Maze Solving Algorithm

Gan Zhen Ye , Dae-Ki Kang1)
Div. of Computer and Information Engineering, Dongseo University
Email : zy_gan@msn.com, dkkang@dongseo.ac.kr

ABSTRACT

Path finding and path planning is crucial in today's world where time is an extremely valuable element. It is easy to plan the optimum path to a destination if provided a map but the same cannot be said for an unknown and unexplored environment. It will surely be exhaustive to search and explore for paths to reach the destination, not to mention planning for the optimum path. This is very much similar to finding for an exit of a maze.

A very popular competition designed to tackle the maze solving ability of autonomous called Micromouse will be used as a guideline for us to design our maze. There are numerous ways one can think of to solve a maze such as Dijkstra's algorithm, flood fill algorithm, modified flood fill algorithm, partition-central algorithm [1], and potential maze solving algorithm [2]. We will analyze these algorithms from various aspects such as maze solving ability, computational complexity, and also feasibility to be implemented.

Keywords

Autonomous robots, Maze solving algorithm, Micromouse

I. Introduction

Path finding and path planning is crucial in today's world where time is an extremely valuable element. It is easy to plan the optimum path to a destination if provided a map but the same cannot be said for an unknown and unexplored environment. It will surely be exhaustive to search and explore for paths to reach the destination, not to mention planning for the optimum path. This is very much similar to finding for an exit of a maze.

A very popular competition designed to tackle the maze solving ability of autonomous called Micromouse will be used as a guideline for us to design our maze. There are numerous ways one can think of to solve a maze such as Dijkstra's algorithm, flood fill algorithm, modified flood fill algorithm, partition-central algorithm [1], and potential maze solving algorithm [2]. We will analyze these algorithms from various aspects such as maze solving ability, computational complexity, and also feasibility to be implemented.

II. Literature Review

2.1 Dijkstra's Algorithm

Dijkstra's algorithm finds the shortest path to solve the maze from a directed graph of a given set of nodes. The input of this algorithm consists of a weighted directed graph and the starting vertex. All vertexes in the graph are given a label and the edge of one vertex (vertex a) to another vertex (vertex b) holds the cost (edge(a,b)) of moving to vertex b from vertex a. This algorithm calculates the shortest path or minimum cost from the starting vertex to all other vertexes in the graph. The algorithm in steps is shown below:

```
Step 1: Start "Ready set" with starting node
        Set start distance to 0, dist[s] =0;
        Others to infinite: dist[i]= (for i s);
        Set Ready = { }.
Step 2: Select node with shortest distance from
the starting point that is not in Ready set
        Ready = Ready + {n}.
 Step 3: Compute distances to all of its neighbors.
For each neighbor node m of n
        Check if dist[n] +edge (n, m) < dist[m]
        If yes, dist[m] = dist[n] +edge (n, m);
Step 4: Store path predecessors.
        pre[m] = n;
Step 5: Add current node to "Ready set".
Step 6: Check if any node is left, if yes go to
Step 2
Step 7: end.
```

1) Corresponding author

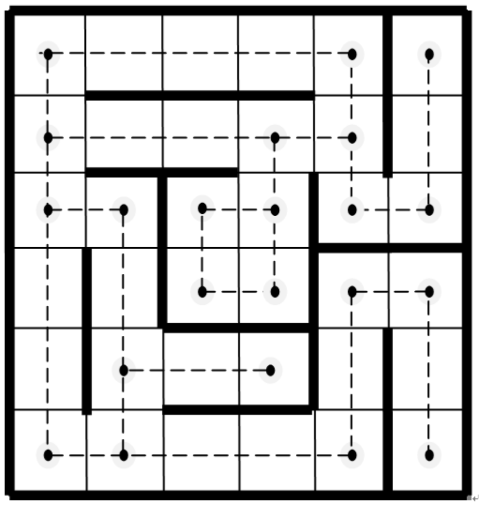

Fig. 1 Nodes perceived by Dijkstra's algorithm

2.2 Flood Fill Algorithm

The basic idea behind this algorithm is to imagine someone pouring water down from the starting cell of the maze. The water will eventually flood the whole maze except the exit of the maze itself. The solution is to follow the path where the water is decreasing in quantity. Each cell in the maze is assigned a value which indicates its distance from the destination cell and the goal will have value of 0. The solution of the maze is to follow the path of decreasing values from the starting position. The pseudo code to implement this method is as below:

```
Let variable Level = 0
Initialize the array [A] so that all values = 255
Place the destination cell in an array called stack1
Initialize a second array called stack2
Start:
Repeat the following instructions until stack1is empty:
{
  Remove a cell from stack1
  If DistanceValue(cell) = 255 then
    let DistanceValue(cell) = Level and
    place all open neighbours of cell into stack2
  End If
}
The array stack1is now empty.
Is the array stack2 empty?
No ->
{
  Level = Level +1,
  Let stack1= stack2,
  Initialize stack1,
  Go back to "Start:"
}
```

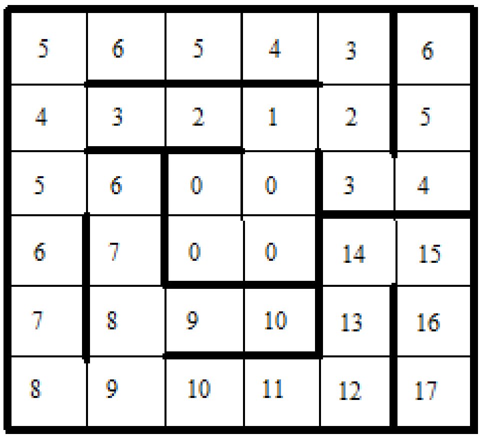

Fig. 2 Maze with flooded cell values

2.3 Modified Flood Fill Algorithm

This algorithm functions almost the same way with the normal flood fill algorithm. The difference is that the cell values are assigned manually before exploring the maze using the same manner of that in flood fill algorithm. The main difference of this method from flood fill algorithm this method only updates the cell values when necessary after each exploration and not all cells values are being updated every time. The algorithm will be explained in more detailed with the aid of the pseudo code below:

```
Creates a stack Stack1" and make sure that the stack is empty
Push current cell robot is in onto Stack1
Repeat till stack is empty:
    {
    Pop a cell from Stack1
    If distance value in this cell not equal to 1 + minimum value of its open neighbor and cell is not destination cell,
    Yes:
    Change cell value to 1 + minimum value of its open neighbor and Push all open neighbours of cell onto Stack1
    No:
    Do nothing
}
```

2.4 Partition-central Algorithm

This algorithm is proposed by Jianping Cai et al [8]. This proposed method is used to explore the maze to find the shortest solution path of the maze. This exploring method involves partitioning the maze into several parts and different navigating rules are applied to different part of the maze. The divisions of the map into several partitions can be viewed in Figure 3 below.

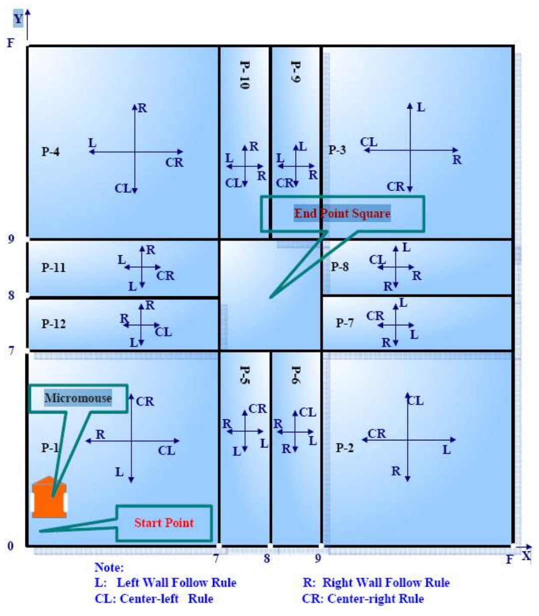

Fig. 3 Maze partition division

Referring to figure 3 above, the maze is divided into 12 partitions labeled in the diagram with P-1 to P-12. Take P-1 for example, if the mouse is in P-1 with its absolute direction heading north, it will follow the center-right rule. This means that whenever possible the mouse will tries to move forward along the path. When it reaches a T-junction, the mouse will choose to travel right instead of taking a left-turn. Once the mouse changes its direction it will then obey another set of rules based on its absolute direction and also the partition the mouse is located in the maze.

2.5 Potential Maze Solving Algorithm

A new way to solve a micromouse maze is proposed by Wyard and Meng [9]. The basic idea of this algorithm is somewhat similar to flood fill algorithm in the sense of assigning different values to different cells where the values represents distance from the goal position. The difference in this algorithm is that the values assigned are potential values obtained from the means of the sensor's receiver. The implementation method of this algorithm is explained in the form of flow chart in figure 4 below.

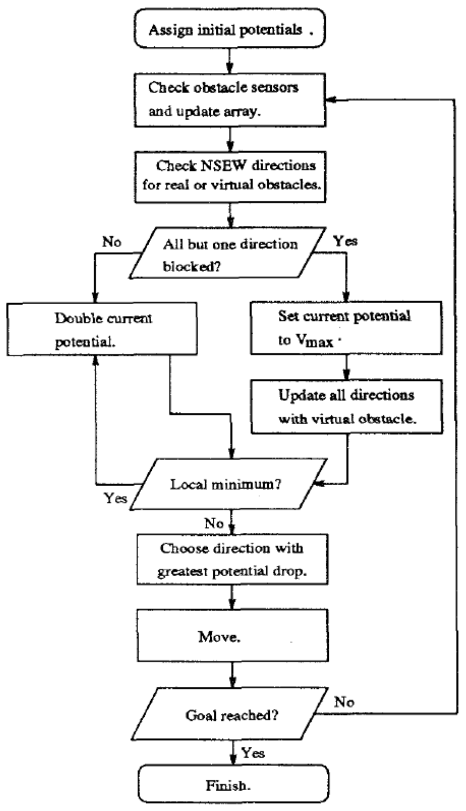
Assign initial potentials .
Check obstacle sensors and update array.
Check NSEW directions for real or virtual obstacles.
All but one direction blocked?
No
Yes
Double current potential.
Set current potential to V_{max}.
Update all directions with virtual obstacle.
Local minimum?
Yes
No
Choose direction with greatest potential drop.
Move.
Goal reached?
No
Yes
Finish.

Fig. 4 Maze solving flowchart using Potential Maze Solving Algorithm

III. Discussion

All algorithms reviewed are able to generate shortest path to the goal. However, some of it has certain drawbacks. Dijkstra's algorithm and Partition-central algorithm requires thorough exploration of the maze, which can be time consuming and lack of robustness. Some of the cells in the maze might not even be accessible. Furthermore, partition-central algorithm has a different set of rules to be applied depending on the mouse's location in the maze. This means that it requires much memory to store those rules and also the computation time to check which rules to be applied every time the mouse needs to make a decision Flood-fill algorithm can calculate the shortest path but each and every time the mouse moves into a new cell, the entire cell's value are required to be updated even if there might not be any changes of the cell's values. This is considered as a waste of time. This drawback is however solved in modified flood fill algorithm. As for the Potential maze solving algorithm, this method very much depends on the sensor's ability to detect accurately and also is limited by the sensor's sensing range. This method selects the path that allows the mouse to travel the furthest as its next direction. This might not necessarily leads the mouse closer to the goal.

IV. Conclusion

Having considered the drawbacks of each algorithm, we conclude that modified flood fill algorithm is the best algorithm to be applied in a Micromouse maze solving competition for its robustness and accuracy.

Acknowledgment

This research has been conducted in the project, "A study on HTML & Javascript engine analysis and their extension plan", which has been commissioned by Electronics and Telecommunications Research Institute's project "Development of collaboration service technologies among smart screens using dynamic relocation of Web fusion contents" in 2011.

Reference

[1] Jianping Cai, Xuting Wan, Meimei Huo, and Jianzhong Wu, "An Algorithm of Micromouse Maze Solving," in 2010 10th IEEE International Conference on Computer and Information Technology (CIT 2010), June 2010, pp. 1995-2000.

[2] Wyard-Scott and H.M. Meng, "A Potential Maze Solving Algorithm for a Micromouse Robot," in Proc. IEEE Pacific Rim Conference on Communications, Computers, and Signal Processing, May 1995, pp. 614-618.

[3] Swati Mishra, Pankaj Bande, "Maze Solving Algorithms for Micro Mouse," in Proc. of 2008 IEEE International Conference on Signal Image Technology and Internet Based Systems, Nov. 2008, pp. 86-93.