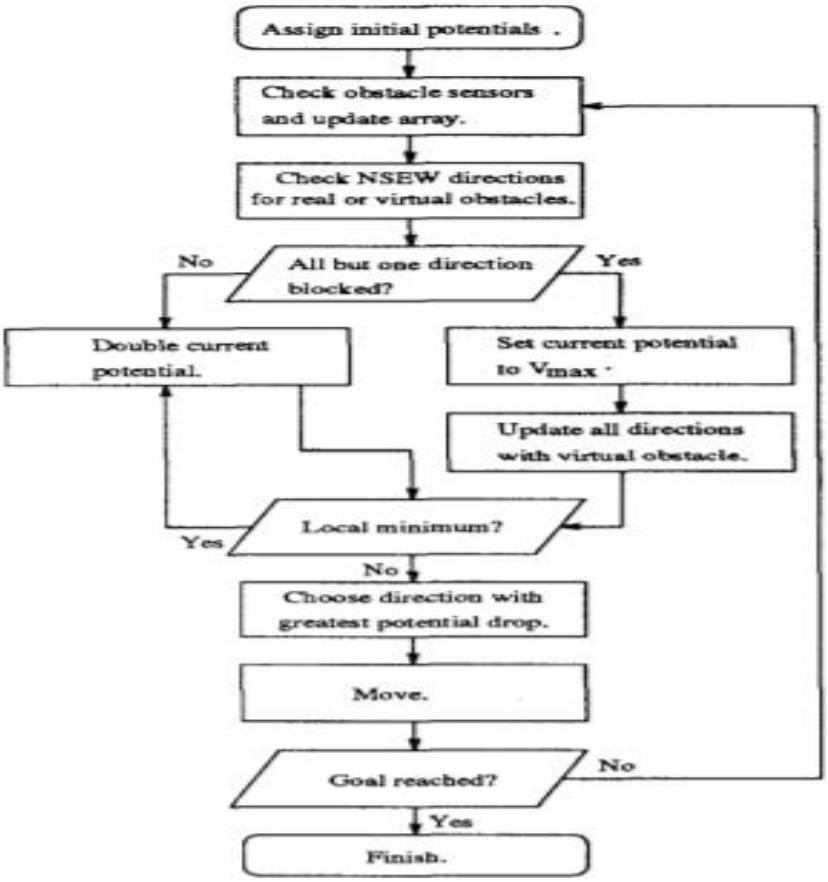
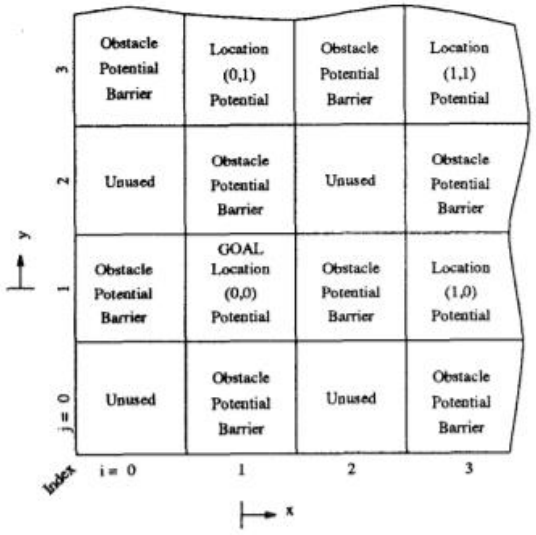


Figure 3: Initial potential distribution where examination of the problem yields that the highest probability of solution is in routes along the bounds of the terrain.





Sensory data is assumed to contain short range information about obstacles within one square unit of resolution of the potential assignments (for the Cartesian method) around the point which has been assigned the discrete potential

Allocation of the array in this manner is an exploitation of the nature of the maze. The values stored in the array between the potential values are the potential values of infinitely thin Obstacles. A value of zero indicates that no wall is present, and a large value, V,,, , is assigned if a wall is present.

Conclusion:

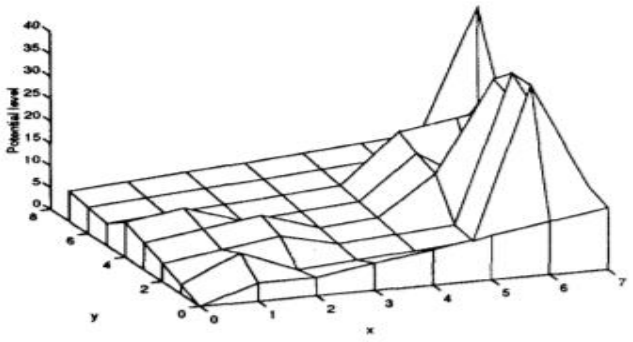
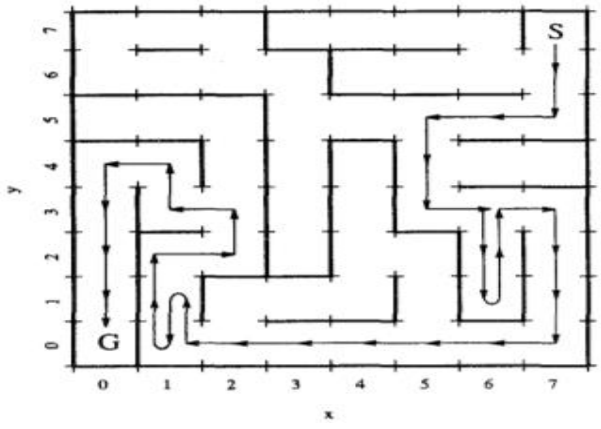


Figure 8: The final potential distribution after the goal has been reached.



Final Potential Distribution of the above maze.

1] **https://harvardrobotics.com/micromouse#mm1**

My Notes :
This micromouse is built by the Harvard Robotics Club. Following are the specifications.

2016 Design Specifications

- 3 IR distance sensors and 2 Hall Effect motor encoders to determine the robots position and orientation as well as for detecting walls.
- Powered by 2 DC motors, controlled through an H-bridge motor controller via a pulse-width modulated signal
- Controlled by an Arduino Micro
- Integrates the information from all onboard sensors into a **flood-fill maze** navigation algorithm.

2017 Design Specifications

- 5 ToF sensors at the front of the chassis, giving 180 degree input range.
- Custom PCB design - smaller, faster, minimal wires for sleeker design.
- One magnetic Hall effect quadrature encoder on each motor.
- Quadrature encoder library keeps track of motor shaft position.
- PID control:
    - run on shaft position for dead reckoning.
    - run on ToF sensor reading for wall-following and front-aligning.
- Teensy 3.2 microcontroller
- Integrates the information from all onboard sensors into a **flood-fill maze** navigation algorithm.

2]
http://www.physics.unlv.edu/~bill/ecg497/Drew_Tondra_report.pdf

My Notes :
This article gives an entire overview of the micromouse maze solver project. Right from what the project idea is, history of mazes, to components of the micromouse(hardware), its efficiency and algorithmic analysis as well as testing and simulation. A must read basic guide before starting the project! Also helped me identify that my interest lies more in the software and simulation part of this project.
Flood fill algorithm is more efficient that depth-search or wall following algorithm

3]
2010 10th IEEE International Conference on Computer and Information Technology (CIT 2010)
An Algorithm of Micromouse Maze Solving Jianping Cai, Xuting Wan, Meimei Huo* , Jianzhong Wu School of Computer Science Zhejiang University City College Hangzhou, China e-mail: caijp@zucc.edu.cn

My Notes :
The partition-central algorithm divides a maze into 12 partitions, and applies different rules to different small areas; the exploring process is more flexible and improves the 1999 intelligence of a micromouse. The partition-central algorithm performs more central trending feature than classic central algorithm, and its advantage shortens the maze exploring time to some extent, which is verified in most experiments that have been finished; meanwhile, shorter exploring time reduces possible collision to the maze wall when the micromouse running. In turn the micromouse is able to traverse the maze to reach the destination faster. A 2-dimensional micromouse simulation software is developed for micromouse debugging and competition preparation purpose, which is helpful to optimize the algorithm in practice.

4]
Design and Practice from the Micromouse Competition to the Undergraduate Curriculum Shenghua Dai, Bingxian Zhao, Zhengjiao Li, Ming Dai School of Electronic and Information Engineering Beijing Jiaotong University Beijing, China, 100044 Email: zhaobingxian@bjtu.edu.cn

My Notes :
Types of Algorithms implemented till now :

1) Random mouse algorithm

2) Wall following

3)Pledge algorithm

4)Trémaux's algorithm

5)Dead-end filling

6) Recursive algorithm

7) Maze-routing algorithm

8) Shortest path algorithm

MAZE-SOLVING USING MICROMOUSE

VEDANT GHODKE

- **WHAT IS A MICROMOUSE?**

Micro-mouse is an autonomous robot designed to negotiate a maze in a shortest possible time. It cannot jump over, fly over, climb, scratch, cut, burn, mark, damage, or destroy the walls of the maze. The maze is composed of multiples of an 18 cm x 18 cm unit square. The maze comprises 16 x 16 unit squares.

The walls of the maze are 5 cm high and 1.2 cm thick that can have 5% of tolerance. The outside wall encloses the entire maze. The sides of the maze walls are white, the top of the walls is red, and the floor is black. The maze is made of wood, finished with non-gloss paint.

The start of the maze is located at one of the four corners. The start square is bounded on three sides by walls.

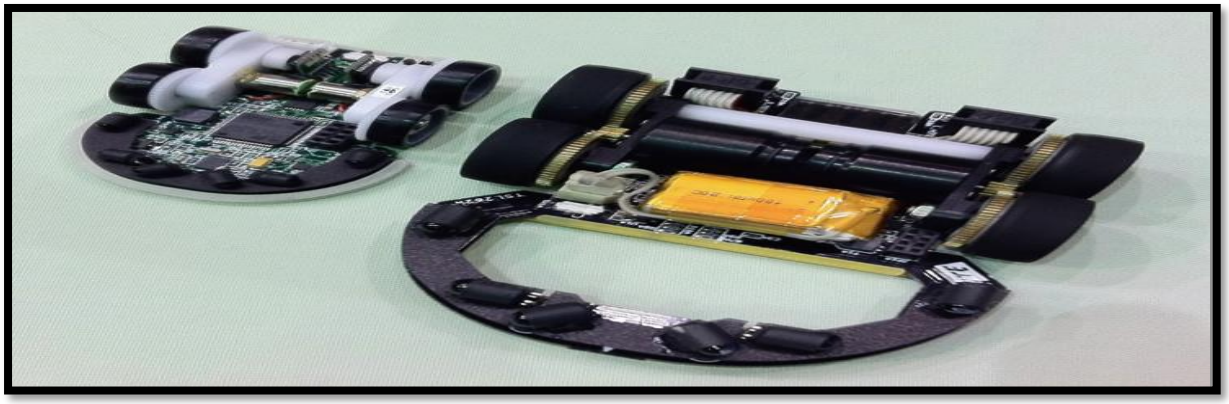


Fig 1. Micromouse for classic (right) and half-size (left) micromouse contests made by Khiew, Tzong-Yong from Singapore

Fig 2. A Micromouse prototype

The following is a list of a few very important and relevant IEEE research papers that will contribute and help towards the successful operation of this project:

1. **Algorithms for Micro-mouse:**

   Authors: Manoj Sharma (Kaizen Robeonics)

   Link : https://ieeexplore.ieee.org/document/5189850

This paper covers one of the most important areas of robot, "Decision making Algorithm" or in lay-man's language, "Intelligence". The environment around the robot is not known, so it must have decision-making capabilities. For starting in the field of micro-mouse it is very difficult to begin with highly sophisticated algorithms. This paper begins with very basic wall follower logic to solve the maze. And gradually improves the algorithm to accurately solve the maze in shortest time with some more intelligence. The algorithm is developed up to some sophisticated level as "**Flood Fill Algorithm.**"

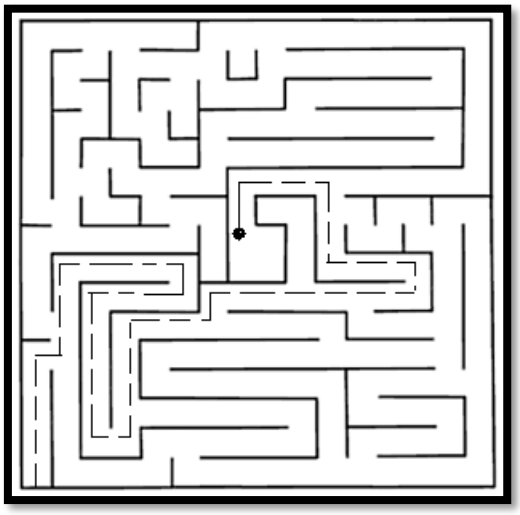


Fig 3: The maze solver using Flood Fill Algorithm

2. **Flood Fill Maze Solving with Expected Toll of Penetrating Unknown Walls for Micromouse:**

   Authors : Zhuang Cai, Lu Ye, Ang Yang

   Link : https://ieeexplore.ieee.org/document/6332345

This paper begins with expounding the fundamental working logic and calculating procedure of traditional Flood Fill Algorithm. After summarizing the Flood Fill's intelligence in the information of known walls and its non-intelligence in the unreasonable assumptions about the unknown walls, this paper introduce a new conception and present a modified algorithm called ET-Flood Fill.

It is assumed that the unknown walls are penetrable and the penetrating causes extra expense, which is quantified by the expected toll value T. The toll value is an expectation value due to the uncertainty of the existence of unknown walls.

With reasonable estimating and adjustment of the toll value, ET-Flood Fill has a better performance than traditional Flood Fill. Its intelligence reduced the turning times and the steps taken from starting cell to destination cell.

Additionally, the computational complexity remains the same with Flood Fill. For each step of the cell-exploration process, the worst case time complexity is $O(n^2)$.

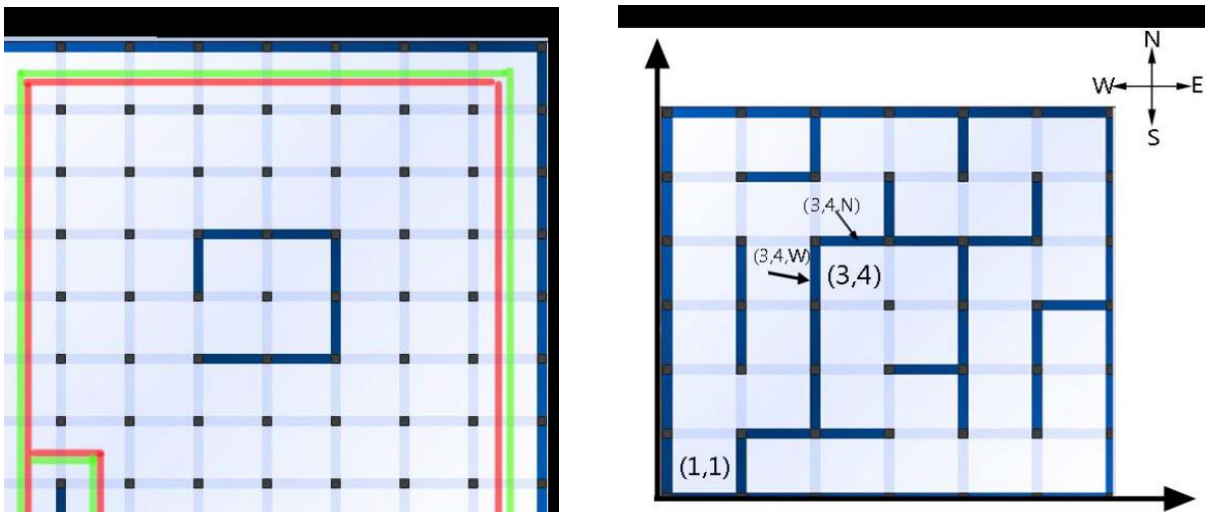


Fig 4. (a) Example of dead loop maze for wall followers algorithm, (b) The coordinate system used in the maze interpretation

3. **A Simple and Efficient Diagonal Maze-solver for Micromouse Contests and Intelligent Mobile Robot Education:**

   Authors : Juing-Huei Su, Hsin-Hsiung Huang, Chyi-Shyong Lee

   Link : https://ieeexplore.ieee.org/document/6852576

A simple and efficient time-based diagonal maze solver for classic and half-size micro-mouse contests is presented in this paper. The algorithm is first developed in the graphical user interface development environment (GUIDE) of MATLAB, and then used as a training and tutorial tool for students who are involved in contests oriented projects about intelligent mobile robots.

It is also found that the firmware implementation of the algorithm in a dsPIC 16-bit microcontroller consumes just 4.2ms to finish searching for the best route from the start cell to the goal area in a 16x16 maze of classic micro mouse contests.

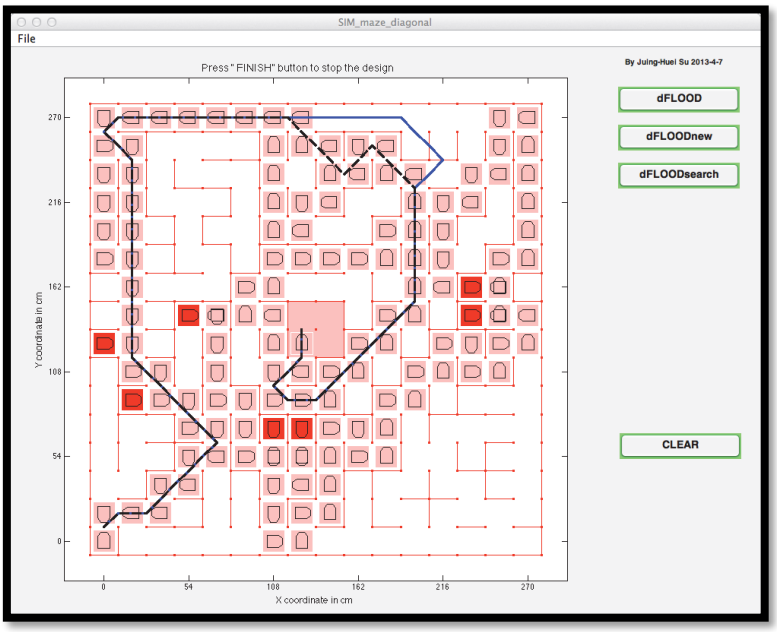


Fig. 5. A graphical user interface for maze solving in a classical micromouse contest

4. **Survey on Techniques used in Autonomous Maze Solving Robot:**

   Authors: Bhawna Gupta, Smriti Sehgal

   Link : https://ieeexplore.ieee.org/document/6949354

This paper presents a survey on techniques used in Autonomous Maze Solving Robot aka Micromouse. Autonomous movement is an important feature which allows a robot to move freely from one point to another without the mediation from human being. The micromouse is required to solve any kind of maze in shortest interval of time. Autonomous movement within the unknown area requires the robot to investigate, situate and plan the outlaying area. By solving a maze, the referenced algorithms and their pros and cons can be studied and analysed.

A simply-connected maze have pathways that never re-interface with each one in turn, so every way you pick either prompts extra ways or to a deadlock. The solution to a simply-connected maze can always be found by following the "left hand rule " or “the right hand rule”—simply walk forward, keeping your left hand or right hand respectively on the wall at all times. The maze shown above is a simply-connected maze.

A multiply-connected maze holds one or more entries that circle go into different sections, instead of prompting deadlocks. A generally planned multiply-connected maze is more hard to understand than an essentially joined maze, for clients will invest a lot of time basically going around in loops.

Fig. 6. Multiply Connected Maze