Variable Priority in Maze-Solving Algorithms for Robot's Movement (Specially useful for Low Quality Mechanic Robots)

Babak Hosseini Kazerouni, Mona Behnam Moradi and Pooya Hosseini Kazerouni

Students of Electrical and Computer Engineering Department, Shahid Beheshti University
P.O. Box 19976-63913 Tehran, Iran
bkazerouni@yahoo.com
mona_b_82@yahoo.com

ABSTRACT: Most existing Maze-Solving algorithms assume a constant priority for the robot's movement. Thus, each moment, the robot will determine next movement only by the assumed constant priority. As turning would take a lot of time the fastest path and the easiest to go through is the one that has less turns. Because of the fastest path and in some projects the easiest one is preferred, and the constant priority might not lead the robot to the one with less turns, a movement priority that considers non-turning paths engenders less solving-time. In many projects the mechanic of the robot is not high quality so it should move through paths with less turns. In this paper a *"variable priority for robot's movement"*, is introduced. This variable priority besides causing less solving-time is useful in projects that a robot with low quality mechanic is used, as it cause less turning. So, it would increase the solving efficiency a lot. One more advantage is that this manner is general and not just for a specific maze. It is important because in today's projects, most of the time, the environment around the robot is not known.

KEYWORDS: Maze-Solving Algorithms, Open Path's Length, Less Turns, Variable Movement Priority.

1. INTRODUCTION

Today, it is tried to use robots in more project as substitution for human. In the most of these projects, robots should move or walk and find their paths, for example robots that are used in mines or robots that are sent to planets. Therefore, nowadays, the ability of robots to consciously find their way around the terrain plays a more important role in the human life. At the present time, a mazesolving robot, self-contained without using an energy source, is more important than it was in previous years. The speed of robot to find its path, affected by the applied algorithm, acts the main part in the present projects. In these cases, the main purpose is to find the fastest path not the shortest one. And in many projects the easiest path is preferred as the robot might not having a high-quality mechanic.
In this paper the suggested method is used to increases the solving speed. These days, a lot of maze-solving robotic competitions are held around the world to achieve faster and superior robots [1], [2], [3]. To test the new method, one of these competitions is used, called *"Micro Mouse"* and the method is tested in this kind of competition's mazes. Flood Fill algorithm is one of the best maze solving algorithms. So, the suggested method is compared with this algorithm. After overview the problem in section 2 of this paper, the flood fill algorithm with an example is introduced in section 3. Finally, in section 4 the new method is presented and in section 5 the results are stated.

2. PROBLEM OVERVIEW

There is a movement priority in path-finding algorithms. When the algorithm permits more than one way for the next movement, the next movement direction is chosen by this movement priority. This priority in the most of existing algorithms is constant and is not changing during the process. For instance, if the destination is on the Northeast of the start point the priority shall be North, East, South and then West; or something like that. Priority is important for some reasons. The most important reason is a bad selection could throw the robot in a path with many turns. It could cause wasting extra times to find the destination and for the robots with low-quality mechanic it may cause not finding destination. Less turning is desired because turning would take time and it needs highquality mechanic When robot avoids undesired turning it has more time to go straight. Therefore, it accelerates more and move faster.

Considering this reason make the new method, introduced in this paper, solve the maze faster.

3. FLOOD FILL ALGORITHM

In this section the *"Flood Fill"* algorithm is introduced. Most of the information in this section is taken from CSUN World Wide Web about Micro Mouse [4].
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 values for a 5X5 maze without walls would look like this:

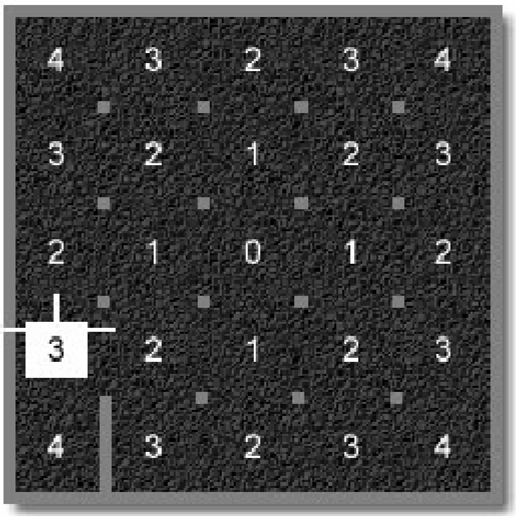

Figure 1. Flood Fill Algorithm Explanation

Of course for a full sized maze, you would have 16 rows by 16 columns = 256 cell values. Therefore you would need 256 bytes to store the distance values for a complete maze.
When it comes time to make a move, the robot must examine all adjacent cells which are not separated by walls and choose the one with the lowest distance value. In our example above, the mouse would ignore any cell to the West because there is a wall, and it would look at the distance values of the cells to the North, East and South since those are not separated by walls. The cell to the North has a value of 2, the cell to the East has a value of 2 and the cell to the South has a value of 4. The routine sorts the values to determine which cell has the lowest distance value. It turns out that both the North and East cells have a distance value of 2. That means that the mouse can go North or East and traverse the same number of cells on its way to the destination cell. As our movement priority, North, East, South then West, the mouse will choose to go to the North cell. Now the mouse has a way of getting to center in a maze with no walls. But real mazes have walls and these walls will affect the distance values in the maze so we need to keep track of them. Again, there are 256 cells in a real maze so another 256 bytes will be more than sufficient to keep track of the walls. There are 8 bits in the byte for a cell. The first 4 bits can represent the walls leaving you with another 4 bits for your own use. A typical cell byte can look like this:

Bit No.	7	6	5	4	3	2	1	0
Wall					W	S	E	N

So now we have a way of keeping track of the walls the mouse finds as it moves about the maze. But as new walls are found, the distance values of the cells are affected so we need a way of updating those. Returning to our example, suppose the mouse has found a wall.

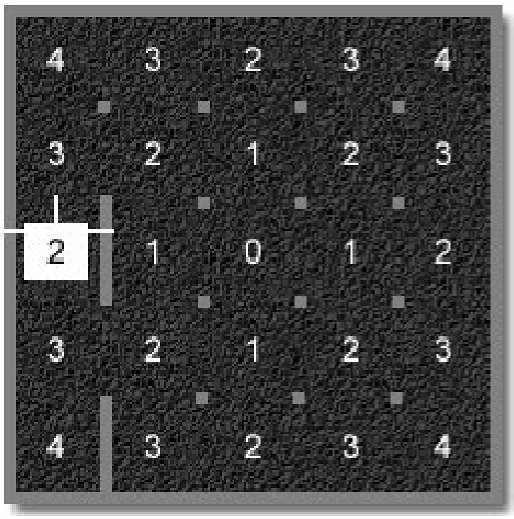

Figure 2. Flood Fill Algorithm Explanation

We cannot go West and we cannot go East, we can only travel North or South. But going North or South means going up in distance values which we do not want to do. So we need to update the cell values as a result of finding this new wall. So we add one to the minimum distance value of possible cells. Now the present cell's distance value is at least one more than the rounds so the robot will move to that cell. In above example, concern to movement priority, the robot moves to the North.

Sometimes the robot goes to a cell with walls all around it. We call it *"dead end"*. Now we "flood" the maze with new values. As an example of flooding the maze, let's say that our mouse has wandered around and found a few more walls. The routine would start by initializing the array holding the distance values and assigning a value of 0 to the destination cell.

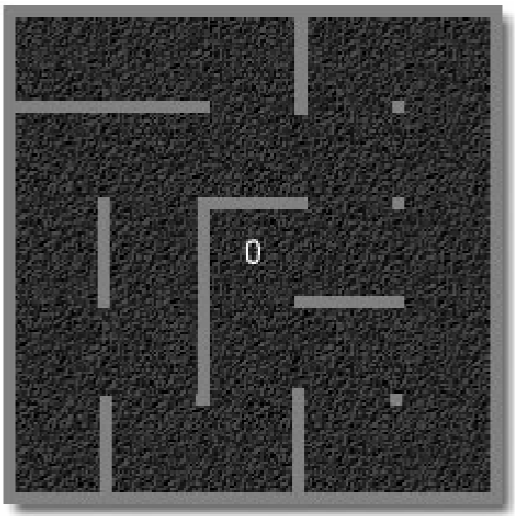

Figure 3. Flood Fill Algorithm Explanation

The routine then takes any open neighbors and assigns the next highest value, 1.

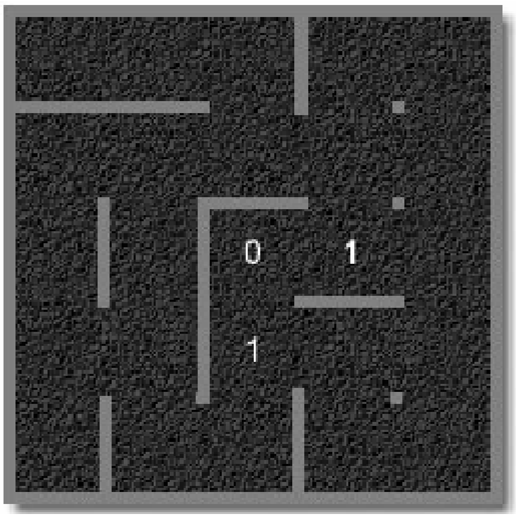

Figure 4. Flood Fill Algorithm Explanation

The routine again finds the open neighbors and assigns the next highest value, 2.

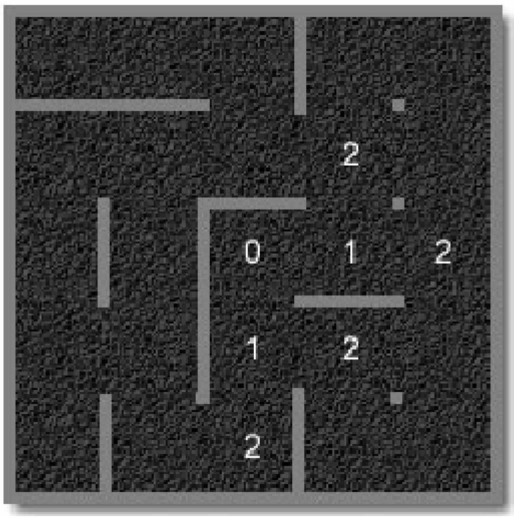

Figure 5. Flood Fill Algorithm Explanation
A few more iterations:

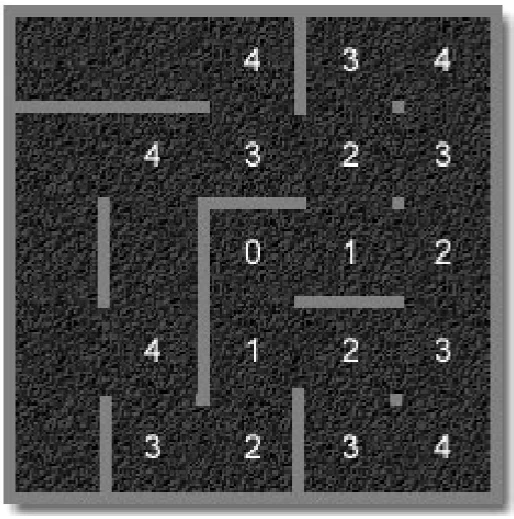

Figure 6. Flood Fill Algorithm Explanation

This is repeated as many times as necessary until all of the cells have a value. It is illustrated below.

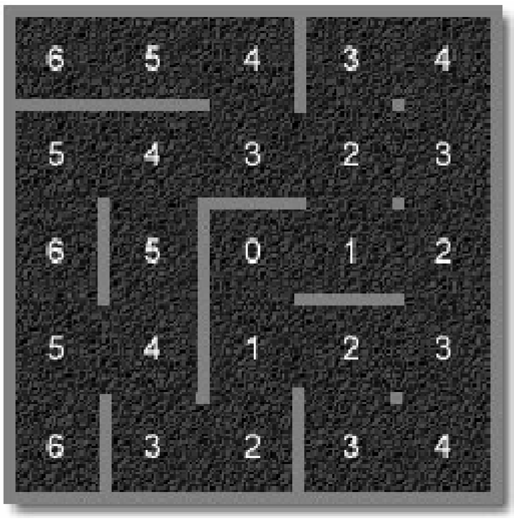

Figure 7. Flood Fill Algorithm Explanation

Notice how the values lead the mouse from the start cell to the destination cell through the shortest path.

In each cell the following steps are taken:

1. DETECT THE WALLS ROUND THE ROBOT AND SAVE THEM.
2. IS THERE ANY DEAD END?
3. IF THERE IS NOT ANY DEAD_ENDS WE SHOULD COMPARE THE CELL'S VALUE WITH ITS NEIGHBOURS TO DETERMINE WHETHER IT IS NECESSERY TO MAKE THE CELL'S VALUE PLUS ONE AND IF SO DO IT.
4. IF THERE IS DEAD_END WE SHOULD UPDATE.
5. NOW WE DETERMINE WHICH MOVEMENT SHOULD BE TAKEN.

4. THE PROPOSED METHOD

In the new method, in the part five of above division when the next movement should be determined, instead of constant priority we use a variable priority.
To define the variable priority, we placed four ultrasonic sensors around the robot; different applications may use different kinds of sensor. These sensors measure the length of the *"open paths"* in each side of the robot. Length of open path is the number of cells that exists in that direction without any obstacles. Higher priority is given to the direction with the longer open path. Because, in the longest open path, the robot has more time to move, accelerates more and spends less time to stop the chassis.
At the end of the execution of the Flood Fill algorithm in each cell of the maze, the final priority is selected for the next movement. At first, the Flood Fill algorithm shows which ways the robot is allowed to go through. As explained in the previous section, these ways are the ones that their distance value is fewer than the current cell's value. Each of these directions is given a digit. This digit shows the number of open path's cells in each direction. For example if the algorithm gives digit 3 to the East direction, it means that in the East direction there are three cells and then an obstacle. Then the algorithm compares these digits. So, the direction with the most open path's cells, is selected by the algorithm for the robot's next movement. This new method decrease turning a lot. Thus, it increase solving efficiency specially in such projects with the low-quality mechanic robot.

Therefore, in each cell, steps are as follows:

1. DETECT THE WALLS ROUND THE ROBOT AND SAVE THEM.
2. IS THERE ANY DEAD END?
3. IF THERE IS NOT ANY DEAD_ENDS WE SHOULD COMPARE THE CELL'S VALUE WITH ITS NEIGHBOURS TO DETERMINE WHETHER IT IS NECESSERY TO MAKE THE CELL'S VALUE PLUS ONE AND IF SO DO IT.
4. IF THERE IS DEAD_END WE SHOULD UPDATE.
5. MEASURE THE EACH DIRECTION'S OPEN PATH LENGTH.
6. COMPARE WITH THE OTHER DIGITS
7. NOW WE DETERMINE WHICH MOVEMENT SHOULD BE TAKEN (THAT IS THE ONE WITH MOST OPEN PATHS CELL).

5. RESULTS

In producing the result given in this section, a maze based on the APEC Micromouse competition rules [5] is assumed. It is just because a maze should be built for the test. But the features discussed in this paper are not dependent on the construction of the maze. Any environment in industrial and other applications could be modeled as a maze. The *"Flood Fill Algorithm"* with the old main priority is used as old method to solve the maze. And the flood fill algorithm beside the new variable priority is used as the new method. We assume that the robot goes from a cell to another in 2 seconds and the time for turning is 0.5 second for 90 degrees turn. It is assumed that the robot cannot accelerate. If the robot can accelerate the solving time decreases a lot. Since in this new method, the robot goes through ways with most open path's length, acceleration can increase its speed a lot. But because of variable acceleration for variable robots and simplicity in calculations we did not assume acceleration in this test.
We showed the results in two mazes. One of them is the maze that has been used in 1996 IEE World MicroMouse Championships, University of East London, 6th July 1996 and the other one is the one that has been used in the Micromouse competition, London (Wembly) Final, July 1981. Solving these mazes with the new method and old method is illustrated below. Finally, the time needed by the robot to solve the maze is calculated by numbers of turns and movements. It is obvious this new method would help the robot to solve faster and goes through paths with less turns.

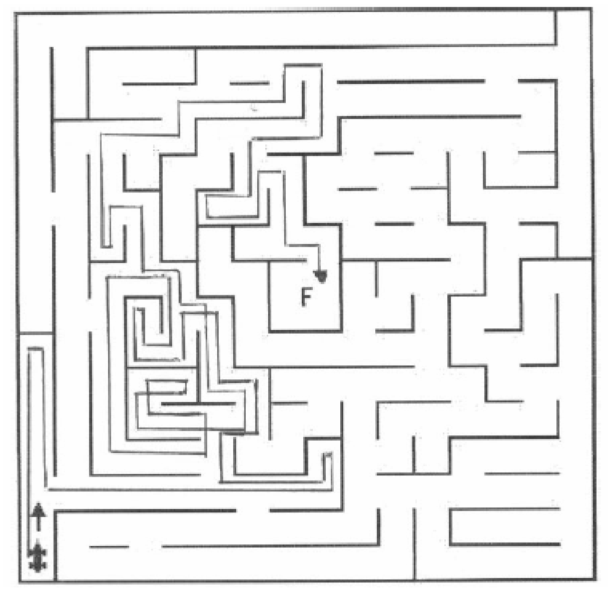
Figure 8. 1996 IEE World MicroMouse Championships solved by the old method

Solving time for this maze by the old method is 93*2 + 44*0.5= 208 sec

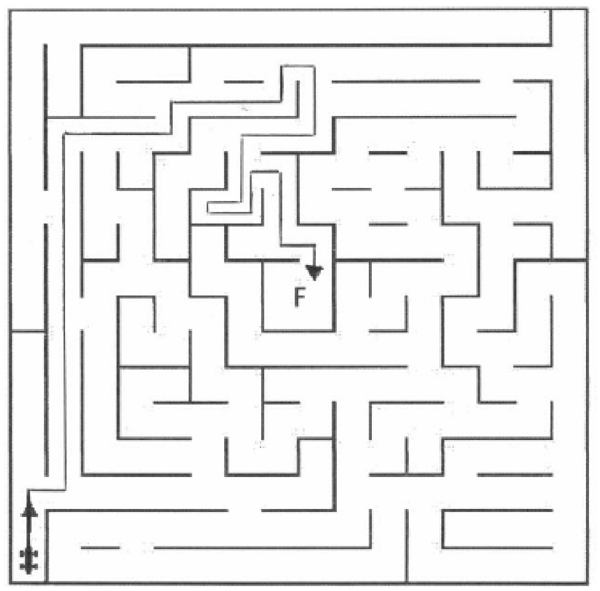
Figure 9. 1996 IEE World MicroMouse Championships solved by the new method

Solving time for this maze by the new method is 33*2 + 14*0.5= 73 sec

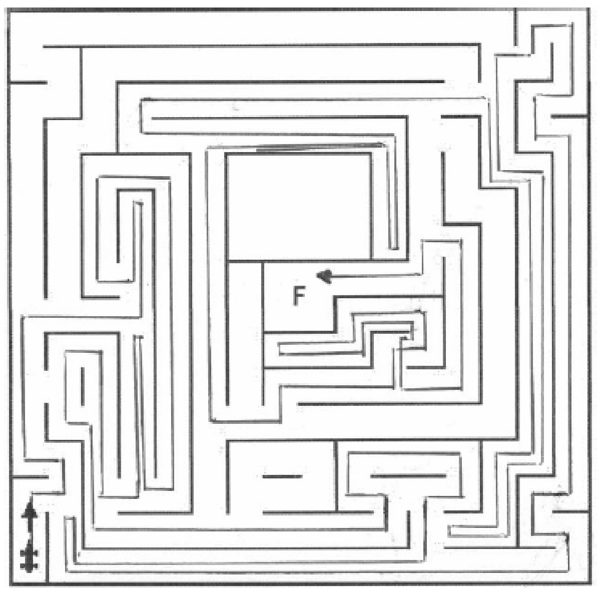
Figure 10. Micromouse competition, London (Wembly) Final solved by the old method

Solving time for this maze by the old method is 214*2 + 67*0.5= 461.5 sec

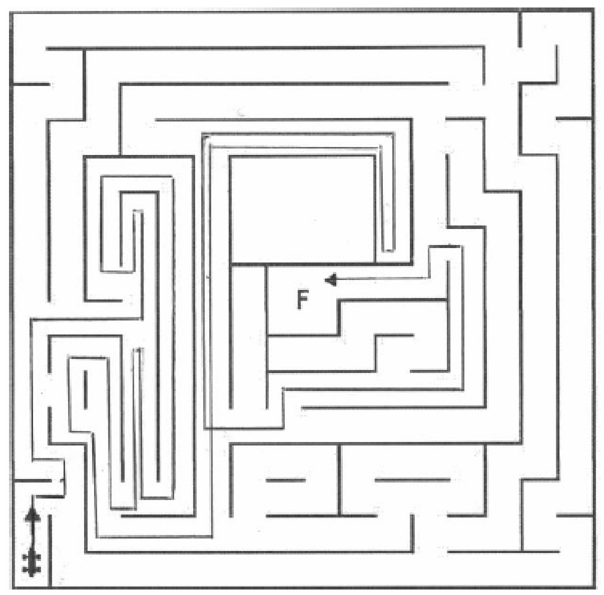
Figure 11. Micromouse competition, London (Wembly) Final solved by the new method

Solving time for this maze by the new method is 109*2 + 31*0.5= 233.5 sec

6. CONCLUSION

While there is no limitation to improve the algorithm, there are some restrictions on developing robot's mechanic or electronic. Developing algorithm is usually cheaper than the other parts. Therefore, Path-Finding algorithms, called *"Maze Solving Algorithms",* are the most important part in projects which a robot is used to find its path. The proposed method could not be good in some projects or could be very useful in other some but in general the variable priority would engender higher throughput and it is certainly improve the efficiency of the robot with low-quality mechanic.
Future works can be concentrated on considering using variable sensors and variable methods to determine some different strategies for *"variable priority for robot's movement"*.

7. REFERENCES

[1] *IEEE Micromouse Competition,* held since 1979.

[2] *APEC Micromouse Contest,* held each year in United States of America.

[3] *UK Micromouse contest,* held each year in United Kingdom

[4] MicroMouse, California State University at Northridge, http://homepage.mac.com/SBenkovic/MicroMouse/index.html

[5] APEC 2003 MicroMouse rules, http://www.apecconf.org/APEC_MicroMouse_Contest_Rules.html