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

Adil M. J. Sadik, Maruf A. Dhali, Hasib M. A. B. Farid, Tafhim U Rashid, A. Syeed
Department of Electrical and Electronic Engineering.
Islamic University of Technology.
Dhaka, Bangladesh
sadik.adil@gmail.com, ahmed.maruf@hotmail.com, hasib.farid@yahoo.com, tafhim.eee.iut@gmail.com, anas.syed@gmail.com.

***Abstract*— Solving a 3-D square maze through an autonomous robot is gaining immense popularity among the robotics aspirants. IEEE has established a set of rule for this and launched a competition named "Micromouse" where an autonomous robot or mice solves an unknown maze. Without deploying Artificial Intelligence technique it's not possible to do this task efficiently. Several algorithms which originate from graph theory (GT) and non graph theory (NGT) are currently being used to program the robot or mice. In this paper we have elucidated how graph theory can be used to solve mazes. With adequate investigation it is verified how graph theory dominates over non graph theory algorithms. The process of generating maze solving algorithm from graph theory is also explained. To compare the algorithms efficiency, they are simulated artificially and a comprehensive study is done by interpreting the statistics of interest. The simulation results lead us towards a conclusion about the nature, behavior and efficiency of these algorithms. Upon considering all the regulating factors which can alter the performance of an algorithm, some proposals have been drawn. It will be helpful to any micromouse aspirant while choosing an algorithm to design the robot.***

***Keywords-Micromouse, Flood-fill, Tremaux, Depth search, Graph Theory.***

## I. INTRODUCTION

A maze is a puzzled way which consists of different branch of passages where the aim of the solver is to reach the destination by finding the most efficient route within the shortest possible time. Artificial Intelligence plays a vital role in defining the best possible way of solving any maze effectively. 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 algorithms.

This paper is based on solving a particular type of three-dimensional maze using graph and non graph theory algorithms where the solver is a self governing robot. The maze may be made up of a 16x16 grid of cells. Each cell is 18cm square with walls 5cm high. There are a set of rules fixed by IEEE for Micromouse competition which can be found in the reference [2]. By incorporating intelligent procedure with the existing graph theory algorithms, some Micromouse Algorithms have been developed. This research implemented the use of graph theory algorithms i.e. flooding, DFS (Depth First Search) in Micromouse. Analytical approach is taken to evaluate BFS (Breadth First Search). The performance and outcome of these algorithms are also analyzed and compared. With adequate reference and observation it is obvious that graph theory technique is the most efficient in solving Micromouse mazes than other techniques.

In section II, the generations of the micromouse algorithms from graph theory algorithms have been elucidated. A more generalized algorithm (not originated from graph theory) is described in section III. Section IV is based on the simulation and analysis of these algorithms. Section V is based on comparison and decision taking. Finally some conclusions have been drawn by interpreting the simulation result.

## II. GRAPH THEORY AND ALGORITHMS

The development of Micromouse algorithms from graph theory algorithms is elucidated in the beneath section. The graph searching algorithms are utilized in a slightly different way to solve mazes.

### *A. Depth Firse 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. Algorithms start searching from a specific vertex and then it navigate the graph by branching out corresponding vertices until it reach the final or destination point. This searching algorithm can be efficiently used in solving micromouse mazes. 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

978-0-7695-4225-6/10 $26.00 © 2010 IEEE
DOI 10.1109/AICI.2010.18

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. How DFS can be used in solving maze is showed in Fig.1.

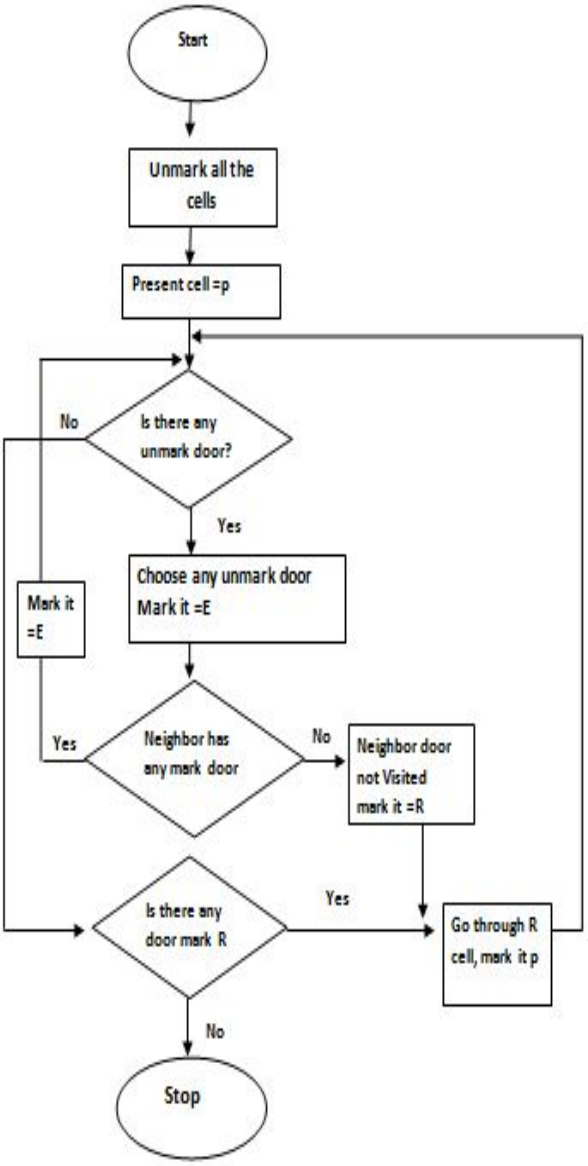


Figure 1: Flow Chart for DFS Algorithm.

*B. 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. [3] These values represent the distance from any cell on the maze to the end/destination cell. So, the destination cell will have distance value of 'zero'. The mice or robot follows the path of decreasing distance value and it is updated in each step. To increase the efficiency of this algorithm, some changes have been made and eventually "modified flood-fill algorithm" appears. Corresponding flowchart at Fig.2.

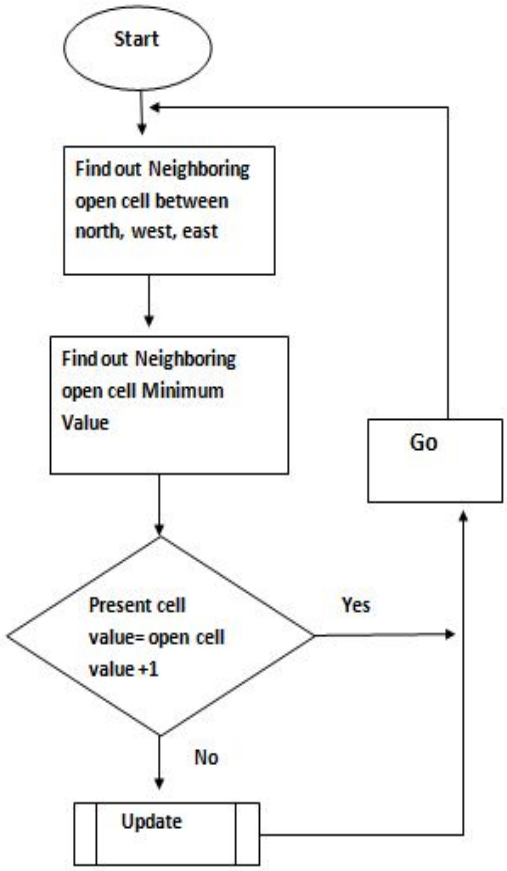


Figure 2: Flow Chart for Floodfill Algorithm

Update Subroutine:
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
}

### C. *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. [3] In BFS, the job is to reach the goal vertex from the source vertex. Starting from the source vertex, the entire layer of unvisited vertex is visited by some vertex in the visited set and recently visited vertexes are added with the visited set. The analogy of BFS with Micromouse maze is drawn by considering the vertexes as maze cells. In Micromouse, the BFS algorithm labels cells searching from the start cell to all the nearing neighbors. Start cell is marked as 'zero' (0). The program keeps records of which cell are immediate neighbors of start cell. BFS is capable to find the shortest path. The search continues until it finds the goal. Fig.3 shows the flow chart for this algorithm.

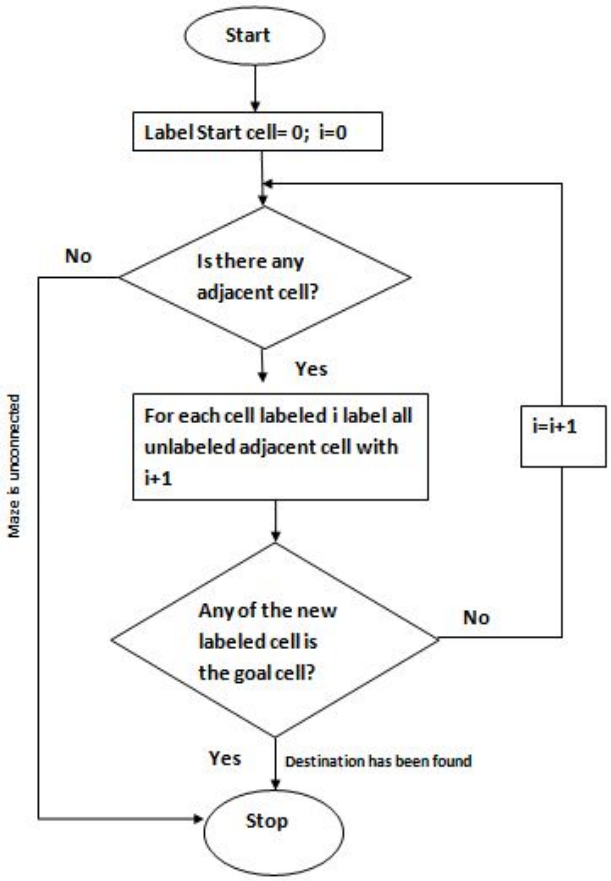


Figure 3: Flow Chart for BFS Algorithm

## III. GENERAL ALGORITHM

Without applying graph theory, there are some methods which can be used to solve Micromouse maze with less efficiency and reliability.

### A. *Wall Follower Algorithm*

It is the simplest algorithm. Wall follower is easy to program and implement. Two types of wall follower algorithm exist: right wall follower and left wall follower. According to the wall follower algorithm, the mice only follow the right or left side wall throughout its whole path. Mazes build with less complexity can be solved by wall follower algorithm. But nowadays the mazes, especially IEEE Micromouse competition mazes are built in such a way so that it can't be solved by using this simplest algorithm.

## IV. SIMULATION RESULT, ANALYSIS AND COMPARISON

Simulation statistics, graphs and corresponding assessments are described in this section. Upon interpreting the simulation outcomes, extensive comparison is done among GT algorithms which lead towards a conclusion about their effectiveness. Comparison between GT and NGT algorithms has also been done by juxtaposing their performance in graphical form.

TABLE I. SIMULATION DATA

| Maze | Algorithm | | Time taken to Solve the Maze *t* (Sec) | Cell traversed (#) | Avg. time (sec) | Avg. Cell Traversed (#) |
|---|---|---|---|---|---|---|
| Maze 1 | NGT | Left Wall Follower | 14.80 | 147 | 15.08 | 155 |
| | | Right Wall Follower | 15.36 | 164 | | |
| | GT | Floodfill | 12.58 | 150 | 9.89 | 195 |
| | | DFS | 7.20 | 90 | | |
| Maze 2 | NGT | Left Wall Follower | 7.45 | 71 | 8.78 | 83 |
| | | Right Wall Follower | 10.12 | 95 | | |
| | GT | Floodfill | 3.65 | 41 | 3.23 | 45 |
| | | DFS | 2.81 | 49 | | |
| Maze 3 | NGT | Left Wall Follower | X | X | X | X |
| | | Right Wall Follower | X | X | | |
| | GT | Floodfill | 2.44 | 25 | 2.6 | 37 |
| | | DFS | 2.75 | 49 | | |

X= Algorithm unable to solve the Maze.
GT=Graph Theory
NGT= Non Graph Theory.

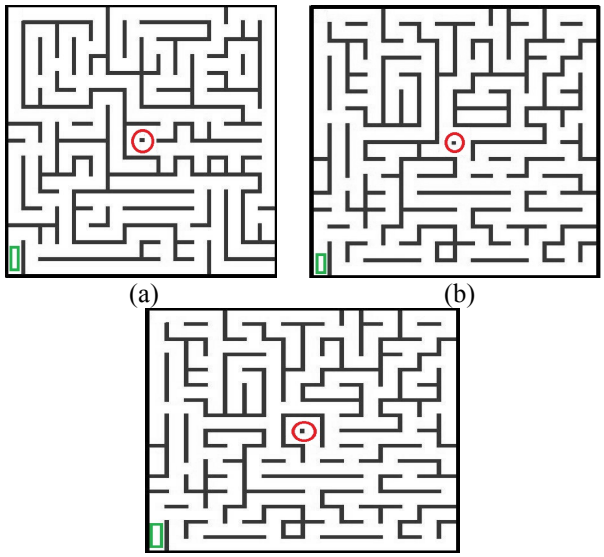


Figure 4: Sample 16x16 Mazes. (a) Maze 1, (b) Maze 2, (c) Maze 3.

Above figure depicts the sample micromouse mazes used for this simulation. Green rectangle represents the starting position where the red circle is the destination.

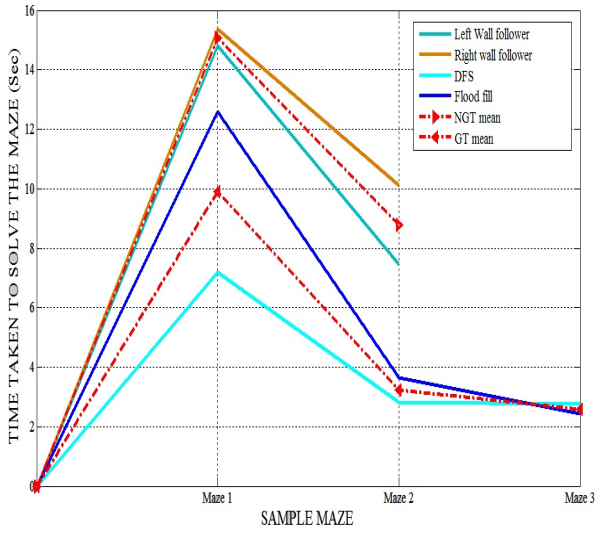


Figure 5: Graphical comparison between GT & NGT considering maze solving time.

The time taken by GT and NGT algorithms to solve the sample mazes are plotted in the figure above. It is obvious that the time requirement is much less in case of GT algorithms. For an example, DFS consumed about 7 second to solve the first maze where right and left wall follower spent more than 14 seconds. The average time requirement is also plotted. The reverse arrow dotted line is for GT and the forward arrow dotted line is for NGT. It can be inferred from their relationship in the graph that the NGT always tends to explore maze extensively before solving it. In some circumstances it also fails to solve the maze which is happened for maze 3.

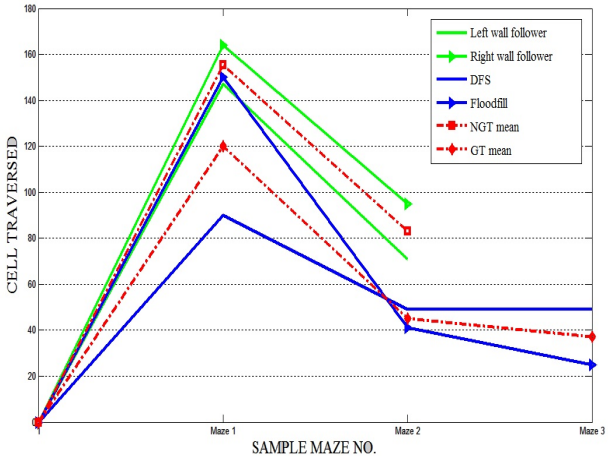


Figure 6: Graphical comparison between GT & NGT considering number of cell traversed.

Apart from time requirement, another important criterion to judge the efficacy of an algorithm is the amount of cell traversed within the maze. How extensively the maze search is done to figure out the shortest path from source cell to goal cell by the algorithm should be analyzed. For this reason, the number of cell traversed also taken into account and with the simulation statistics the above graph is plotted. One important observation can be noted here. Fig-4 and Fig-5 are of almost similar characteristics. Comparing average number of cell traversed, it is apparent that GT suppress NGT algorithm in every possible scenario.

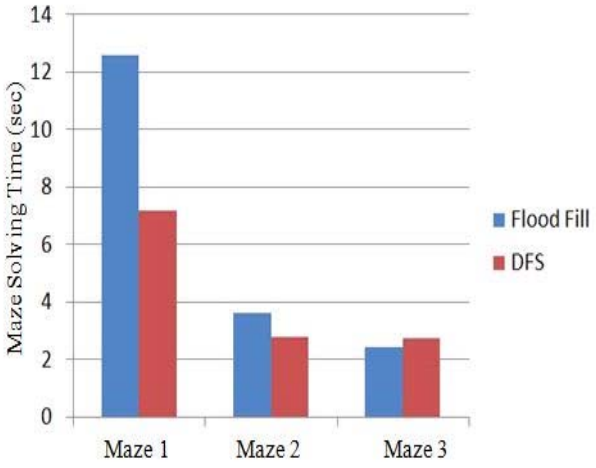


Figure 7: Graphical comparison between Floodfill and DFS considering number of cell traversed.

Eliminating NGT from the consideration, let analyze the performance of GT based algorithms. The pictorial view

Above shows the relationship in terms of maze solving time between Floodfill and DFS. Though Floodfill originates from DFS but its performance varies depending upon maze properties. As for example, Floodfill consumes too much time to reach the goal for maze 1. But the required time for DFS cut to half. Another important inspection for maze 3 (Floodfill solved the maze before DFS) indicates that the Floodfill may have satisfactory performance compared to DFS in some mazes. Theoretical investigation suggests that, Floodfill should traverse less number of cells than DFS in case of DFS, it can't be guaranteed that it will always find the shortest path by traversing minimum number of cells. Moreover it totally depends upon the maze configuration. But Floodfill is designed to find the minimum path by traversing fewer amounts of cells. The simulation data for maze 1 and 2 incorporate this statement. Considering these facts, Floodfill is preferred than DFS.

## V. COMPARISON AND DECISON

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. So, Floodfill is a strong candidate as BFS. But some inherent problem lies with Floodfill. In case of even mazes, some difficulties arise while coding. The reason behind this is there will be four different destination cells in case of one. Floodfill also requires large processing power. So, in case of large mazes it will be clumsy to use Floodfill. 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.

The simulation result of NGT algorithms indicates that, it is unable to solve all kind of mazes layout. The 'X' notation in the Table-I imply that, NGT failed to solve the corresponding maze. If the destination is isolated from surrounding maze boundary (fig. 4(c)) then wall follower won't be able to reach to the destination. [1] Because of these erratic and unpredictable characteristics of NGT algorithms, they are not considered as ideal.

## VI. CONCLUSION

Though it can be inferred from simulation results that the GT algorithms are far more proficient compared to the NGT but NGT should not be completely excluded from the maze solving scenario. The high computational complexity and programming skill make the GT rarely realizable to the beginners. The present trend of designing maze for competition is to make it more puzzled and perplexed where NGT fails to succeed in almost all the cases. On the other hand the maze solving phenomena is focused to be implemented in the extensive practical ground where the efficiency and accuracy of GT algorithm makes it superior to NGT.

## REFERENCES


[1] Sadik A. M. J, Farid H. M. A. B., Rashid T. U., Syed A., Dhali M.A. "Performance analysis of micromouse algorithms," Proc. 25th International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC), Pattaya, Thailand on 4-7 July, 2010; PID 0242, pp. 544–547.

[2] "Micromouse 2010 Competition Rules." IEEE Region 2 Student Activities Conference 2010 Web Page. Web. 21 Nov. 2009.[http://www.temple.edu/students/ieee/SAC/competitions.html.]I . S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[3] Manoj Sharma,"Algorithms for Micro-mouse", Proc. 2009 International Conference on Future Computer and Communication. DOI 10.1109/ICFCC.2009.38