Advances in Computer Science Research, volume 88

International Conference on Computer, Network, Communication and Information Systems (CNCI 2019)

A Novel Maze Representation Approach for Finding Filled Path of a Mobile Robot

Changgang Zheng[1, a], Han Liu[1, b], Mengyu Ge[1, c] and Yilin Liu[1, d]

School of Glasgow, University of Electronic Science and Technology of China, Chengdu, China

[a]changgangzheng@std.uestc.edu.cn, [b]arnold981017@gmail.com,

[c]lomo.gmy@gmail.com, [d]lucy131421@gmail.com

Abstract. Finding a way in a given maze is a common problem for mobile robots. Many routing algorithms are presented to solve the maze, but they usually propose a theoretical algorithm which can not be well applied to realistic problem. The biggest obstacle to practical use is to build the model in accordance with the true scale of the robot and the maze. In this study, we present a new path routing approach for mobile robots to get an intuitive filled path, which can solve the difficulty of modelling and be well applied to realistic problem. This approach is based on an optimized representation method of the maze. Specifically, we encoded adjacent pixels of same row into a segment, then use four basic elements in a column of a coded matrix to represent the segment information. Finally, an A* like algorithm is applied to the coded matrix to obtain filled path. The proposed method has a high compression rate and is capable to form a walkable domain for robots instead of a single shortest path. Also, we compared this algorithm with many classical maze coding methods and routing algorithms, the result shows the effectiveness and efficiency of our approach in compression rate and maze-solving time. Furthermore, many practical engineering fields can utilize this approach where it is a priority to find a walkable path in a relatively short time.

Keywords: Path finding, representation method, routing algorithm, filled path.

1. Introduction

The objective of motion planning for mobile robots is to find a walkable path where the collisions between robots and obstacles are rare. When robots are designed to search for a walkable path in maze, firstly a modeling about maze and robot is required. A two-dimensional planform constructed in accordance with the true scale of the robot and the maze needs to be produced. Then some image preprocessing algorithms will be applied to the planform, lastly the searching algorithm is applied to the processed planform to find the walkable path. An efficient searching algorithm can soon find the desired path and avoid collisions with obstacles at the most extent. Many searching algorithms are put forward to solve the maze, but they usually ignore the first step or just propose a theoretical algorithm which can not be well applied to realistic problem. The biggest obstacle of these algorithms is the difficulty of modelling. Generally, the width of the maze and the size of the robot are different in different situations, so there is a need of a new model for each situation. Also, these conventional searching algorithms will always find the shortest path which do not consider the possible collisions with walls. In our approach, we generalize the model to well applied to a wide range of situations, which can find a walkable domain for robots without colliding with the obstacles. Generally, searching algorithm is not related to the image preprocessing, but our searching algorithm is based on the image preprocessing or a new representation of maze. A well compressive representation of the maze is necessary for robots, as usually the memory of MCU (Microcontroller Unit) in robots is limited while the data amount of maze can be huge. Also, in real world, the shape of the maze varies greatly, so the representation method should be applicable for different kinds of situations. In this paper, we present a new representation method, which reduces the required memory and helps the following searching procedure. We have compared our approach with several data compression methods in their compression rate. Also, we have compared our searching algorithm with other

searching methods and analyzed their strengths and weaknesses. In modern society, accidents like conflagration, explosion and collapse happen frequently. To find injured people and offer assistance in time, the detection and searching function of robots plays a significant role. And the accident scene can be regarded as a maze scenario, which means the core task of the robot is to find a viable path inside the maze model and operate commands. In this case, our method can be highly beneficial with highly effective compression because of the complexity of real accident scene.

1.1 Compression Method

For compressing the required memory, some methods are used to simplify the maze. Usually there are two ways: The first way is to gather the closed similar pixels as one point; The second way is doing the down sampling to reduce the image size.

For gathering the closed and similar pixels, Kambhampati and Larry S. Da [1] proposed Quadtree-related method which used squares with different sizes to represent the free space. In reality, lots of roads contain much curvilinear figure and in this situation the complexity of algorithm will increase significantly.

For doing the down sampling, Thorpe et al. [2] in 1984 computed Grid search method, laying grids on the maze picture and using the grids in path area to represent free space. However, since the grids are fixed, the cumulative error is the main shortcoming and the path could only be shortened by reducing the margin of grid which will increase the complexity a lot. Achour et al. [3], in 2011 proposed a more flexible method which used uniform random sampling to represent the free space.

Compression method is strongly related with the result of the final searched path. For the first method, the searched path is partly filled. For example, the path will be the connection of squares if using the quadtree compression method. The second method is compressing the maze with distortion, and it still uses individual pixels to represent the maze. The path it finds after applying the searching method would be a line. Nowadays, compression is not only used for reducing the memory, but also used to support the searching method. Our approach will code the maze with a high compression rate with no distortion. It will help to find a filled path to avoid the complexity of modelling.

1.2 Searching Method

Searching method is mainly divided into three types, which are depth-first, breadth-first and neural network algorithm. They can also be categorized according to whether it can find optimal path or not.

For the breadth-first method, Lee's algorithm [4] is proposed long ago and widely used in computer-aided design systems. For the problem of inefficiency, plenty of work was done to improve this classical algorithm (Jeffrey H. Hoel, 1976 and Mercedes et al,. 1997 [4])

Depth First and Breadth first search (Moore et al. [5], 1961) starts at an arbitrarily chosen root node and explore about node which has not been extended. The depth of the searching consistently increases before backtracking. In contrast, nodes neighboring previous ones are extended in Breadth first search (BFS). Thus all the nodes in same depth level are explored. Based on that, A* (Hart et al. [4] [7]) is a heuristic searching algorithm which evaluates adjoint nodes at current location and selects optimal one as the root node for next search. A* always follows the node who has the smallest sum of the previous moving steps from starting point and straight-line distance to the ending point. Preventing repetitive search, nodes that have been extended are recorded.

Branch and Bound(Land et al. [8], 1960) is an algorithm for optimization problem. This algorithm compares one candidate solution with other branches, evaluating the upper and lower bounds of the problem. Solutions that unable to produce better result are eliminated. For the path searching problem that a set of solutions have been found, Branch and Bound evaluates cumulative path length of candidate path and filters out the one with minimum value.

Reinforcement learning is another favored method to solve such kind of question in recent years, such as on-policy SARSA learning (Sutton and Barto 1998 [9]) which is firstly motivated by findings in the midbrain dopaminergic system (Morris et al [10]. 2006), and Q-learning will be discussed in detail in experiment section.

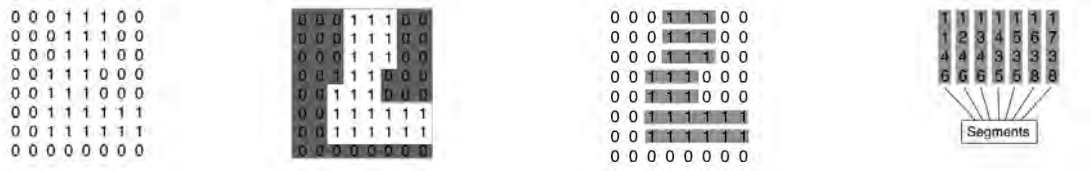
2. Model and Methods

In this section we explain the implementation of our approach in detail including how the matrix is encoded, in addition with concrete algorithm procedures of how to find a walkable domain for the mobile robots. The whole procedures can be divided into four steps:

1. Binarizing the matrix
2. Coding the matrix
3. Finding the path
4. Optimization of walkable domain

2.1 Matrix Binarization

The input to the robots should be a planform of maze with the coordinates of the beginning and destination. The maze picture does not need to construct in accordance with the true scale of the robot and the maze. Firstly, the maze pictures need to be binarized. Pixels of a maze picture can be classified into two types. As seen from Fig. 1(a), pixels located in the wall area is called wall pixel and other pixels in the free-space is called path pixel. Without affecting the result of going through the maze, the maze matrix can be simplified by the following way: the wall pixel is represented by 0 and the path pixel is represented by 1. In this way, we create a matrix named binarization matrix, of which the scale is equal to the scale of the original maze picture and each pixel of original maze picture corresponds to an element at the same position in the binarization matrix, as shown in Fig. 1(b). After binarization process, an efficient matrix coding process is required, as usually the maze picture contains a huge amount of data which is challenging to many routing algorithms such as A*. Besides, almost every routing algorithm would find a line but not a path for the robots if the starting point and destination are fixed. Therefore, we present a new matrix coding method and the routing algorithm based on it.



(a) Maze (b) Binarization Matrix (c) Adjacent Pixels (d) Coding Matrix

Fig. 1 The original maze (a) with binary number on it. Extract the numbers to get binarization matrix (b). Gather the adjacent pixels in same row as segment (c).Coding matrix (d) with the extracted the segments.

2.2 Matrix Coding

Matrix Coding is essential in searching algorithm as it reduces the memory space of image which leads to lower computational cost and decides the appearance of the final path.

In this paper, the presented matrix coding methods compresses the adjacent pixels in the same row into one segment, shown in Fig. 1(c), which contributes to our filled- path routing algorithm as well as a more compressed matrix.

This objective of coding matrix is to indicate the position of path pixel in the maze. For example, a binarized matrix of representing the maze is given in the Fig. 1(b), pixels with value ‘1’ represent the free space while value ‘0’ represent wall region. Then we represent all the adjacent pixels of value ‘1’ in a row to be one segment. As it can be seen in Fig. 1(c), each gray region forms a segment. Then every segment is represented in the coded matrix by a column with four basic elements from top to bottom. The first element represents the pixel type: ‘1’ represents a walkable path segment, ‘0’ for a wall segment and ‘3’ for already walked path segment. The second element represents this segment belongs to which row in the binarization matrix. The other two elements represent the beginning and ending position of the segment in that row respectively. In this way, the binarization matrix is transformed to a coded matrix which indicates all the accessible path area and hide the remaining wall area. A coded matrix for the given maze in Fig. 1(c) is shown in Fig. 1(d), each column in coded matrix represents a segment in maze with order from top to down. The whole process is given by Algorithm 1 Coding the Matrix in detail.

Algorithm 1 Coding the Matrix

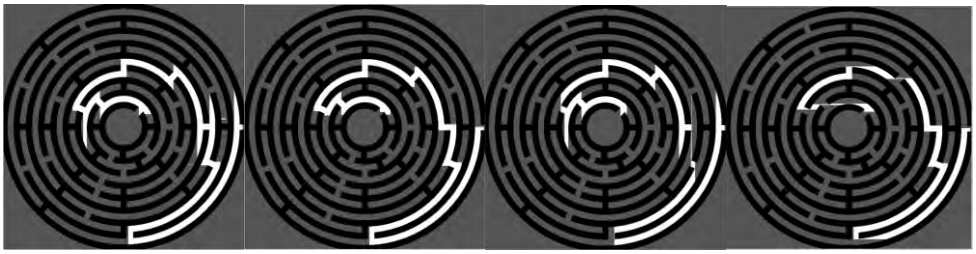
```
Require: M_binary: Binarization form of input maze picture.
Mc ← ∅
count ← 1
for l in range length(M_binary) do
    for w in range width(M_binary) do
        if M(l,w) not equals to M(l,w−1) and M(l,w−1) equals to 1 then
            begin ← 1 end if
        if M(l,w) not equals to M(l,w−1) and M(l,w−1) equals to 0 then
            end←w
            Mc (1, count) ← 1
            Mc (2, count) ← l Mc(3, count) ← begin Mc (4, count) ← end
    count ← count + 1
        end if
    end for
    end for
return Mc: The compressed maze by each segment.
```

Apart from the main elements, for each column of the coded matrix, some extra information could be added to optimize the routing. For example, four extra plug and play elements could be added to reduce computational complexity when finding paths. The first two represent how many new paths are connected to the upper or lower bound of the segment, the last two represent how many accessible paths are connected to the upper or lower bound of the segment. These four elements can be used to make the routing algorithm easier to apply.

2.3 Path Finding

Original A* algorithm cannot be applied to the coded matrix directly, so we provide an A*-like routing algorithm which enable A* and our proposed algorithm coalesced as a whole.

Before searching the path, the starting point and end point of the robot path need to be set. Then the algorithm would locate which segment the starting point belongs to by traversing all the segments in coded matrix. After fixing the starting point, each time before the robot move, the priority to choose which path to take is set for the robots. The rules of setting the priority is as follows: Firstly, comparing the vertical coordinates of the end point and midpoint of the current segment, if the end point is on the top of midpoint, then moving up has higher priority and vice versa. Next, if there exists more than one path of moving up or down, then compare the horizontal coordinates of end point with the adjacent point with these paths. The path with horizontal coordinates more closed to end point has higher priority. Thirdly, if robot meets a dead end at some segment, then it returns back to the original segment and set that dead end segment be to a wall segment (changing the first element of the column that represents the segment to be '0'). Lastly, the algorithm would mark these segments that already walked to prevent repeated routing (changing the first element of the column that represents the segment to be '3'), then the robot would traverse each road according to its priority until finding a walkable path.



(a)Result of the path finding (b) Fixing the strings (c) Path finding after rotation (d) Final filled path

Fig. 2 This picture shows the steps we use to get the filled path. We apply the path finding and receive (a). Then, fixing the 'string' (b), followed by find the path again after rotation and some processing. Finally, fix the 'string' again to get the filled path (d).

In this way, a walkable domain for the robots from starting point to end point can be roughly determined, as can be seen from Fig. 2(a). However, some thorn and string problems may be raised.

2.4 Optimization of Walkable Domain

Stings usually appear when path turns around. The phenomenon of turning around typically happens when the robot meets a dead end and has to returning to the original segment to find another path. In this case, only one segment connects the entrance and exit of this path, so a 'string' is generated. As can be seen in Fig. 2(a), there are several very thin paths, which look like 'string'. This is similar with the prior routing algorithm such as A* as it produces a line instead of a path, therefore, we need to find a way to deal with these extra strings. Another problem called 'thorns' may appear as well when we implement the algorithm. As it can been seen from Fig. 2(a), there are many redundant paths that robots walk from starting point to end point, this phenomenon is called 'thorns'. Fixing the strings: The pseudo-code for fixing 'strings' is given in Algorithm 2 Fixing the String. Before fixing the strings, we need to judge and locate the string segment in the path by the following formula:

$$Path\left(2, P_{steps} - 1\right) = Path\left(2, Psteps + 1\right)$$

$Path$ represent the collection of in order segments as the filled path. P_{steps} is the current step number, every time the robot moves, the P_{steps} will increase by 1. $Path\left(i, P_{steps}\right)$ denotes the i-th element of column in coded matrix which represents the current segment where the robot is currently located when the step number is P_{steps}. Then the algorithm will traverse all the segments and locate the 'string' segment according to the formula:

$$N_{insert} = \frac{S}{2}\left(\sum_{i \in turn} Path(4, i) - Path(3, i)\right) \qquad \text{turn} = \{Psteps + 1, P_{steps} - 1\}$$

N_{insert} is the number of segments we need to insert to fix the 'string'. S denotes the sensitivity, which can be changed according to the shape of the maze, usually it is set to 1. The formula implies we can insert several segments around the 'string' place to solve the problems, the number of inserted segments is set to be N_{insert} , where we set the value of N_{insert} to be a multiple of arithmetic average of the length of two segments at both ends of the 'string' with the sensitivity. After tacking the 'string', the maze picture is shown in Fig. 2(b), from which we can see that all the 'string' have been successfully deleted, and path is filled.

Algorithm 2 Fixing the String

Require: P_{raw}: Raw path in the form of segments

for l in range length(P_{raw}) **do**

if $P_{raw}(2, l-1)$ equals to $P_{raw}(2, l+1)$ **then**

$P_{fixed} \leftarrow$ insert n segments at proper place in P_{raw}

$P_{raw} \leftarrow P_{fixed}$

$l \leftarrow l + n$

end if

end for

return P_{fixed}:Path with fixed strings

Fixing the thorns: The appearance of 'Thorn' is because our algorithm represents all the adjacent data in the same row as one segment, and all elements of the segment that robots have walked to be a walked path. In this way, robots may walk some redundant path as some elements in a segment may not be walked.

To tackle this problem, we can firstly record the path from starting point to end point that robots have already walked to be the walkable domain and other regions are all set to be wall regions. Then we rotate the maze by 90 degrees and implement the same algorithm to let robots go through the maze again, the result maze picture is shown in Fig. 2(c). Lastly, we can fix the strings by the following method, and the final image is shown in Fig. 2(d), so far we have successfully found a walkable domain from starting point to end point.

3. Experiment

To verify the performance of our algorithm, we have compared our approaches with other image coding methods and other searching algorithms.

3.1 Comparison with Other Image Coding Methods

In order to let robots find a walkable path in maze, coding the image and uploading it to the robot are necessary. In this paper, we have compared our approach with two classic image coding methods. The first one is Compressed Sparse Row (CSR) [11], the second is quadtree image coding.

Compressed Sparse Row (CSR) is a typical method of compression. Three types of data are required to represent image matrix: numeric values, column indexes, and row offsets. The row offset represents the start offset of the first element of a row in values. This form of storage requires the memory size of $2nnz + n + 1$where $'\text{nnz}'$ represents the amount of data that is not zero in the matrix and 0n0 is the number of columns. For quadtree image coding [12], it divides the image into quarters iteratively until all values of pixels in same block are equal. For all nonzero blocks, let the northwest corner pixel represent the total number of pixels in the block it belongs to, which can fully represent the image matrix.

In order to compare the compression effect of these methods, the compression rate can be defined as follows:

$$\eta = \frac{N_i}{N}$$

being respectively the total number of bits for storing the image matrix after compression in i-th method, N is the total number of bits before the compression.

These image coding methods are all undistorted. In our comparison, to ensure that the initial amount of data for all methods is constant, we resize the all the image size to be 512512 before coding. And the comparison of those methods in compression rate is provided in Table 1.

Table 1 Coding rate comparison

Method	Total bits to represent maze	Compression rate
CSR	217921	0.8313
Quadtree image coding	114699	0.4375
Our approach	22516	0.0859

3.2 Comparison with Other Searching Algorithms

To test practical usability of our approach, we have compared our methods with two popular methods, Q-learning based on reinforcement learning and A* algorithm respectively. For Q-learning, the advantage is that it can ensure to find a shortest path and for A*, the advantage is that it runs relatively fast and its modifiability. Final assessment criteria of those algorithms include computational complexity, whether it can form a path, whether it guarantees to find a shortest path, and whether it requires a matrix coding before implementation of the searching algorithm. On all the methods, we code the maze as 0-1 sparse matrix with size of 650650, and robots in maze only have 4 directions to move including up, down, left and right. A detailed analysis of their weaknesses and strengths is given as follows:

Q-learning Algorithm [9]: It is a useful algorithm to solve certain kind of problem that involves a robot which needs to interact with its environment. It is a derivative of reinforcement learning (RL) [13] which could train an AI system. The basic principle is that in the interactive environment, the robot uses its own experience and feedback to learn through trial and error.

Q-learning is an off-policy algorithm. Intuitively, the result of the algorithm is to create a Q table to record the weight of different policies (actions) in each state since we regard the robot in the maze (the agent could be robot in the application scenario) as an FSM (finite state machine) and weights are constantly updated based on historical experience.

Theoretically, the Q-learning algorithm is mathematically guaranteed to converge to find the shortest path and after training, the optimal action for every possible state could be known. However, for such a huge sparse matrix, the algorithm will spend a large amount of time to produce the final Q-table and the required memory to store the Q-table will grow as the square of the number of states. For example, this algorithm will cost more than 2 hours when iteration number is set as 3. However, the iteration number needs to be set far more than 3(typically hundreds of times) to get the optimal Q-table. Besides, our experiment shows that iteration number have to be set more than 400 so that the algorithm could find the correct path for solving a 66 maze. Therefore, though the algorithm is guaranteed to find a shortest path, it is inapplicable for solving our maze with such complexity.

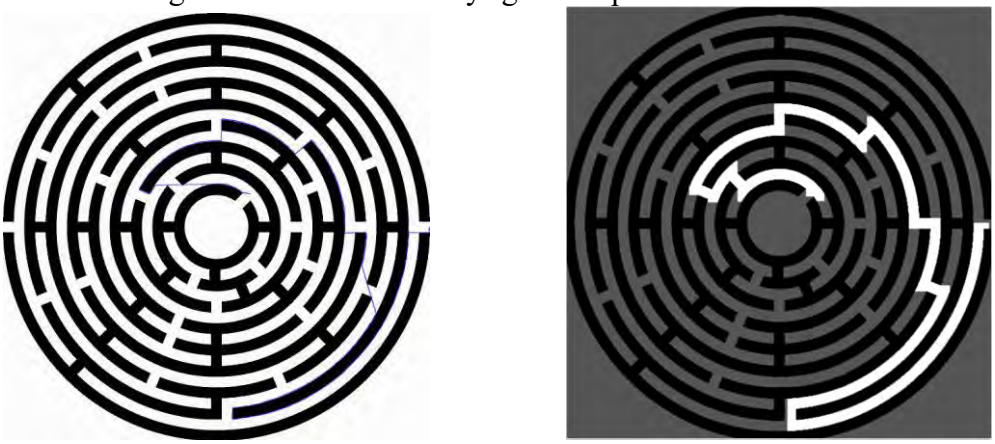
A* Algorithm [6]: it is one of the most effective direct searching method for finding the shortest path in static road network. It also acts as a heuristic approach for many other problems. Robot starts with start point A, checking the adjacent squares, and extends them out until it finds the target point. The core formula is expressed as:

$$f(n) = g(n) + h(n)$$

Where $f(n)$ is the evaluation function of node n from the initial point to the target point, $g(n)$ is the actual cost of going from the initial node to the node n in the state space and $h(n)$ is estimated cost of the optimal path from node n to the target node.

The performance of A* algorithm exceeds the Q-learning algorithm in time consuming. In our experiment, A* costs only 3 seconds to solve the maze completely. What is more, we also have to concede that the A* algorithm takes less time than our method. However, there exists two deficiencies when applying this algorithm in practice. Firstly, as can be seen in Fig. 3, the left figure is the searched path by A* algorithm, and the right figure is the searched path by our approach. To run A* algorithm, we must assume that the robot is a pixel in the figure, so an accurate modelling needs to established in accordance with the true scale of the robot and the maze, then we can decide how many pixels the width of the maze occupy. After running A* algorithm, we can find that it has searched a same path as our approach, but it is almost walking against the wall as the robots only occupy one pixel. Although

it is a shortest path, but in practice, this is impossible because the robot has inertia, and there is a possibility of hitting the wall when it is walking against the wall. As our approach find a filled path from beginning to destination, so there is no need to build the model in real scale, and it can also ensure that the robot will not hit the wall. Secondly, this algorithm cannot always guarantee to find the shortest path. Let D represents the distance from state n to the target state and H represents the estimated amount of movement from the current point to the target point. There are 3 conditions. If $H \leq D$, in this case, the number of searching points is huge, and the algorithm undergoes a large searching range. It has a low efficiency but can get the optimal solution. If $H = D$, then the search will be guaranteed to find the shortest path, and the searching efficiency is highest. If $H > D$, the number of searching points and the searching range is small, so the efficiency is high, but cannot guarantee to find the shortest path. Since H is often called heuristic that means we cannot know the actual length of the path in advance. A* algorithm could not always get the optimal solution.



(a) Searched path of A* (b) Searched path of our approach

Fig. 3 Comparison of A* and our approach

We have test running time of A* and our algorithm in this certain maze. Our CPU is Intel i7-7700 and 4.2GHz in dominant frequency including four cores with eight threads, which can be seen in Table 2.

Table 2 Time consumption comparision

	A*/s	Our Method/s
First Test	0.87	8.31
Second Test	0.71	10.56
Third Test	0.93	7.48
Fourth Test	0.68	15.21
Fiveth Test	1.12	9.33
Average	0.86	10.18

In conclusion, we test the same maze on different algorithms and made a brief comparison given in Table 2 Q-learning can guarantee the algorithm's ultimate convergence to shortest path, but the required computing time is too long and consume relatively large computer memory. Although A* can solve the maze in a relatively short time, it is not guaranteed to always find shortest path since it depends on what heuristic function is used and cannot avoid collisions with walls in practical application. By contrast, the proposed new algorithm could solve this sort of awkward situation by means of forming a practical walkable domain.

Table 3 Path finding algorithm comparison

Method	Complexity	WDF[1]	GBO[2]	RMC[3]	Disadvantages
Q-learning	High	NO	Yes	NO	Long calculation time
A*	Medium	NO	Uncertain	NO	Cannot guarantee optimal
Our method	Medium	YES	NO	YES	Cannot guarantee optimal
Basic Theta	Low	NO	NO	NO	Long calculation time
JPS	Medium	NO	NO	NO	Cannot guarantee optimal

1 Walkable Domain Finding
2 Guaranteed to be optimal
3 Require matrix coding

4. Conclusion

The maze path planning problem is extensively researched. However, in some specific application, such as routing of pathfinding cars or mobile robots, building a accurate model in accordance with the true scale of the robot and the maze is not line with actual requirements. Also, some shortest paths for robots to take do not consider the size of robots which may cause collisions with walls. New approaches to find a practical path in a relatively short time need to be developed as well.

We develop a new approach for finding a walkable domain of a mobile robot, which will avoid collisions thoroughly. Our method starts by coding the maze in order to enable fast execution of searching algorithm as well as save storage space. Then we apply our searching algorithm. During the process, some inevitable thorns and strings may appear, which are addressed in following step and then we can get final walkable region. It is compatible with most maze problems using this algorithm for finding actual walkable path. In addition, we make a comparison with many other methods. Although our method does not perform as well as A* algorithm does in running time, our results could find actual walkable path which could avoid collisions with walls thoroughly. Compared with Q-learning, there is an incomparable advantage in running time and memory usage.

The performance of our algorithm is quite promising. However, there is still plenty of room for improvement in our approach. In the future, we will make a further improvement to let our algorithm more compatible in practical engineering field.

5. Availability

Code for our own methods and comparison algorithms is available on GitHub: https: //github.com/CheneyFeng/OptimizedArrayPicturalStorage. Our work is still in progress and might be subjected to some subtle changes.

Acknowledgements

First and foremost, thank to everyone involved in this project. In particular, Glasgow College, UESTC who found this project. Institute of Science and Management of the School League Committee offer the place to discuss. Data Mining Lab offer the guidance on paper writing.

Also, I would like to thank Dr. Wei Han, Dr. Liyan Zhang and Dr. Jinjun Zheng for all their kindness and help.

References

[1]. Samet, H. and Webber, R.E., 1985. Storing a collection of polygons using quadtrees. ACM Transactions on Graphics (TOG), 4(3), pp.182-222.

[2]. Thorpe, C.E. and Matthies, L.H., 1984. Path relaxation: Path planning for a mobile robot (pp. 576-581). Carnegie-Mellon University, the Robotics Institute.

[3]. Achour, N. and Chaalal, M., 2011. Mobile robots path planning using genetic algorithms. In The seventh international conference on autonomic and autonomous systems, Baker, ICAS (pp. 111-115).

[4]. Hoel J H. Some variations of Lee's algorithm[J]. IEEE Transactions on computers, 1976, 100(1): 19-24.

[5]. Lee C Y. An algorithm for path connections and its applications[J]. IRE transactions on electronic computers, 1961 (3): 346-365.

[6]. S.G. Cui, H. Wang, L. Yang A Simulation Study of A-star Algorithm for Robot Path Planning 16th international conference on mechatronics technology (2012), pp. 506-510

[7]. Hart P E, Nilsson N J, Raphael B. A formal basis for the heuristic determination of minimum cost paths[J]. IEEE transactions on Systems Science and Cybernetics, 1968, 4(2): 100-107.

[8]. Land A H, Doig A G. An automatic method of solving discrete programming problems[J]. Econometrica: Journal of the Econometric Society, 1960: 497-520.

[9]. Sutton R S, Barto A G. Reinforcement learning: An introduction[J]. 2011.

[10]. Morris G, Nevet A, Arkadir D, et al. Midbrain dopamine neurons encode decisions for future action[J]. Nature neuroscience, 2006, 9(8): 1057.

[11]. DAzevedo E F, Fahey M R, Mills R T. Vectorized sparse matrix multiply for compressed row storage format[C]//International Conference on Computational Science. Springer, Berlin, Heidelberg, 2005: 99-106.

[12]. Gargantini I. An effective way to represent quadtrees[J]. Communications of the ACM, 1982, 25(12): 905-910.

[13]. Watkins C J C H, Dayan P. Q-learning[J]. Machine learning 1992, 8(3-4):279-292.