IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 17, Issue 1, Ver. IV (Jan – Feb. 2015), PP 24-29
www.iosrjournals.org

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

Keshav Sharma[1] Chirag Munshi[2]
School of Computer Science and Engineering VIT University Vellore, India
School of Computer Science and Engineering VIT University Vellore, India

Abstract: *This paper presents an efficient maze solving algorithm. IEEE has launched a competition named "Micro mouse" where an autonomous robot or mice solves an unknown maze. The mouse find its way from the starting position to the central area of the maze without any intervention. To solve the maze, the mice implements one of many different searching algorithms such as the DFS, flood fill, BFS, modified flood fill. Several algorithms which originate from graph theory (GT) and non-graph theory (NGT) are currently being used to program the robot or mice. To compare the algorithms efficiency, they are simulated artificially and a comprehensive study is done by interpreting the statistics of interest.*

I. Introduction

A maze is a puzzled way which consists of different branch of passages where the aim of the micro mouse is to reach the destination by finding the most efficient route within the shortest possible time. Micro mouse is a small wheeled-robot that consisting infrared sensors, motors and controller and act. 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. 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. The IEEE standard maze has 16×16 square cells, and each cell's length and width are both 18 centimeters. Each cell can be considered as a node which is isolated

by walls or edges. By incorporating intelligent procedure with the existing graph theory algorithms, some Micro mouse Algorithms have been developed i.e. flooding, DFS (Depth First Search) in Micro mouse. Analytical approach is taken to evaluate BFS (Breadth First Search). To increase the efficiency of flood fill algorithm, some changes have been made and eventually "modified flood-fill algorithm" appears. 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 Micro mouse mazes than other techniques. 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. Related Work

I. Depth First Algorithm (DFS)

Depth- First search is an intuitive algorithm for searching a maze in which the mouse first starts moving forward and randomly chooses a path when it comes to an intersection. If that path leads to a dead end, the mouse returns to his intersection and choose another path. This algorithm forces the mouse to explore each possible path within the maze, and by exploring every cell, the mouse eventually finds the center. It is called "depth first" because if the maze is considered a spanning tree, the full depth of one branch is searched before moving onto the next branch. The relative advantage of this method is that the micro mouse always finds the route. Unfortunately the major drawback is that the mouse does not necessarily find the shortest or quickest route and it wastes too much time exploring the entire maze.

II. Flood Fill algorithm

The speed of robot to find its path, affected by the applied algorithm, acts as 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.

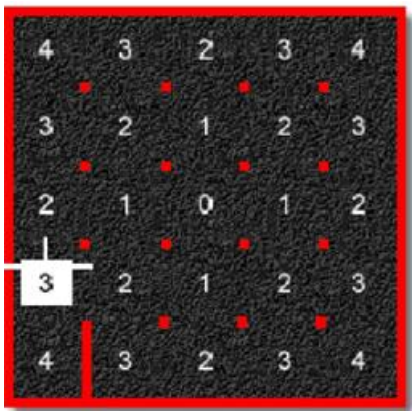

Figure1: Flood Fill Algorithm

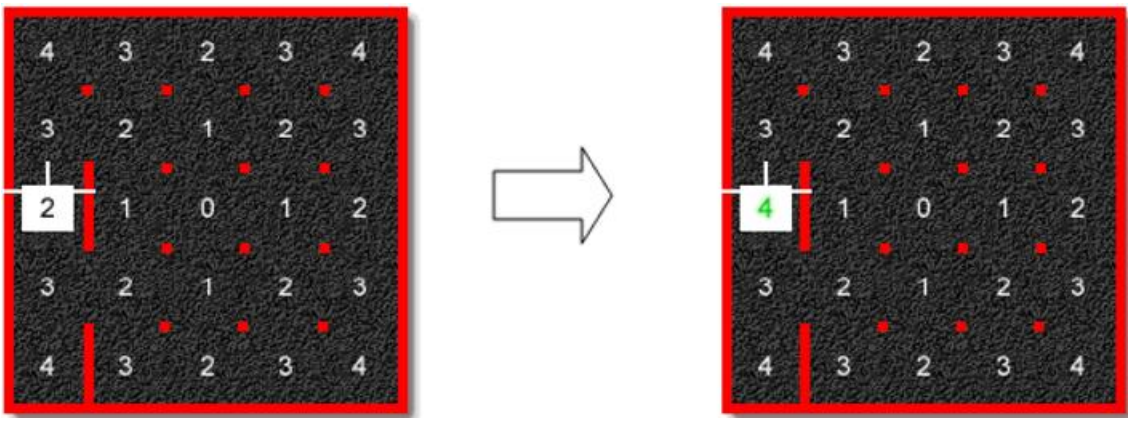

Figure2: Modified Flood Fill Algorithm

If the mouse is standing in a cell with a value of 3, it is 3 cells way from the goal. Assuming the robot cannot move diagonally.

Problems Of The Flood Fill Algorithm

Although the robot using flood-fill algorithm can find the shortest path to destination, but some problems still exist when using this flood fill algorithm in the process of maze solving. The problems are shown as follows.

- First, in each cell the robot arrived, it must run the four steps: update walls, flood maze, turn determination and move to next cell. That's the problem. Time is the key factor in this competition. This will affect the robot searching speed extremely.
- Second, flooding one time of a maze with 256 cells will costs the robot much time.
- Third, in order to find the destination as soon as possible, which direction the robot should to make when it gets to a cross cell which has two or more directions have the same value.

This algorithm is rather simple and provides a reliable method to finding the center of the maze; however, the new proposed algorithm can find the center more quickly so we will introduce this algorithm.

Modified Flood Fill Algorithm

The modified flood fill algorithm is similar to the flood fill algorithm for it also uses distance values to navigate the maze. The primary difference is that the modified flood fill algorithm does not "flood" the entire maze with values. It modifies only the values that need to be changed. For example, if a wall is encountered and the robot is not in the destination cell, it updates the value of that cell to 1 + the minimum value of its open neighbors.

Examining Figure2, when the robot encounters a wall to the east and can only move north or south. The north and south cells (open neighbors) are checked and we find that the current cell's new value is 1 + the minimum value of its open neighbors or 1+3=4.

Once the robot has found the destination cell, it can return to the beginning of the maze using the distance values. On return, it is often a good practice to double check the wall positions. Once the micro mouse

has returned to the beginning, the maze is solved and the mouse can scamper off for a speed run to the center of the maze.
So, the modified flood fill process for updating the distance values is:

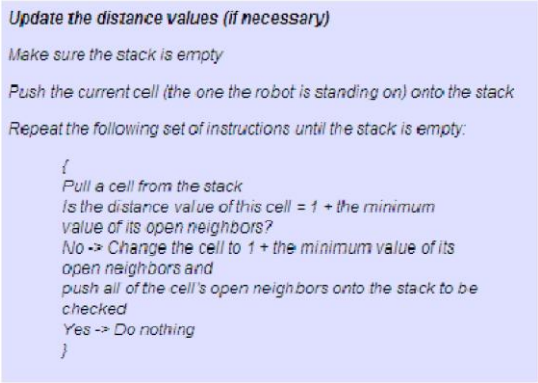

Figure 3: Updating Distance Values with Modified Flood Fill Algorithm

Using the algorithm from Figure3, the modified flood fill algorithm now becomes the following and should be executed every time the mouse enters a new cell.

1. Update the wall map
Is the cell to the North separated by a wall?
Yes -> Turn on the "North" bit for the cell we are
Standing on and No -> Do nothing
Is the cell to the East separated by a wall?
Yes -> Turn on the "East" bit for the cell we are standing on and
No -> do nothing
Is the cell to the South separated by a wall?
Yes -> Turn on the "South" bit for the cell we are standing on and
No -> do nothing
Is the cell to the West separated by a wall?
Yes -> Turn on the "West" bit for the cell we are standing on and
No -> do nothing
Now we need to call the modified flood fill to flood only that part of the maze which is required. The cell values are updated according to the following rule,
If a cell is not the destination cell, its value should be one plus the minimum value of its open neighbors. When the cell values violate the above rule, they need to be updated.

2. Update the distance values (only if necessary) (modified flood fill)
Make sure the stack is empty
Push the current cell (the one the robot is standing on) onto the stack
Repeat the following set of instructions until the stack is empty:
{
Pull a cell from the stack
Is the distance value of this 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
}
The stack empty check is not required in the program presented in this article.
The wall map is updated, required part of the maze is also flooded now what? Guess we need to make the right move.

3. Determine which neighbor cell has the lowest distance value
Is the cell to the North separated by a wall? Yes -> Ignore the North cell
No -> Push the North cell onto the stack to be examined

Is the cell to the East separated by a wall? Yes -> Ignore the East cell
No -> Push the East cell onto the stack to be examined
Is the cell to the South separated by a wall? Yes -> Ignore the South cell
No -> Push the South cell onto the stack to be examined
Is the cell to the West separated by a wall? Yes -> Ignore the West cell
No -> Push the West cell onto the stack to be examined
Pull all of the cells from the stack (The stack is now empty)
Sort the cells to determine which has the lowest distance value

4. Move to the neighboring cell with the lowest distance value.
After making the move, the above process is repeated again and again till the mouse reaches the center.

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. 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 Micro mouse maze is drawn by considering the vertexes as maze cells. In Micro mouse, 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.

III. General Algorithm

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

Wall Follower Algorithm (NGT)

Here, we are developing the wall follower logic. This works on the rule of following either left wall or right wall continuously until it leads to the center. The micro mouse senses the wall on the left or right, and follows it to wherever it leads until the center is reached.

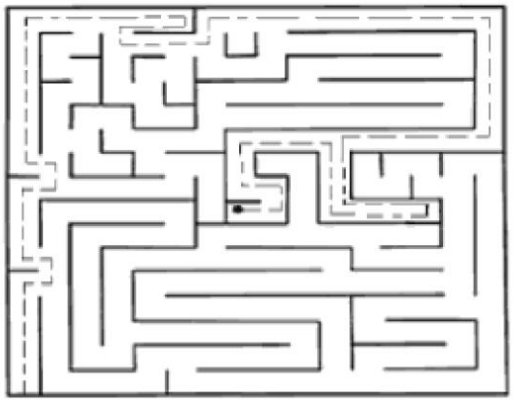

Fig4: Left wall follower: solvable maze

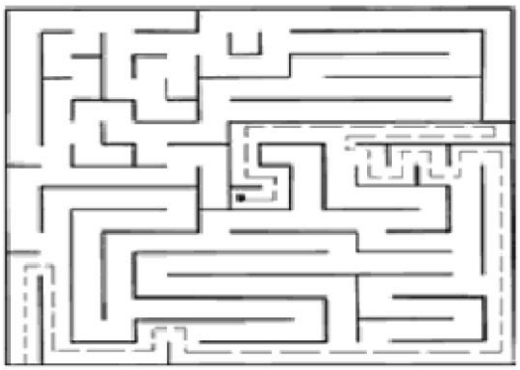

Fig5: Right wall follower: solvable maze

The time taken by the robot can be calculated. If we assume the cell to cell movement time to be 2.5 sec. and the turning time to be 1 sec. then the total time taken by the robot is (140×2.5) + (27×1) = 377 sec. for reaching the center of this maze.

The above maze is solved using the right wall follower logic. The total time taken by the robot is (124×2.5) + (25×1) = 335 sec. for reaching the center of the maze.

The problem with both the wall follower algorithms is that all the mazes cannot be solved.

IV. Simulation Result, Analysis And Comparison

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.

Maze Names	Total number of cell traversed to find best run		Total number of distance updates to find best run	
	Flood Fill	Modified Flood Fill	Flood Fill	Modified Flood Fill
IEEE Region 6 2012	530	530	135,936	3197
Sample 16×16 cell Maze	179	179	46,080	793
APEC 2002	322	322	82,688	3307
Seoul 2002	585	585	150,016	3887
Minos 2003	246	246	63,232	2643

Fig6: Comparison of different algorithm

	Left wall Follower	Right wall Follower	Flood fill Algorithm
Cell to cell movements	140	124	50
turns	27	25	16
Time taken (in sec.)	377	335	266

Fig7: Total Number of Distance Updates To Find The Best Runs

Theoretical investigation suggests that, Flood fill 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 Flood fill is designed to find the minimum path by traversing fewer amounts of cells.

V. Comparison And Decision

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:

In case of large mazes BFS is more appropriate than DFS.

- Memory requirement for coding is higher in DFS than BFS.
- 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.
- Following statements can be drawn to compare BFS and Flood fill.
- If the maze is large then Flood fill should be avoided. BFS will give satisfactory performance in this scenario.
- For small mazes (i.e. 8x8) Flood fill is suggested to use. It will find the shortest path quickly than BFS.

Compared to the Flood Fill algorithm, the Modified Flood Fill algorithm offers a significant reduction in cell distance updates (Fig7). With a lesser number of distances to update, the micro mouse which uses the modified flood fill can traverse from cell to cell in a higher speed. Furthermore, if the dead end cells are marked on the first arrival, these marked cells will not be explored in subsequent runs, resulting in a reduced total number of cells explore.

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. With less demand for computation speed and with the assurance that the best run gives the smallest number of cells traversed, the Modified Flood Fill algorithm may be the choice for micro mouse competitions.

References

[1]. Sadik A. M. J, Farid H. M. A. B., Rashid T. U., Syed A., Dhali M.A. "Performance analysis of micro mouse 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]. "Micro mouse 2010 Competition Rules." IEEE Region 2 Student Activities Conference 2010 Web Page. Web. 21 Nov. 2009.[http://www.temple.edu/students/ieee/SAC/com petitions.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.