An improved algorithm of the exploring process in Micromouse Competition

Conference Paper · December 2010

DOI: 10.1109/ICICISYS.2010.5658360 · Source: IEEE Xplore

Li Xihua
Beihang University (BUAA)
5 PUBLICATIONS **8** CITATIONS
SEE PROFILE

Xiang Jia Xiang Jia Xiang jia
Fudan University
22 PUBLICATIONS **60** CITATIONS
SEE PROFILE

Xudan Xu
Ohio University
1 PUBLICATION **3** CITATIONS
SEE PROFILE

An improved algorithm of the exploring process in Micromouse Competition

——Dead-end exclusion algorithm

——Osmosis algorithm (Dead-zone exclusion algorithm)

Xihua Li, Xiang Jia, Xudan Xu, Jin Xiao, Haiyue Li
School of Automation Science and Electrical Engineering
Beihang University
Beijing, 100191, China
lixihua9@gmail.com, lhfy607@163.com

***Abstract*—This article discussed an algorithm for the Micromouse Competition, which is an international event in the field of artificial intelligence and is held by the International Institute of Electrical and Electronics Engineering (IEEE) every year. The discussion proposed "dead-end exclusion algorithm" to exclude one-line dead-end and "osmosis algorithm" to exclude dead-zone of any shape. The two improved algorithms were demonstrated in practical testing and competition. The key idea of the improved algorithms was to exclude dead-zone which did not contain the shortest path basing the obtained information. In this article, "osmosis idea" was used in maze algorithm for the first time, which realized the judgment and exclusion of the dead-zone effectively. Compared with classical algorithms, these improved algorithms had better temporal and spatial optimization result.**

Keywords-Micromouse; dead-end; dead-zone; osmosis

I. BACKGROUD INTRODUCTION

Micromouse Competition is an annual international event held by the International Institute of Electrical and Electronics Engineering (IEEE), which was launched by American Mechanical magazine in 1972. The Micromouse is a small microprocessor-controlled robot vehicle which can decode and navigate in a complex maze. Since it appeared, the competition has had a strong appeal to teachers and students all over the world. In 2007 and 2008, Shanghai Computer Society hosted the first and the second IEEE Standard Micromouse Competition Invitational (Yangtze River Delta region) in China, more than 30 institutions participated in the game. The 2009 Competition was expanded to the whole country, a total of 9 Divisions of 52 colleges and universities participated [1].

A. *Rules of Micromouse Competition* [2] [3]

The basic function of a Micromouse is to travel from the start grid to the destination grid. This is called a "run". The time it takes is called "the run time". While traveling from the destination grid back to the start grid is not considered a run. The total time from the first activation of the Micromouse until the start of each run is also measured. This is called the "maze time". If a mouse requires manual assistance at any time during the contest it is considered "being touched". The contest is scored according to the three parameters: speed, efficiency of maze solving, and reliability of the Micromouse.

The score of a Micromouse is got by computing a "handicapped time" for each run. The "handicapped time" means adding the "run time" to 1/30 of the "maze time" and subtracts 10 seconds from it if the Micromouse has not been touched. The run with the shortest "handicapped time" for each Micromouse shall be the final score of that Micromouse. Each competition Micromouse is given a period of 15 minutes in the maze. Within this time limit, the Micromouse could make as many runs as possible to record the situation of the maze, so that it could make the optimal run at last according to the memory before.

B. *Standards of Maze and Micromouse*

The maze is formed by 256 grids and each grid is 18 cm^2, and they are arranged in 16 rows×16 columns. Figure 1 is a photo of maze. Figure 2 is a Micromouse which uses the ARM7 processor LM3S615 as its main controller. Five sets of infrared distance sensors can sample the maze grid of around obstacles when fixed in accordance with a given frequency, getting the maze partition information.

Figure 1. The Maze (left)

Figure 2. The Micromouse (right)

C. *The Existing Algorithm*

The optimal solution of maze routing problem was the shortest path from the maze entrance (start grid) to the maze exit (destination grid). Classical algorithms of maze routing problem were breadth-first-search (BFS) and depth-first-search (DFS) algorithms [4]. But the above two algorithms required the exploring of the whole maze, the temporal and spatial complexity of classical algorithms would increase exponentially, which meant these algorithms were time-consuming. To solve these problems, the article made some improvement on classical algorithms. The key idea was to use obtained information to exclude maze grids that do not contain any information of the shortest path. The improved algorithms would not explore these maze grids in order to save time. Firstly, the article talked about "dead-end exclusion algorithm" to exclude one-line dead-end, and then "osmosis algorithm" was proposed to exclude dead-zone of any shape. In the article, the section 2 proposed the design thoughts and design flows for the two algorithms separately, and section 3 demonstrated the two algorithms by actual testing and analyzed the results. Conclusion was shown in the section 4.

II. ALGORITHM INSTRUCTIONS

A. *Dead-end Exclusion Algorithm*

1) The basic idea

This algorithm divided the maze into many maze grids first and the maze could be placed with virtual partitions, which was equivalent to an actual partition and just only could not be detected by sensors. Each maze partition (include the virtual partition and do not distinguish them later) would be marked on both sides, so that it was easy to be distinguished. Meanwhile, if Micromouse entered a grid that had been marked in three directions, it would return along the same way. The grid was useless for finding the shortest path, therefore the algorithm would mark the fourth direction of the grid, which meant closing the maze grid before the mouse entered it. So the Micromouse would not enter the grid, in order to shorten the time of exploring.

2) The Flow chart

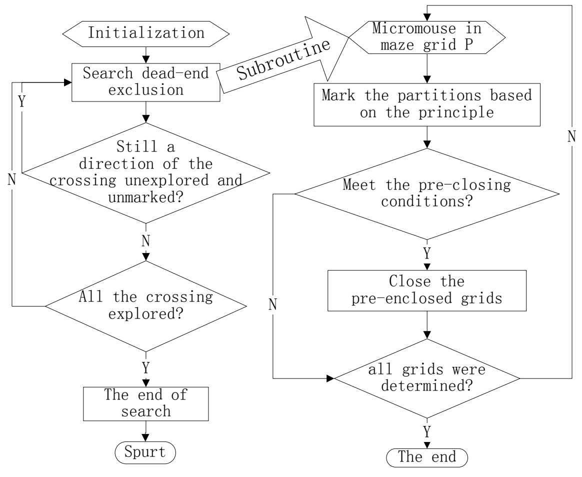

Initialization
Search dead-end exclusion
Subroutine
Still a direction of the crossing unexplored and unmarked?
Y
N
N
All the crossing explored?
Y
The end of search
Spurt
Micromouse in maze grid P
Mark the partitions based on the principle
Meet the pre-closing conditions?
Y
N
Close the pre-enclosed grids
all grids were determined?
N
Y
The end

Figure 3. Flow for the Dead-end Exclusion Algorithm

The flow chart for the algorithm was shown is figure3. It contained several modes as following:

Initialization: Marking the boundary. The maze grids did not have partitions by default, and other initialization such as for sensors and motors.

Maze exploring: Partitions that could be detected by sensors were marked by the reprogrammed rules as Micromouse moved on. And the nodes information and marking information were stored in iGucMapBlock[][], The Micromouse judged the marking information of the maze, if it met the closing conditions, closed it.

Closing conditions: If the grid Q was three directions marked and was not the destination, and Micromouse was not in the pre-enclosed area, then closed it.

Partitions marking rules: As shown in Figure 4 below. First, all maze inner borderlines would be marked during initializing as the red short dash lines showed. Second, as the Micromouse moved on, if sensors detected obstacle in one direction, the opposite direction of the adjacent maze grid would be marked. Third, if one grid met the closing conditions, it would be marked on the fourth direction and also the opposite direction of the adjacent maze grid would be marked.

3) The Diagrams

To further illustrate how “Dead-end exclusion algorithm” worked in actual process of exploring, we did the following explanation as shown in Figure 4.

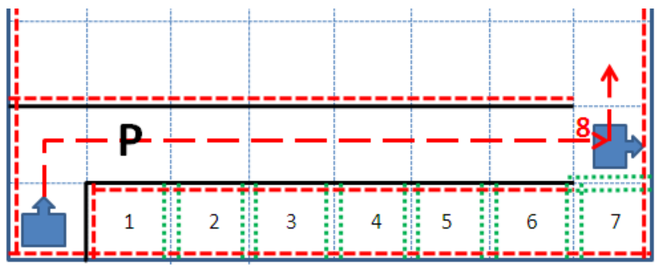


Figure 4. A Specific Example for Dead-end Exclusion Algorithm

As shown above, the red long dash line showed the path of Micromouse when exploring the maze, and the red short dash lines represented makers of detected partitions. The green dotted lines represented markers of virtual partitions when it needed to close a maze grid.

Step1: When Micromouse moved to P grid, the top of grid 1 was marked (as the partitions marking rules described), and the left had been marked when departure, so grid 1 is marked in three directions (the below was marked when initializing), satisfying the closing condition. So it would be marked on the right, as the green dotted line showed. Grid 1 would be closed and at the same time grid 2 would be marked on the left.

Step2: Continued along the red long dash line. Grid 2 also had three directions marked, so closed it. In the same way, grid 3, 4, 5, 6, 7 would be closed and grid 8 would be marked on the below.

Step3: When Micromouse moved to grid 8, the right sensors did not detect obstacle, but the right of grid 8 was marked in step 2 (marked on the below), so it would turn left. The time of exploring the maze grids 1, 2, 3, 4, 5, 6, 7 were reduced in this way.

B. *Osmosis Algorithm (Dead-zone Exclusion Algorithm)*

1) *The basic idea*

a) *Exploration to new ideas*

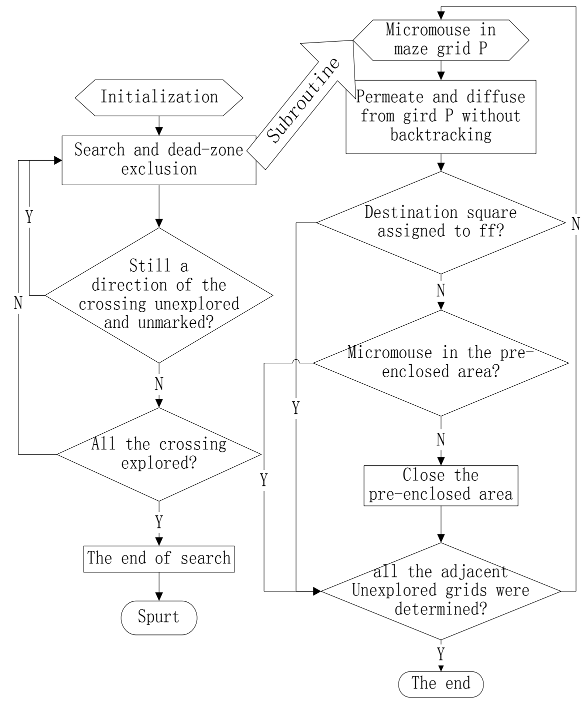
The "dead-end exclusion algorithm" realized exclusion of one-line dead-end with the obtained maze information, but it could not exclude dead-zone with rows and columns. And the algorithm for marking the partitions needed complicated looping statements, which would affect the speed of information processing and stability of movement. Similar to the "dead-end exclusion algorithm", osmosis algorithm (dead-zone exclusion algorithm) used obtained maze information to exclude dead-zone that would not contain the shortest path. However, it was difficult to realize continuous closing of maze grids in space and time. Here we introduced a new idea to solve this problem.

b) *Osmosis* [5]

When an area had only one exit and the destination grid was not in it, the area was called dead-zone. Osmosis algorithm inherited the dead-end exclusion algorithm's marking thought. It analyzed the obtained partition information and marked the only exit with a virtual partition. The key point was how to discriminate dead-zones. In chemic, "penetration" meant that low concentration solution spread into the high concentration solution through a semi-permeable membrane. This idea was applied in the osmosis algorithm. It was assumed that, there was a "semi-permeable membrane" at the exit of a single export area, namely dead-zone. And the maze was filled with "high concentration solution". Whenever met a crossing, Micromouse would make "low concentration solution" penetrate into the "high concentration solution" in the intersection direction. Then "high concentration solution" became low concentration. If " low concentration solution" could not eventually spread into the destination grid, the area would be a dead-zone. Then the crossroad of this area would be marked with a virtual partition so as to exclude the dead-zone. Here the closed area was an unexplored area, in case Micromouse was locked in the explored area.

2) *Flow chart*

Figure 5 was the flow chart for the osmosis algorithm. In the initialization, all unexplored maze grids were assigned to 0, after "low concentration solution" penetrated into the region, the values changed into 0xff. "Solution diffusion" was to assign the maze grids to 0xff, which were next to the assigned maze grids and with no virtual partitions between them. Continued this assignment until the entire region was assigned to 0xff and could not go on. Hereunder was the explanation of the algorithm.



Initialization
Search and dead-zone exclusion
Still a direction of the crossing unexplored and unmarked?
All the crossing explored?
The end of search
Spurt
Subroutine
Micromouse in maze grid P
Permeate and diffuse from gird P without backtracking
Destination square assigned to ff?
Micromouse in the pre-enclosed area?
Close the pre-enclosed area
all the adjacent Unexplored grids were determined?
The end
Y
N

Figure 5. Flow for the osmosis algorithm

Initialization: Marking the boundary. The maze grids did not have partitions by default, and other initialization such as for sensors and motors.

Maze exploring: Partitions that could be detected by sensors were marked by the reprogrammed rules as Micromouse moved on. And the nodes information and marking information were stored in MapBlock[][], The Micromouse judged the marking information of the maze, if it met the closing conditions, closed it.

Partitions marking rules: It was the same to the first improved algorithm.

3) *The Diagrams*

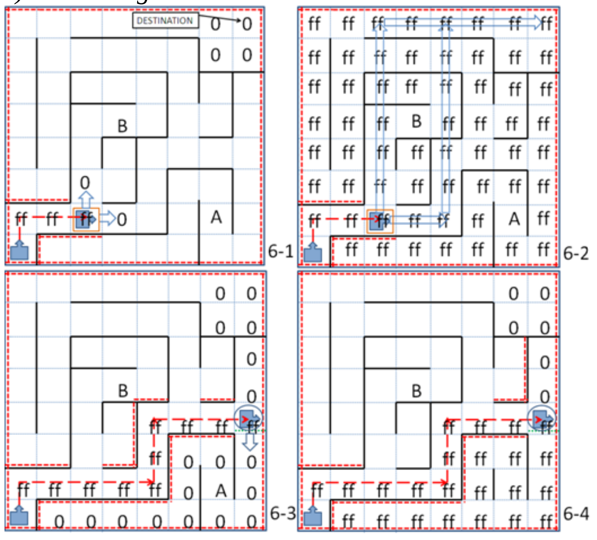


Figure 6. A Specific Example for Osmosis algorithm

As the specific example showed in Figure 6, below was a further explanation to the osmosis algorithm.

The maze grids were assigned to 0 by default, blue wide arrows indicated the penetrating direction of "ff". The red dash lines were the pre-marked virtual partition, while the green dotted line marked the exit of the dead-zone with virtual partition.

Step1: In the initialization, the maze border was marked with virtual partition. Micromouse ran along the red long dash line. The passed maze grids were automatically assigned to "ff". When it moved into the circled maze grid in Figure 6-1, the value of "ff" successively penetrated upwards and rightwards, as shown in Figure 6-2. Two blue arrows indicated a permeability path in which the destination grid was assigned to ff.

Step2: Because after the penetration and diffusion, the destination grid was assigned to "ff" eventually, it indicated the two directions were not the exits of dead-zones. Without closing them, Micromouse moved forward.

Step3: When Micromouse moved into the circled maze grid in Figure 6-3, it found the border of the region A below was being pre-marked with virtual partitions, and the maze grid below was assigned to 0, "ff" would penetrate into it.

Step4: In Figure 6-4, in the region A, "ff" continued to diffuse until the entire region was covered. As there was only one exit in the entire region and it could not return, "ff" could not diffuse to the destination grid, so the region A was a dead-zone. Mark a virtual partition below the maze grid where the Micromouse was in Figure 6-4.

Step5: Micromouse turned left, reducing the time of exploring the whole region A.

III. ALGORITHM DEMONSTRATION

To demonstrate the algorithms above, the testing was done basing on the three kinds of mazes as shown in Figure 7. As below, 3 groups of testing were made for the above two improved algorithms and the classical one. The measured data were listed in the table 1.

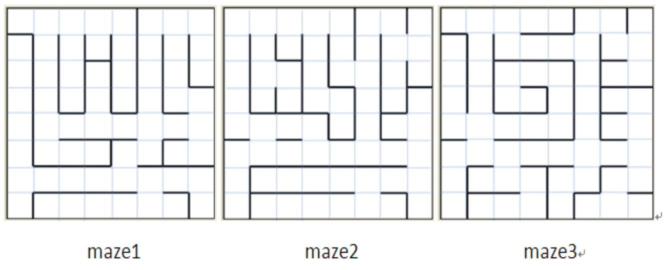
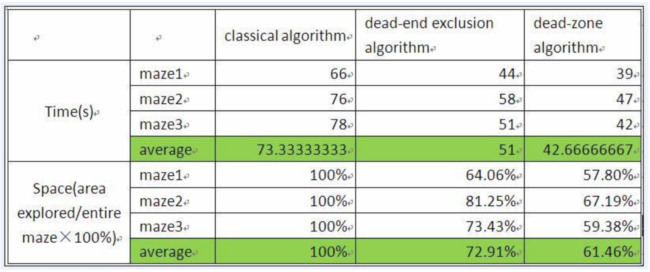


Figure 7. Three Kinds of Maze



		classical algorithm	dead-end exclusion algorithm	dead-zone algorithm
Time(s)	maze1	66	44	39
	maze2	76	58	47
	maze3	78	51	42
	average	73.33333333	51	42.66666667
Space(area explored/entire maze×100%)	maze1	100%	64.06%	57.80%
	maze2	100%	81.25%	67.19%
	maze3	100%	73.43%	59.38%
	average	100%	72.91%	61.46%

TABLE I. MEASURED DATA

Although the three mazes were artificially placed at random, but there was considerable universality. As can be seen from Table 1:

1) *Exploration time:* The classical algorithm> dead-end exclusion algorithm> osmosis algorithm. Dead-end exclusion algorithm and osmosis algorithm took 69% and 58% separately of the time that the classical algorithm took.

2) *Space explored:* The classical algorithm> dead-end exclusion algorithm> osmosis algorithm. Dead-end exclusion algorithm and osmosis algorithm each explored 73% and 61% of the space that the classical algorithm explored.

3) *In time and space:* Dead-end exclusion algorithm and osmosis algorithm were both optimized in comparison with the classical algorithm, the former to level of 30%, the latter to 40%.

To visually reflect the effects of the space optimization, we used the data of maze 3 to make a comparison, in which the blue areas and the yellow areas represented the unexplored areas when using dead-end algorithm and osmosis algorithm, the optimization effect was clearly evident.

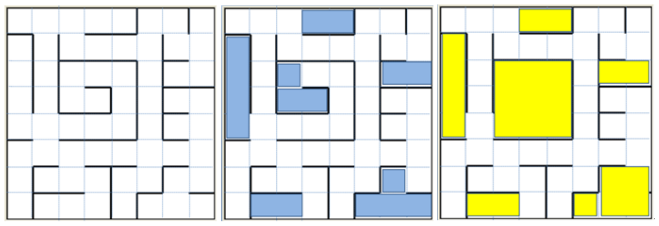


Figure 8. Unexplored areas For Each Algorithm

At the same time, we also discovered that as a result of dead-ends' exclusion, the nodes and the branches of the nodes that the procedure recorded decreased, then caused the micro controller's computation load and the memory utilization ratio reduces, also reduced the procedure complexity and the micro controller's performance requirement.

IV. CONCLUSION

This article began with the classical maze algorithms and then leaded to the optimization of maze algorithm. First, it proposed "dead-end exclusion algorithm" to exclude one-line dead-end and then "osmosis" to exclude dead-zone of any shape. "Osmosis" found a more intuitive way to describe the dead-zone, reducing time and space complexity during the judgment of dead-zone and reducing the pretreatment time of maze information. Finally it realized the exclusion of the dead-zone perfectly. The key idea of the improved algorithms were to exclude dead-zone basing on obtained maze information. As long as there was a dead-zone, it could be improved in both time and space. If the dead-zone algorithm was combined with other algorithms (such as the A* algorithm, clustering algorithm, genetic algorithm), it just changed the way of getting maze information, the exclusion of dead-zone would not be affected. The optimization effects would be more apparent if it was realized. In addition, with the improvement of the new algorithms, maze zone that had more than one exit but did not contain the shortest path information could be excluded with the obtained maze information. This would be our main job during the next period of time.

ACKNOWLEDGMENT

First, it needs to thank the Shanghai Computer Society who hosted the 3th IEEE Standard Micromouse Competition Invitational of China. Second, we give our thanks to the ZLG-MCU Company who gave so much capital support of this competition. Third, thanks for all the people who helped us during the competition and in writing this article.

REFERENCES

[1] Ligong Zhou; “Developer's Guide of IEEE Micromouse Competition”; 2009.

[2] http://www.ieee.uc.edu/main/files/sac2007/mm_rules.pdf

[3] http://micromouse.com.cn/index.htm

[4] Shoujiang Xu; “A Summary of the Maze Algorithm”; China Computer & Communication; 2009.

[5] http://hyperphysics.phy-astr.gsu.edu/hbase/kinetic/diffus.html

[6] Haoqiang Tan; “C Programming (the 3th Edition)”; 2005.