IRACST – Engineering Science and Technology: An International Journal (ESTIJ), ISSN: 2250-3498,
Vol.3, No.4, August 2013

Optimization of Flood fill algorithm using Look-Ahead Technique

Garima
Deptt. Of CSE
BPIT, Delhi, India

Vipul Aggarwal
Deptt. Of CSE
BPIT, Delhi, India

***Abstract*— In this paper further optimization of Flood fill algorithm is proposed. Our aim is to reach the center of the maze with shortest distance and in fastest time possible. With the present work, a Micromouse traverses a maze of unknown dimensions by travelling the shortest path, irrespective of looking the path that exists beyond it, thereby increasing time complexity. Combining Flood Fill algorithm with Look Ahead technique, time optimization is achieved by not allowing the Micromouse to travel those paths beyond which no further route(s) exist. A performance comparison has been done with normal flood fill execution, line follower algorithm, and look ahead alone. The results show that it can significantly improve the performance which in turn will help in improving the performance of flood fill algorithm.**

Keywords- ***flood-fill algorithm; flood-ahead algorithm, IR sensors.***

I. INTRODUCTION

The micromouse is a miniature robot designed to race inside a maze from a starting point to the destination. A maze is a tour puzzle in the form of a complex branching passage through which the solver must find a route. Along the way, there are walls and rooms with hedges that micromouse has to go through and decide the shortest and fastest way the reach the destination. The destination is at the centre of the maze. [7]
In order to make the micromouse navigate properly to reach the destination inside the maze, it must be equipped with at least sensors, microcontrollers and motors. The most important equipment inside the micromouse is controller that must be programmed with software to ensure micromouse making wise decision and navigating smoothly. What determines micromouse is making wise decision using software algorithm inside the microcontroller. Developing algorithm for the micromouse ultimately controls the autonomy positioning of the robot. [8] To accomplish this task, the micromouse needs to map the maze through intelligent exploration, and then track the optimal (shortest) path which will allow the mouse to run in the shortest possible time. [7]
MicroMouse applies the knowledge from different fields, such as computer science, computer engineering, electrical engineering, and mechanical engineering. It is a great opportunity to utilize interdisciplinary theoretical fundamentals to create a physical project. [7] The contributions to this paper are thus twofold. Firstly, to design and develop the optimizing techniques (time and distance) for Flood fill algorithm and secondly to integrate Look Ahead technique with flood-fill algorithm as it will help in optimization of distance traveled and time taken. This paper spans span across two major searching techniques, i.e. Look-Ahead technique and Flood-Fill algorithm. The concept has been coined as **Flood-Ahead Algorithm.**
This paper is organized as follows. In the first section, brief introduction about the motivations for the research and development of MicroMouse optimization is presented. In the second section literature review of earlier proposed techniques are shown. The third section comprises of the framework that we have proposed for our entire Micromouse system to optimize the time complexity. In section four, details about the dataset used i.e. ------ is explained. In section five and six the experiments conducted and subsequent performance evaluation are shown. Finally, at the end of this paper we conclude and give suggestions for future work in this field.

II. RELATED WORK

Many researchers have contributed in the field of Artificial Intelligence optimizing the flood fill algorithm. The core algorithm was given by Lee [1]. After analyzing it, we saw that it was a Breadth First Search (BFS) algorithm; an application to Dijkstra's algorithm. But its drawback was that it suffered from huge memory requirements and slow performance. Many other authors have contributed in this area. In [2], the authors tried to improvise the Ackers [3] algorithm in which a two bit encoding technique was used per cell instead of saving the real distance. The authors proceeded in two phases: The filling phase: here they filled the neighboring cells to the starting point (S) with a one; the next cells were marked with two (these values are the distance of each cell from the source). They repeated this till they reached the target point (T). Retrace phase: If the target was reached in step I, they traced their way back by going to the cell with value I-1. They repeated this till they reached the starting point S. But this algorithm did not provide viable results for every maze. Hence, lead to its drawback. Consequently, an update procedure is used to modify the last search without re-starting. This technique is known as the Modified Flood Fill algorithm [4], which again is widely used by micromouse contesters. In [5], the authors used three approaches primarily. The wall follower logic: This worked on the rule of following either left wall or right wall continuously until it lead to the center. The micro mouse sensed the wall on the left or right, and followed it to wherever it lead until the center was reached. This simple logic used for solving the maze. Mathematical approach: Here, they had an array of 16 rows and 16 columns. If the array was denoted by M, then each cell was represented by M[R][C]. They assumed the present position of array as M2 [R2] [C2], the previous position as M1 [R1] [C1], and the next position as M3 [R3][C3]. Now they proceeded as follows:

If R2-R1>0, then it is currently moving straight, upwards through the maze array.
If R2-R1<0, then it is currently moving straight, downwards through the maze array.
If C2-C1>0, then it is currently moving rightwards, through the maze array.
If C2-C1<0, then it is currently moving leftwards, through the maze array.

Depth-First algorithm: Depth- First search was an intuitive algorithm for searching a maze in which the mouse first started moving forward and randomly chose a path when it came to an intersection. If that path leads to a dead end, the mouse returned to the intersection and chose another path. This algorithm forced the mouse to explore each possible path within the maze, and by exploring every cell, the mouse eventually found the center. It was called "depth first" [6] because if the maze was considered a spanning tree, the full depth of one branch was searched before moving onto the next branch. The relative advantage of this method was that the micromouse always found the route. Unfortunately the major drawback was that the mouse did not necessarily find the shortest or quickest route and it wasted too much time exploring the entire maze.
Different researchers have tried in the area of optimization of various algorithms of micromouse. But if their technique were simple, the time complexity of the algorithm was too large and if the time complexity was minimal, then the technique used was not practically possible. Till date the best algorithm to execute the program of micromouse has been the flood fill algorithm. In our paper we have tried to further optimize the flood fill algorithm to provide better results. We have proposed a framework in which a cell looks ahead (for the cells with and without walls) every time it follows the flood-fill path. It helps the MicroMouse o eliminate those nodes where dead end exists and thereby optimizing the distance travelled and the time taken.

III. FRAMEWORK

It is divided into two phases:
A. Initial Framework
B. Proposed Framework

A. INITIAL FRAMEWORK

Here we throw light on the existing techniques and algorithms being used contemporarily, i.e. Flood Fill algorithm and Look Ahead algorithm.

1.) FLOOD FILL ALGORITHM

Flood fill, also called seed fill, is an algorithm that determines the area connected to a given node in a multi-dimensional array. It is based on Bellmen- Ford algorithm. The flood fill algorithm is one of the most popular algorithms used for maze solving. It combines both solution speed and easy handling of locations to give a quick solution to the maze concerned. The basic idea behind the algorithm is to imagine one pouring water down the destination cell and to follow the path of flow of the stream. The water will flow and cover all cells one by one until it reaches the destination cell. The solution is the path of water from the source to the destination. The algorithm works as follows:

1.1 Each cell is assigned a value which indicates its distance from the destination cell. Thus the destination cell is assigned a value 0.
1.2 Any open neighbor to the destination cell is assigned a value 1. Similarly, all the open neighbors to the current set of open neighbors are assigned a value 2 and so on.
1.3 The value of the cell therefore is an indication of how far we are from the destination. Any cell will have at least one adjacent cell holding a value one less than the current value.
1.4 To find the solution one has to simply follow the path of decreasing values from their current position, it is assured that they will reach the destination. This has been taken from [2].

Dynamic flood fill algorithm: It is the basic method to flood fills a maze. If we have all the wall information, then all we need to do is follow the numbers from the current value till machine reaches zero. Let's say we start from 10, then jump to the neighboring cell that has value 9, and then to 8 and so on. But when we start our machine for the first time it has no information of walls and its memory is completely clean. Assume there are no walls. Now flood fills the maze. Jump to the neighboring cell with one value less than the current cell. Use sensors to read the walls and update the maze. Again flood fill and jump to the neighboring cell with one value less than the current cell. Go on doing this till we reach the destination. By this method we need not follow the shortest path when we run the machine for the first time. Repeat the steps and come back to start position. Again follow these steps. But this time we have more wall information and so we will go by a different path (this too may not be the shortest). Eventually may be in 3rd - 4th run) we will have enough wall information (need not have all wall information) and we will follow shortest path.

2.) LOOK-AHEAD ALGORITHM

In backtracking algorithms, **look ahead** is the generic term for a sub-procedure that attempts to foresee the effects of choosing a branching variable to evaluate or one of its values. The two main aims of look-ahead are to choose a variable to evaluate next and the order of values to assign to it. Look Ahead algorithm incurs extra cost for assigning values as it needs to propagate constraints. Look ahead algorithm; helps to restrict the search space significantly as it remove values from future domains which does not suit the context. Look Ahead algorithm has no change on worst case performance but it can significantly reduce the amount of path traversed by the MicroMouse in best case.

B. PROPOSED FRAMEWORK

In our proposed framework we have combined Look Ahead algorithm with the existing Flood Fill algorithm to achieve optimization of distance and time. Modified version of Flood Fill algorithm helps us to put a constraint forward to check whether another variable can take a consistent value. In other words, it first extends the current partial solution with the tentative value for the considered variable. By combining the above two algorithms we aim to evaluate nodes ahead of the current node where MicroMouse has to (will) travel. In our framework, nodes with walls (sensed through IR sensors) ahead of them will be eliminated and it will lead to optimum path utilization.
Figure 1 indicates the working of flood-ahead algorithm. It shows that how the micromouse when kept at the starting of the maze reaches at the center using most optimum path and in minimum time. It uses the flood-ahead technique wherein each cell does not look only for its neighboring cell but also the cells of its neighbors to get the most optimum path.

ALGORITHM OF FLOOD-AHEAD

1. Set micromouse at the starting of the maze, say (X, Y).
2. Check if cell (X+1, Y) is blocked. If yes, then ignore this path.
3. If no, then check if the neighbors of this cell are blocked. If yes, then ignore this path.
4. If no, then go to step 14.
5. Check if cell (X, Y-1) is blocked. If yes, then ignore this path.
6. If no, then check if the neighbors of this cell are blocked. If yes, then ignore this path.
7. If no, then go to step 14.
8. Check if cell (X, Y-1) is blocked. If yes, then ignore this path.
9. If no, then check if the neighbors of this cell are blocked. If yes, then ignore this path.
10. If no, then go to step 14.
11. Check if cell (X, Y-1) is blocked. If yes, then ignore this path.
12. If no, then check if the neighbors of this cell are blocked. If yes, then ignore this path.
13. If no, then go to step 14.
14. Compare the values in the unblocked neighboring cells of (X, Y).
15. Move micromouse to the cell which has the minimum value and is closest to the centre i.e. (0, 0).
16. Repeat these steps until micromouse reaches at the centre of the maze.

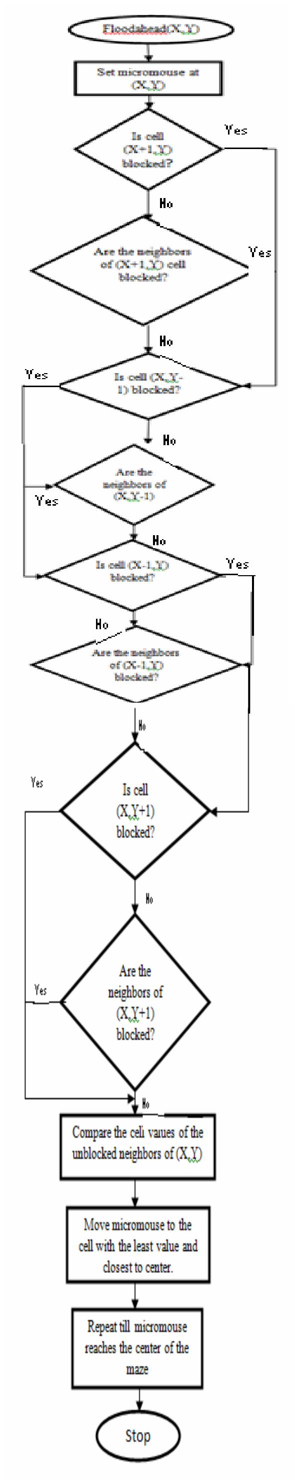

Floodahead(X,Y)
Set micromouse at (X,Y)
Is cell (X+1,Y) blocked?
Yes
No
Are the neighbors of (X+1,Y) cell blocked?
Yes
No
Is cell (X,Y-1) blocked?
Yes
No
Are the neighbors of (X,Y-1)
Yes
No
Is cell (X-1,Y) blocked?
Yes
No
Are the neighbors of (X-1,Y) blocked?
No
Is cell (X,Y+1) blocked?
Yes
No
Are the neighbors of (X,Y+1) blocked?
Yes
No
Compare the cell values of the unblocked neighbors of (X,Y)
Move micromouse to the cell with the least value and closest to center.
Repeat till micromouse reaches the center of the maze
Stop

Figure1: Flow chart of flood ahead algorithm [9]

IV. DATASET

A. MICROPROCESSOR

The electronic design centers on a Microchip processor. The PIC16F877 has 5 ports that make our interface with external hardwareseasier.PIC could be interfaced with external EEPROM memory to facilitate extensive programming. To keep the hardware small and compact, the inbuilt EEPROM code memory of 8k was used for programming and the data memory of 256 bytes were used for runtime memory map storage. Other data storage requirements are implemented on the 256 byte RAM. The processor is the only onboard programmable chip, other peripherals included a Schmitt trigger IC 74HC14N.the voltage levels from the sensors were a change from 1.45v to 0.25 volts when an obstacle was detected. The inverting Schmitt trigger was interface to bring the detecting signal to TTL logic.

B. STEPPER MOTOR

A defining characteristic of a stepper motor is that, under normal circumstances, it is made to take up and hold one of a number of fixed positions. Movement from one of these positions to another is only by stepping it through a series of intermediate positions. A typical stepper motor might have 200 such positions in a single, complete rotation of the drive shaft. With little effort, this can be increased, on the same motor, to 400 steps. [8]

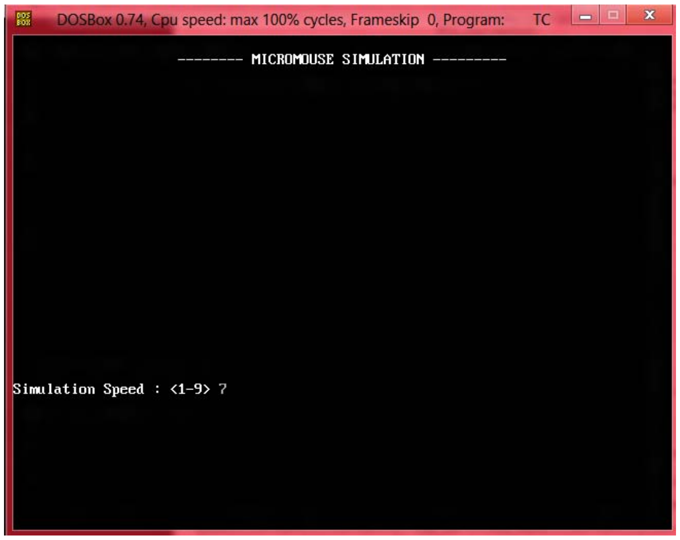
C. IR SENSORS

A defining characteristic of a stepper motor is that, under normal circumstances, it is made to take up and hold one of a number of fixed positions. Movement from one of these positions to another is only by stepping it through a series of intermediate positions. A typical stepper motor might have 200 such positions in a single, complete rotation of the drive shaft. With little effort, this can be increased, on the same motor, to 400 steps. [8]

V. PERFORMANCE EVALUATION

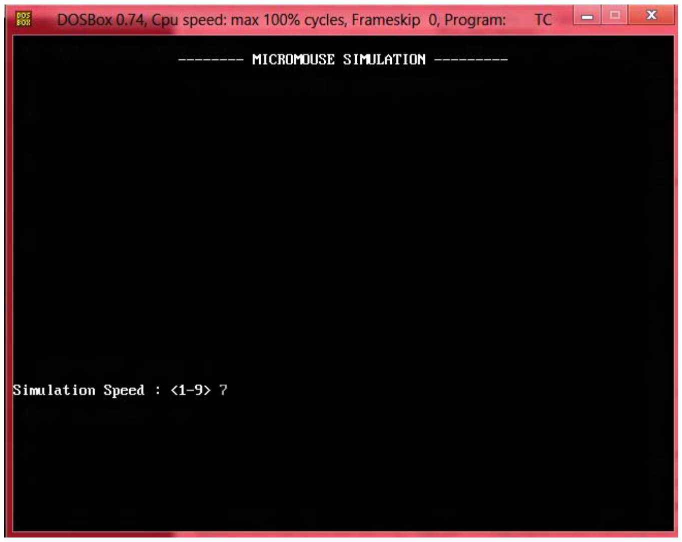
Performance evaluation is the key aspect of undertaking any research work. So we have evaluated our work and finally concluded by elaborating the efficiency of our work. But before discussing our execution, we first give an overview of the work done and then its relevant efficiency.
The existing framework makes the MicroMouse move towards centre from the starting point. The maze is number in an increasing order as we move from centre to the outside of the maze. The MicroMouse then observes the neighborhood cells and the number assigned to them to move forward in the maze. However, it does not intelligently scan the neighboring cells to move forward and hence produces ambiguity in obtaining the most optimal path.
Optimizing the given problem statement means applying and integrating the exiting techniques with other techniques to produce more optimal results. An optimized e of solution has been presented in this section.

The following snapshots are from the earlier execution of MicroMouse using flood-fill algorithm:


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

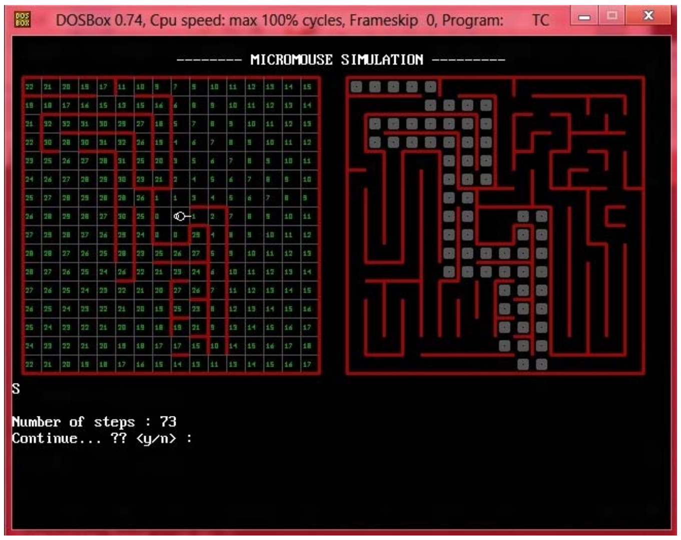
```
-------- MICROMOUSE SIMULATION ---------

Simulation Speed : <1-9> 7
```

Figure 2(a): Execution of flood fill


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

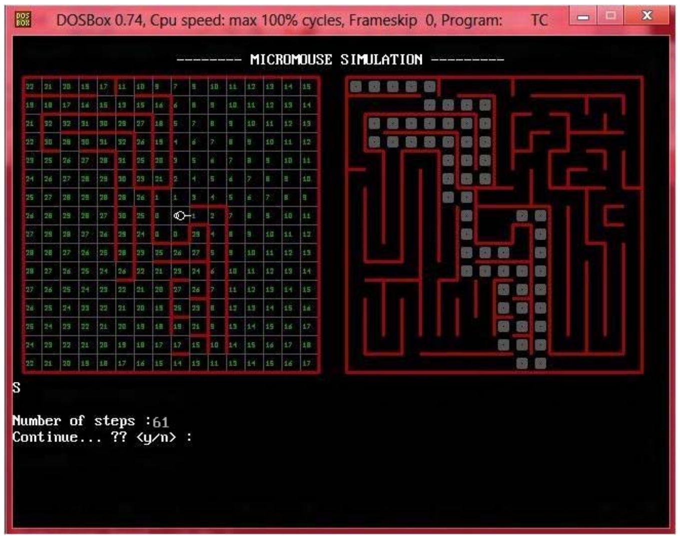
```
-------- MICROMOUSE SIMULATION ---------

Simulation Speed : <1-9> 7
```

Figure 3(a): Execution of flood ahead


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
-------- MICROMOUSE SIMULATION ---------
S
Number of steps : 73
Continue... ?? <y/n> :
```

Figure 2(b): Simulation of flood fill


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
-------- MICROMOUSE SIMULATION ---------
S
Number of steps :61
Continue... ?? <y/n> :
```

Figure 3(b): Simulation of flood ahead

In the above snapshots we can clearly see the ambiguities arising wherein the MicroMouse has to travel extra distance to reach the centre.

On applying flood-ahead algorithm, not only we helped in saving the distance travelled by MicroMouse but also saved the time to reach the centre (destination). The following screen shots show the execution of flood-ahead algorithm:

Clearly, flood-ahead algorithm results in lesser number of steps the MicroMouse has to travel. Also, as distance is directly proportional to time, MicroMouse will take lesser time to reach the destination, thereby reducing time complexity as well.
The working of flood-ahead algorithm in the above snapshot, in place of ambiguity is explained as follows:

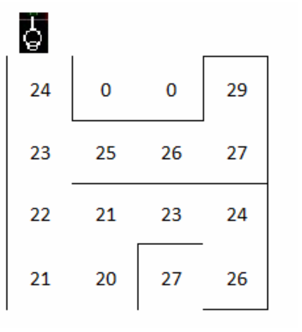


Figure 4: Ambiguity resolution through flood ahead

In the above figure as the MicroMouse travel from cell 23 to 25, i.e. towards the centre it follows the wrong path as there is a dead end ahead of it. In the proposed flood-ahead algorithm the MicroMouse will follow the natural steps of Flood-Filling algorithm but will travel lesser steps. The MicroMouse halts at cell 26 and look ahead for a clear path ahead. On receiving false Boolean value, as there is a wall (sensed by IR sensors) head of cell 29, it will not traverse the additional four steps it would have traversed earlier (26-27-29-27-26). Hence, saving the time and minimizing the distance the MicroMouse will have to travel. Flood-Ahead algorithm optimizes the distance, by eliminating the miscalculated steps; the MicroMouse has to travel to reach its destination.

VI. PERFORMANCE COMPARISON

Now the performance comparison between various execution techniques is given below for easy understanding:

A. LINE FOLLOWER AND FLOOD AHEAD

In line follower algorithm, the micromouse blindly moves forward in a straight line in the maze without using any intelligence. This algorithm has a large time complexity as the micromouse does not foresee what is coming in its way. Whereas, in flood-ahead algorithm, the micromouse intelligently and cautiously moves forward while calculating the minimum distance it has to traverse to reach the center

B. FLOOD FILL AND FLOOD AHEAD ALGORITHM

In flood fill algorithm, the micromouse intelligently moves forward without estimating whether there is a path ahead or not. It compares the cell values of the neighboring cells and moves to the cell with the least value. On the other hand, the flood-ahead algorithm not only compares the value of neighboring cells, but it also foresees whether there is path available ahead or not.

C. LOOK AHEAD AND FLOOD AHEAD ALGORITHM

Flood-ahead is an algorithm whereas, look ahead is a condition. According to the look-ahead condition, whenever a problem is given to the decision maker, he not only looks out for the possible steps which can be taken, but also the possible solutions in the second level of the problem. That is why it is called look ahead i.e., to look ahead for the best possible path. On the other hand, in flood ahead algorithm, the micromouse makes an intelligent and decisive choice by not only considering the second level solution of the problem, but also ensuring that the first level solution is the cell having the minimum value and at least one neighbor of that cell should not be blocked further and should be closer to the center of the maze.

The flood-ahead algorithm integrates the Flood fill algorithm and the look-ahead condition. Like in Flood fill algorithm, it compares the cell values of the neighboring cells. But before going any further, it estimates whether the path beyond the neighboring cells is available or not. It also uses the look ahead condition wherein its decision making also depends on the condition that which cell is closer to the center of the maze.

CONCLUSION AND FUTURE WORK

In this paper, we have proposed an efficient way to optimize the path micromouse has to travel to reach the center of the maze. The experimental results obtained in the above figures showed that the proposed work decreased the distance micromouse travelled. Also, the time taken by micromouse to reach the center was less compared to other algorithms. With the proposed work, the effectiveness of optimization has been studied carefully. Further investigation to the topic reveals that flood-ahead algorithm can give good results. The concept has been worked out and can be used in future with flood fill algorithm.

ACKNOWLEDGEMENT

We take this opportunity to express our sincere thanks and deep gratitude to all those who extended their wholehearted cooperation and have helped us in completing this work successfully. We express our sincere thanks to Mr. Suyash Gupta (DTU) for his encouragement and valuable suggestions

REFERENCES

[1] Lee, C.Y., "An Algorithm for Path Connection and its Applications", IRE Transactions on Electronics Computers, 1961.

[2] Ahmed N. Mosa Ayman M. Saad Moustafa A. Mohamed, "MicroMouse: An Intelligent Maze Solver Robot" IEEE Egypt section, September 2005.

[3] Akers, S.B., "A Modification of Lee's Path Connection Algorithm", IEEE Transactions of Electronics Computers, February 1967.

[4]http://www.micromouseinfo.com/introduction/mfloodfill.html[accessed on July 2013]

[5] Sandeep Yadav, Kamal Kumar Verma, Swetamadhab Mahanta, "The Maze Problem Solved by Micro mouse", International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249 – 8958, Volume-1, Issue-4, April 2012.

[6] Gorden Mc Comb, Myke Predko,“Robot Builder’s Bonanza”, Mc-Graw Hill, 2006 [accessed on July 2013]

[7] Work in Progress – Undergraduate Interdisciplinary Research – IEEE Micromouse Competition, Robert D. Milos, Allan J. Hall, Neal Shine, Kyle D. See, Tyler R. Bednarz, Yonglian Wang; Department of Electrical and Computer Engineering and Computer Science Ohio Northern University[accessed on July 2013]

[8]http://www.google.com/url?q=http%3A%2F%2Fmadan.wordpress.com%2F2006%2F07%2F24%2Fmicromouse-maze-solving-algorithm%2F&sa=D&sntz=1&usg=AFQjCNF0x7vp7y7STl2vZ-PVHlyu4f6fcQ

[9] http://en.wikipedia.org/wiki/Micromouse [accessed on July 2013]

AUTHORS PROFILE

Garima was born in 1991 in New Delhi, India. She has completed her B.Tech in Computer Science and Engineering from Bhagwan Parshuram Institute of Technology, IP University, Delhi, India. Her areas of interests include Artificial Intelligence, Digital forensics and Data mining.

Vipul Aggarwal was born in 1991 in New Delhi, India. He has completed her B.Tech in Computer Science and Engineering from Bhagwan Parshuram Institute of Technology, IP University, Delhi, India. His areas of interests include Artificial Intelligence, Data warehousing and C++Programming.