Robotica: (2021) volume 39, pp. 928–944. © The Author(s), 2020. Published by Cambridge University Press.
doi:10.1017/S0263574720000818

One Way to Fill All the Concave Region in Grid-Based Map

ZiYing Zhang†, Xu Yang†, Dong Xu†*, Ke Geng‡, YuLong Meng† and GuangSheng Feng†

†*College of Computer Science and Technology, Harbin Engineering University, Harbin, China*
‡*College of Computer Science and Technology, HeiLongJiang Institution of Technology, Harbin, China*

(Accepted August 12, 2020. First published online: September 10, 2020)

SUMMARY
The search space of the path planning problem can greatly affect the running time and memory consumption, for example, the concave obstacle in grid-based map usually leads to the invalid search space. In this paper, the filling container algorithm is proposed to alleviate the concave area problem in 2D map space, which is inspired from the scenario of pouring water into a cup. With this method, concave areas can be largely excluded by scanning the map repeatedly. And the effectiveness has been proved in our experiments.

KEYWORDS: Path planning; Concave obstacles; Filling container; FC algorithm; Grid-based map.

1. Introduction

Path planning is one of the core issues of autonomous robots. Researches have been carried out for a long time and some methods have been issued, which include A*,[1] Rapid-exploring Random Trees[2] (RRT), Probabilistic Roadmap Method[3] (PRM) and artificial potential field method.[4] Some biology-inspired AI (Artificial Intelligence) techniques have been proposed in these years, such as neural networks, genetic algorithms, reinforcement learning and ant colony algorithm.[5]

As one of the fundamental elements of path planning, the presentations of map influence the running time and planning effect greatly. The common and popular presentations include grid-based map and topological map, in which the grid-based map can be built and maintained easily and it also can produce accurate metric maps.[6] Currently, the grid-based map is the most common and fundamental presentation of maps and is widely used in different fields, such as Simultaneous Localization and Mapping (SLAM), electrical games and Mars Exploration Rover.[7]

The concave area in map, also called the deadlock, local minimum or limit-cycle problem, can result in the invalid search time for the path planning algorithm and even navigation failures.[8] This kind of problems is solved by indicating manually in early electrical game development, while it always costs a long time and is not flexible enough for further operations. Currently, the popular solution in robot navigation and electrical game development is the hierarchical representation of the grid map which includes multi-resolution grid-based map, navigation mesh, navigation points and the topology abstractions. Although the above practices can alleviate the running pressure of the path planning algorithm and even reduce the impact of the concave obstacles, they do not specifically address the problem of the concave region, nor does it avoid to search for such area, so this problem still needs to be solved. In addition, the above method is only suitable for the global path planning. And it still takes time when detecting in local path planning or other scenario.

In this paper, a general solution for the 2D path planning is proposed to greatly reduce the impact of concave areas. The solution of filling container method is abbreviated as FC in this paper, which is

* Corresponding author. E-mail: xudong@hrbeu.edu.cn

https://doi.org/10.1017/S0263574720000818 Published online by Cambridge University Press

inspired from the daily life. The concave area can be treated as the container since both of them are composed by bottom and walls. When the liquid is filled into the container, the bottom will support the liquid and the liquid will flow out the wall when the container is full. The filling procedure is abstracted and can be transferred into the method of filling the concave area.

The FC algorithm is suitable for any grid-based map with concave obstacles, but the result depends on the obstacle distribution. In the case of fewer noise points in the map, the FC algorithm can achieve good results. However, the convex transformation is not performed strictly by the FC algorithm with the definition of the concave set for the path planning problem,[9] so it cannot guarantee to complete the concave polygon. And in some cases, it can only reduce but not eliminate the adverse effects on the path planning algorithm. The experiment shows that filling the concave area can reduce the planning time greatly because the invalid searching time is reduced. The result can be more obvious when the planning is executed on the same map for multiple times.

The paper is organized as follows: the related works will be discussed in Section 2. Section 3 is the descriptions of the grid-based map, the concave area and the filling area. In Sections 4 and 5, the details of FC method will be discussed. And the experiment is introduced in Section 6, and the conclusion and future works are discussed in Section 7.

2. Related Work

Navigation on native high-resolution grid-based map always occupies high memory and system resource. The general solution for this problem is to reduce the complexity of the map by preprocessing, typically including replacing traditional grid-based map with multi-resolution grid-based map, constructing topological map and generating navigation mesh with grid-based map. In this section, all these above methods, the convex hull in computer graphics and the traditional solution of concave areas will be discussed.

2.1. Multi-resolution grid map

The implementation of higher resolution on grid-based map may waste the storage space and be excessive overhead in path planning algorithm, while the implementation of low resolution may lead to poor accuracy and discontinuity of map. In order to solve this problem, the grid-based maps of variable resolutions are developed to replace the traditional grid maps.[10–13]

Chan et al. have proposed an approach to automatic path planning based on quadtree representation.[10] Each non-leaf node in a quadtree has accurately four children, each child indicates an equal size region divided from the parent area. A two-dimensional space is usually represented by recursively fractionizing it into four quadrants or smaller regions.[14] A non-collision path can be derived when the corresponding quadtree is traversed. The following research[11,12] in this area will be mainly concentrated on the division of grid space, the procedure of building quadtree with SLAM and the coding rule to speed up the operation of locating quadtree nodes.

Multi-resolution grid-based map has its own shortcoming. The cells closed the obstacles will generate a lot of quadtree nodes because of the discontinuous free area. And the situation can be worser with irregular obstacle edges. In addition, because of the specific map representation method of multi-resolution grid map, most methods designed for building topological maps need to be redesigned.

2.2. Topological map

The topological map based on metric map draws more attention in these days, because topology map has lower space complexity and it can permit efficient planning.[6] The research in this area can be classified into two groups: generalized Voronoi graph (GVG) and spectral clustering. The GVG-based segmentation is proposed by Thrum.[6] And its fundamental theory is to build Voronoi diagram from free space and then derive the skeleton of the environment so as to find out some narrow passages and build the critical lines. In this way, the free space can be divided into serials of disjoint regions and the topological graph is generated. This method is easily accessible in the structured environments while it often produces unnecessary sets of unconnected point sub-sets with obstacle irregularities. Some methods have been proposed to overcome this drawback, for example, GVG. Liu et al. introduced discretized Voronoi diagram, which can build topological graph from the maps built by SLAM.[15] This method still generates unnecessary sets of unconnected points even the areas are

https://doi.org/10.1017/S0263574720000818 Published online by Cambridge University Press

connected. Tsardoulias et al. tried to use Manhattan distance instead of Euclidean distance and the method was called approximate GVD.[16] They introduced a single plane sweep method to calculate the skeletonization of free space and thus reduce the execution time. However, an inevitable situation is that some complex and useless connected graph will be generated when there are small noises such as from the table legs or the paper boxes in the map.

Clustering methods can build topological maps by dividing the grid cells into different clusters. Cluster analysis is the organization of a collection of patterns into clusters based on similarity. Patterns within a valid cluster are more similar to each other, compared with the patterns belonging to different clusters.[17] The similarities of cells of grid-based maps are usually calculated with the Manhattan distance or the Euclidean one. Because of the shortcoming of the clustering algorithm, the above two evaluation methods always generate the same size regions, which is unrealistic in real map. The distance-based evaluation is not sensitive to the objects, such as wall. And it can be found that the cells from the same cluster appear on both sides of a wall. Zhou et al.[18] implement wavefront distance to evaluate the similarity between grids to overcome the drawback and thus that the cells from one cluster will not distribute on both sides of a wall. Another drawback of clustering is that the number of clusters must be determined manually, which influences the result of clustering greatly.[19] And clustering is not a suitable choice for an automatic system. Besides the above problem, the computational complexity of spectral clustering also hinders its performance.[15] Kaleci et al.[20] choose the free cells obtained in a fixed time interval during SLAM procedure. The cluster number will be adjusted automatically to avoid the appearance of over-sized affinity matrix, while this process will result in the number of clusters related to the robot execution time and will generate a large number of meaningless topological points.

The above methods that build topological maps using metric maps are mostly time consuming, especially the clustering methods. So they are not the good choice for a faster solution.

2.3. Navigation mesh

In addition to representing the obstacles with grids, we also represent the walkable areas with non-overlapping polygons, which are called a navigation mesh. Marcelo Kallmann[21] proposed a new approach called Local Clearance Triangulation, which connects the vertices of obstacles including extracting the vertices from the curved obstacles, so the map can be divided into non-overlapping triangles. Some projects prefer arbitrary polygons instead of triangles, such as the game engine of Unity 3D. However, we did not find any research applying the navigation mesh in non-structured or semi-structured environments, which is because building navigation mesh requires vertex information of obstacles and it is impossible in unstructured environments.

2.4. Solution to the concave region

All the researches above concentrate on how to reduce the complexity of path planning by changing the representation of maps. And we will concentrate on the solution of concave areas and provide a method for grid-based maps to remove those concave areas. The FC method we adopt is not sensitive to the edge of obstacles and we believe it can combine with some other methods.

The problem of concave polygon convexification can be treated as a convex hull problem in computer graphics. Generally, a limited number of points, which are not repeated or collinear, are considered in graphics.[9] Only integer coordinate points are chosen for accurate computation. Many methods have been proposed to solve the convex hull problems, such as the quickhull algorithm,[22] the Graham scan algorithm[23] and the divide and conquer algorithm.[24] In fact, in order to guarantee the accuracy, some extra processing is needed to solve the convex hull problem of graphics. Similar to the construction of navigation mesh, most of the algorithms need a set of vertices which are defined earlier. With the attempt of applying this method in map processing, vertices of the obstacles should be identified. In ideal situation, the method should be applied to identify vertices and determine whether the objects are convex or concave before convex hull processing. For the maps with many obstacles arranged together, the process will become even more complicated because the obstacles interact with each other.

Some researches process the concave regions during path planning. For example, in the artificial potential field method, the path planning will fail for that the concave region causes local minimum. Several methods, such as simulated annealing[25] and virtual obstacle,[26] can lead the robot to jump out

https://doi.org/10.1017/S0263574720000818 Published online by Cambridge University Press

of the local optimal point. And the planning algorithm costs a long time and thus the result is not optimal. Other modifying path planning algorithms have the similar problems. In the field of bionic algorithms, ACO proposed by the Italian scholar M. Dorigo is investigated popularly as an important path planning solving method. However, some problems existed in Ant Colony Optimization (ACO) cannot be ignored. The ACO deadlock state is easily generated in the map with U-type obstacles. Therefore, Zhang et al.[27] proposed a novel ACO-based method, called Self-Heuristic Ant (SHA). In their research, the fallback strategy which allows ants to move backward is integrated with SHA, making full use of each ant as well as the taboo nodes to fill the special areas which contain dense obstacles or many concave blocks. In the meantime, a new information exchange mechanism is established to enhance the ability of ants to identify the U-type traps. As a result, ants are prevented from entering the U-type traps and the planning algorithm costs a long time, leading to an unideal result. Therefore, we believe that the areas should be processed before the path planning.

There have been some researches processing the concave regions before the path planning. Chen et al.[28] proposed an improved OAE-Q(λ)-learning path planning method by introducing the concave area expansion strategy to avoid repeated invalid actions when the agent falls into the obstacle area. This method fills the concave area in the map before the path planning. Firstly, according to the boundary judgment condition, the concave region is traversed to determine the boundary points and the opening direction. Subsequently, the filling is carried out in the order from bottom to both sides, stops the process when obstacles are encountered and ensures that there is a buffer between the filling layer and obstacles, and then switching to the sides of the area and starting to fill transversely in the same way. Although this method has the possibility of expansion, the author has not given too much discussion on the treatment of unstructured concave regions. In addition, this method does not consider the change of the starting and ending point, so it is unsuitable for multi-target scenarios.

3. Related Definitions of FC Algorithm

In this section, we will give the definition of the map space, the concave area and the padding block. And we will also introduce the block association and the updating of block association table.

3.1. Grid cell definition

Definition 1. The map space is a fixed, closed and bounded rectangular region with height $h > 0$ and width $w > 0$. Precisely, the map space is the set $M = \{(x, y) \in \mathbb{Z}^2 | 0 \leqslant x \leqslant w, 0 \leqslant y \leqslant h\}$. Each set of (x, y) indicates a specific cell.

Definition 2. Each cell is equipped with a value to indicate its type, which is recorded as $T(x, y)$. For each cell of (x, y), the definition of $T(x, y)$ is as follows:

$$T(x, y) = \begin{cases} 0, & \text{if the grid cell is free.} \\ 1, & \text{if the grid cell is occupied in the original map.} \\ t, & \text{if the grid cell is in the padding area, } t \geqslant 2 \end{cases} \tag{1}$$

The value of $t \in \mathbb{Z}$ depends on the calculation of FC method. All the cells from the same padding area have the same value of t. When there are more padding regions needed, the value of t increases gradually. In a binary occupancy grid map, true values represent obstacles and false values represent the free workspace. The description of Definition 2 is suitable to indicate whether a cell is free or occupied in a binary occupancy grid map. For probability grid map, the value of a cell indicates the occupancy probability of the cell. Following the tradition of path planning, a threshold can be selected to distinguish the occupied state from the free state.

The type of cell is defined to indicate different padding areas and will be used in block decomposing, which will be explained in Section 5. If the block decomposing is unnecessary, the type values of both the obstacle and the filling block will be set to "1". When the path planning algorithm is executed, the cells with value will be equal to or bigger than "1", which are treated as obstacles. In this way, the invalid searching area is reduced. The cells, which have the type value bigger than "1", can also be treated as the areas which need more energy to pass. And it is the same idea in electrical games like Dragon Age: Origins and Age of Empires. Typically, those areas can be classified into different groups, such as the marsh and the road.

https://doi.org/10.1017/S0263574720000818 Published online by Cambridge University Press

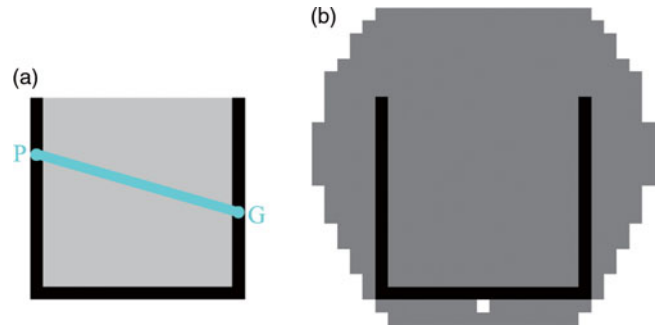


Fig. 1. Nonconvex region. (a) Concave polygon. (b) The area searched by A* planner.

3.2. Concave area

The problem of concave polygon convexification can be treated as a convex hull problem in the field of computer graphics. The convex hull problem is defined as follows:

Definition 3. If $\forall X, Y \in S$, the line segment $(t-1)X + tY \in S$ holds for $t \in [0, 1]$, then the set $S \subset \mathbb{R}^n$ is a convex set.[9]

In binary occupancy grid map, each cell should be a concave cell or nonconcave cell. In order to explain it clearly, we shall consider a scenario of an isolate obstacle. And the concave cell is defined as follows:

Definition 4. $\forall(x,\ y) \in M,\ T(x,\ y) = 0.\ \exists c_1(x_1,\ y_1) \in M$ and $c_2(x_2,\ y_2) \in M,\ T(x_1,\ y_1) = 1,\ T(x_2,\ y_2) = 1$, The obstacle grid cells c_1 and c_2 are both continuous. The concave cell $(x,\ y)$ satisfies $Ax + By + C = 0$:

$$\begin{cases} A = y_2 - y_1, \\ B = x_2 - x_1, \\ C = x_2 \times y_1 - x_1 \times y_2, \end{cases} \tag{2}$$

A concave area is combined of a set of concave cells from continuous obstacle grid cells, and it can belong to multiple concave areas.

Figure 1(a) illustrates a typical concave polygon in which the black cell is the obstacle cell and that the light gray area identifying the cells satisfies Definition 4. The line segment connecting P and G is not completely in the source set. In Fig. 1(b), the dark gray area is the area searched by the A* algorithm. It can be seen that a considerable amount of time is spent in the search of the cells in the concave area.

Theorem 1. *If the start point and the end point do not locate in one concave area, all the cells of this concave area cannot be in the optimal path.*

Proof. Setting that starting point and the target point are not inside the concave obstacle and assuming that the optimal path passes through a point in the concave area. Since the goal is not in the concave area, the path must have two intersections with the boundary of the concave area. A triangle will be formed by connecting the point X that enters the concave region, the point Y that exits the concave region and any point Z in the path of the concave region. Because the sum of any two sides of a triangle is greater than the third side, that is, $Path\ XY < path\ XZ + path\ ZY$, then this $path\ XY$ is shorter. The assumption is contradictory, so theorem is proved.

In Definition 4, the concave cells are easier to identify because there is only one concave area in the simple environment. In a real-world map, the concave regions may be connected together and nested with each other, which makes it impossible to distinguish the parts effectively and difficult to determine which cells belong to the same set of points in the concave region.

3.3. Padding blocks and block association

It is very common that there are more than one concave areas in a map. Generally, each concave area is filled by multiple padding blocks. For example, as shown in Fig. 2(b), the blue area is the padding blocks, which are created at the same stage of the FC algorithm running process. And each of them

https://doi.org/10.1017/S0263574720000818 Published online by Cambridge University Press

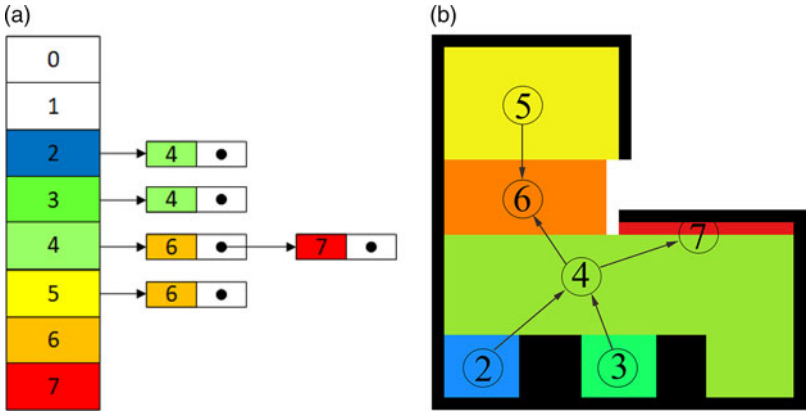


Fig. 2. Block association table. B(0) and B(1) are empty linked lists because they are not the blocks. $T_b(b_4) \in B(T_b(b_2))$ and $T_b(b_4) \in B(T_b(b_3))$, because during the running of the algorithm, b_2 and b_3 are constructed firstly, and b_4 regards them as obstacles and pads upon them. The same block can also exist under the head node of multiple blocks because the block has contact with multiple blocks. The relationship between the blocks is represented in the form of graph structure in (b). (a) BAT. (b) Padding blocks.

is composed with grid cells of the same type value, namely block $b_n = \{(x,\ y)|T\ (x,\ y) = n\}$. To make it clear, we use type numbers to represent blocks. The type of block b_n is recorded as $T_b(b_n)$, so $T_b(b_n) = T\ (x,\ y),\ (x,\ y) \in b_n$. Type increases gradually when more blocks are built.

The former block will be treated as an obstacle by later block, so when building a block, the connections between blocks must be built. The pointer of later block must be stored into the former block. When a block is eliminated, all the other blocks built on that block must be decomposed. We have built a block association table *BAT* to store the relationship between blocks. The $T_b(b_n)$ of each block b_n in the table is equal to the index ordinal. With the running of the program, more blocks are added to the *BAT*. Each slot has a pointer to a linked list $B(type)$. If a block b_{new} under construction is built on the previously established block b_{old}, the linked list node with $T_b(b_{new})$ as the key field will become a new node in $B(T_b(b_{old}))$. We have defined that $T_b(b_{new}) \in B(T_b(b_{old}))$.

This structure is quite similar to adjacency list representation of directed graphs but still has some specialties: (1) there is only one directed edge that points to the other node, and there is no opposite edge. (2) Only nodes with a large key field appear on the successor of a linked list with a small index number. For two blocks b_n and b_m, if $n > m$, then only $T_b(b_n) \in B(T_b(b_m))$ may appear. The structure of the table is shown in Fig. 2(a).

Algorithm 1 shows the procedure of *BAT* updating. Because blocks vary in shape and size, the detecting procedure is updated by level and level. The line 1 to line 5 check the type of each cell in the next level, which is equivalent to the "bottom" of the container. The line 6 to line 11 check the left and right types of current level. Algorithm 1 is executed when there are blocks needed to be built during the operation of FC algorithm.

4. Filling the Container Method

In this section, we will introduce our main contribution of filling the container method. Firstly, we give the description of the basic idea and pseudo code, which is followed by top judgment and an alternative strategy. Then, we will give the full analysis of this algorithm and give the complexity analysis in the last of this section.

4.1. Life inspiration and the FC algorithm

The concave areas in the map are similar to some containers in our daily life. If we treat the edge as the wall or bottom of a container and treat the padding cells as water drops, the water drops will increase from the bottom of the container with the influence of gravity. And the water level will keep grow until the whole area is filled. During the procedure, whether the upper level is an obstacle should be considered, otherwise, the possible channel will be blocked (see Section 5 for details). The

https://doi.org/10.1017/S0263574720000818 Published online by Cambridge University Press

Algorithm 1 UpdateBAT

Input:
Block association list B, current level range l and r, level le, type t, Map M;
Output:
Updated Block association list B

```
1: for i ← l to r do
2:    if T(i, le) ≠ 1 and t ∉ B(T(i, le − 1)) then
3:       Add node with key t to B(T(i, le − 1));
4:    end if
5: end for
6: if T(l − 1, le) ≠ 1 then
7:    Add node with key t to B(T(l − 1, le));
8: end if
9: if T(r + 1, le) ≠ 1 then
10:   Add node with key t to B(T(r + 1, le));
11: end if
12: return B;
```

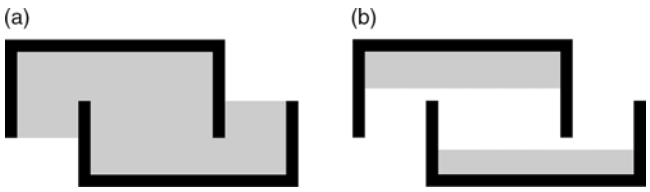


Fig. 3. The situation that needs top judgment. (a) Complete convexification. (b) Top judgment.

FC algorithm is summarized in Algorithm 2. For both binary grid map and occupying grid map, the cell value should transform into the type value based on Definition 2 without further operations.

The outer loop is to ensure that every cell is traversed in the map by starting from any corner of map. The pseudo code from lines 5 to 7 ensures that the padding procedure begins at the correct place, which means that the begin cell is not an obstacle and that there is an obstacle cell below it. Lines 8–20 are the horizontal expansion at the left and right of the cell. The expansion will not stop until the whole level is filled, which indicates that both the left side and the right side are obstacles. The detection of the up and down sides stops only until one of the following two possibilities happens: there is an obstacle on the up level or there is no obstacle on down level. Either situation happens, the inner loop will stop. The lines 21–31 are in charge of updating the type value. If the $level$ is first level of this inner filling procedure, the connected nodes are indicated as the same new type. Otherwise, the type value is same as that of the lower level. And if necessary, *BAT* should be updated. The ending expansion flag F_{exp} is designed for the filling procedure on vertical direction. The loops of line 4, line 9 and line 15 are designed to control the filling procedure on different directions, so they work in parallel and all of them can be called inner loop, which will not stop until the F_{exp} become true.

Algorithm 2 shows the process of downward direction. The FC algorithm can be used in a overlay manner to repeatedly simulate different directions of gravity for building blocks. In this overlay process, the rules in multi-direction are consistent with single direction, and the block type remains unchanged. The experiment section provides a comparison between single direction filling and multi-direction filling.

4.2. Top judgment

In Section 4.1, we mentioned that in the algorithm, the top and bottom are checked simultaneously when processing the node. The check of the lower cell is aimed to look for support, which is crucial to the algorithm. The purpose of the top judgment is to avoid inappropriate filling and drown the accessible road. For the complex concave area with nesting structure, such as Fig. 3(a), it is very difficult to distinguish two obstacles. All the light color areas satisfy Definition 4. And obviously

https://doi.org/10.1017/S0263574720000818 Published online by Cambridge University Press

Algorithm 2 Filling The Container With Gravity Down

Input:
Map M;
Output:
Processed map M
1: Produce an empty block association list B, set current type number $t \leftarrow 2$, current extension level le, ending expansion flag F_{exp};
2: **for all** $T(x, y) \in \{(x, y) \in M | x \neq 0, x \neq h-1, y \neq 0, y \neq h-1\}$ **do**
3: $le \leftarrow y$, $F_{exp} \leftarrow False$;
4: **while** $F_{exp} \neq True$ **do**
5: **if** $T(x, y-1) = 1$ and $T(x, y) \neq 1$ **then**
6: $F_{exp} \leftarrow True$;
7: **end if**
8: $l \leftarrow r \leftarrow x$;
9: **while** $F_{exp} \neq True$ and $T(l-1, le) = 0$ **do**
10: $l \leftarrow l - 1$; // Expansion to the left
11: **if** $T(l, le-1) = 0$ or $T(l, le+1) \neq 0$ **then**
12: $F_{exp} \leftarrow True$; Break;
13: **end if**
14: **end while**
15: **while** $F_{exp} \neq True$ and $T(r+1, le) = 0$ **do**
16: $r \leftarrow r + 1$; // Expansion to the right
17: **if** $T(r, le-1) = 0$ or $T(r, le+1) \neq 0$ **then**
18: $F_{exp} \leftarrow true$; Break;
19: **end if**
20: **end while**
21: **if** $F_{exp} = True$ **then**
22: **if** $le \neq y$ **then**
23: $t \leftarrow t + 1$, add a new head node to B;
24: **end if**
25: **else**
26: **for** $i \leftarrow l$ to r **do**
27: $T(i, le) \leftarrow t$; // Set this level to current block
28: **end for**
29: B = **UpdateBAT**(B, l, r, le, t, M);
30: $le \leftarrow le + 1$;
31: **end if**
32: **end while**
33: **end for**
34: **return** M;

filling all the area will block the path. The treatment of top-level obstacle in Algorithm 2 is called zero tolerance strategy. The level stops increasing when there is an obstacle on the upper level. This is the best option to avoid any extra filling situation.

The complexity of map makes the evaluation procedure of obstacle on upper level very difficult. For the map with little noise, this evaluation method can satisfy the requirement. For the map with lots of noise, such as the map built with SLAM, the treatment may lead to the early termination of the filling procedure.

To improve the filling result of maps with noises, an alternative plan is applied, called random sampling top judgment. With this method, evaluating the upper level cell of each cell is replaced with evaluating the cells chosen randomly. The evaluation procedure is represented in Algorithm 3. The function Random (l, r) chooses samples randomly between the left and right edge of upper level in n times. The sample amount n is set according to the preset ratio and the sampling interval size. This algorithm should be implemented in the line 19 of Algorithm 2.

https://doi.org/10.1017/S0263574720000818 Published online by Cambridge University Press

Algorithm 3 Random Sampling Top Judgment

Input:
Map M, Current level range l and r, level le;
Output:
Boolean variable indicating whether to stop padding F_{pad} ; sampling amount n;
for $i \leftarrow 1$ to n **do**
  $j = Random(l, r)$;
  if $T(j, le + 1) \neq 0$ **then**
    $F_{pad} \leftarrow True$;
  end if
end for
return F_{pad};

The top judgment is rather complicated. For the current situation, we recommend not to choose too low sampling proportion, and we shall set the area filled by FC algorithm as a high consumption area rather than the obstacle area. With this change, the algorithm is guaranteed to find out the path. In cases where any improper padding cannot be tolerated, it is recommended to choose zero tolerance strategy. Top-level judgment needs to be improved in the future.

4.3. Complexity analysis

For a space size of $m \times n$ on a map, the algorithm shows different execution costs due to the complexity of the map space. Capital O is used here to briefly represent the overall time complexity of the algorithm. Each cell has different visits depending on its location:

1. For any cell in free space, if the cell below is free cell, the cell will be visited only once;
2. For each obstacle cell, if the cell below is an obstacle cell as well, the obstacle cell will be visited only once;
3. For each free cell, if it closes to the edge of obstacles, which has type value bigger than "1", and the cell below is obstacle cell, the filling procedure is activated. The filling procedure will be executed on left and right directions. The execution can be classified into two groups:
   (a) Filling success: Such cells are accessed linearly in the inner loop. Although the outer loop revisited this region again, this does not change the fact that such cells are accessed a constant number of times.
   (b) Filling failure: Due to the control of the outer loop, the algorithm will attempt to expand repeatedly in the horizontal direction of the failed filling area. This phenomenon will occur at the top of the filling block. If there are k cells in the horizontal direction, each cell will be accessed $O(k^2)$ times.

In summary, for a space size of $m \times n$ on a map, the algorithm can achieve $O(m \times n)$ level efficiency when the map is simple. But in the worst case, if there is a considerable number of large concave areas and the filling failure may occur in large quantities, the algorithm complexity is close to $O((m \times n)^2)$. It is well known that the complexity of Dijkstra's algorithm is $O((m \times n)^2)$, and Fibonacci heap optimization A* algorithm is $O((m \times n)lg(m \times n))$. The FC algorithm is always a good choice before executing the path planning algorithm.

4.4. Discussion on the completeness of FC algorithm

FC algorithm cannot guarantee the complete filling of concave areas, because it is limited by three conditions:

1. Each cell satisfies Definition 4;
2. Each round is only allowed to expand on the current horizontal level;
3. The evaluation of top level.

When the filling procedure is finished, the cells on both sides of each level are obstacles, which means $T(l - 1,\ le) = 1$, and $T(r + 1,\ le) = 1$. Otherwise, the cells will be lack of support in horizontal directions. So $(x_1,\ y_1) = (l - 1,\ le)$, $(x_2,\ y_2) = (r + 1,\ le)$ and all cells satisfy Definition 4.

https://doi.org/10.1017/S0263574720000818 Published online by Cambridge University Press

For each cell, the FC algorithm needs an extra limitation compared with Definition 4. For each pair of $(x_1,\ y_1) \in M$ and $(x_2,\ y_2) \in M$, y_1 and y_2 must satisfy $y_1 = y_2 = le$, which le is the level coordinate on y axis. This limitation ensures the elegance and effectiveness. In practical problems, the $(x_1,\ y_1)$ and $(x_2,\ y_2)$ are not only the obstacles but also in the same concave area. It is difficult to distinguish concave areas automatically. Therefore, when the filling process is limited to rise layer by layer, and $y_1 = y_2 = le$, the problem of distinguishing concave regions is simplified, because the leaking will happen if the $(x_1,\ y_1)$ and $(x_2,\ y_2)$ are in different concave areas. Meanwhile, the repeated evaluation of segment $Ax + By + C = 0$ is not needed in the FC algorithm, because all cells of the same level satisfy the requirement of segment.

As we discussed in Section 4.2, FC algorithm must satisfy the evaluation of upper level, because it is proposed to preprocess the concave area in the path planning map instead of the concave polygon convexification. The distribution of random obstacle points in the map cannot be known in advance. In this stochastic system, the number of early terminations caused by the zero tolerance top judgment strategy increases with the number and uniformity of random obstacles. This means that when there are few scattered points, the effect of the FC algorithm is only close to the first two conditions; when the scatter is evenly distributed over the entire map space and the number is large, the effect will greatly reduce. The random sample of top judgment is used to reduce the influence of noise, but it will bring more instability to the system. First of all, under this strategy, it is unclear whether the noise is avoided or not, so the filling effect produced by each round may be different. Secondly, sometimes avoiding obstacles in the sampling process may lead to the failure of stopping the filling procedure at the right time, resulting in overfilling. When the sampling rate increases gradually, the effect of random sampling approaches zero tolerance strategy.

5. Decomposition of Block

Performing path planning on a map that already built blocks requires some extra processing, because the random start point or target point may exist in a block, which is usually simple and does not require process of the map again.

5.1. Building exclusion lists

An exclusion list L_{exc} for blocks to stop the path shall be considered. Assuming that the starting point is located in a constructed block b_{start}, firstly, appending $T_b(b_{start})$ to L_{exc}, and then, this vertex can be used as the starting point of the directed graph traversal algorithm, and the whole block association table can be traversed by breadth-first traversal. The termination condition is not to reach a certain vertex but to search for a vertex with no subsequent node, which means that the free space is touched, or the whole region of the starting point is a closed region. Meanwhile, all the vertices traversed in breadth-first search need to be appended to the L_{exc}.

In Fig. 2(b), assuming that block 2 is the block where the starting point is located, the traversal algorithm accesses vertices 2, 4, 6 and 7 successively. Then appending them to the L_{exc} to restore a path.

The size of block association table is usually small compared with the map data; the time cost of appending process is short. In our practice, it takes less than one thousandth of FC algorithm cost. During the path planning, the blocks in the L_{exc} are not regarded as obstacles, while other blocks which are not added into the L_{exc} will still be treated as obstacles, so that the path planning algorithm can avoid a large number of invalid search areas.

Theorem 2. *Using the decomposition method to build an exclusion list in the map processed by the FC algorithm, the existence of the optimal path solution in the map can be maintained.*

Proof. The theorem is proved in two steps.

Lemma 1. *The blocks filled by FC algorithm do not influence the existence of possible path in the map, except for the blocks of start point and goal point are located and the blocks they are associated with.*

Firstly, the independent concave obstacles in space are considered. The following inference can be easily deduced from Theorem 1 – since the cells in the concave region do not exist in the optimal path, the concave regions excluding where the start point and the goal point are located do not exist

https://doi.org/10.1017/S0263574720000818 Published online by Cambridge University Press

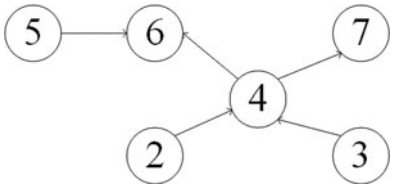


Fig. 4. Regionally connected graph. If the start point is in block 2, the vertices 2, 4 and 6 are the paths having been recovered.

in the optimal path. And whatever actions filling the independent concave obstacles do not affect the existence of the optimal path. Second, the nested concave area needs to be considered, which is likely to form special pathways, as shown in Fig. 3. Such region needs further operations, like top judgment. In Section 4.4, Condition 1 and Condition 2 ensure that the FC algorithm does not fill the cells outside the concave areas. The top judgment guarantees that the path in the concave areas with nested structure will not be blocked.

Therefore, the FC algorithm under zero tolerance top judgment guarantees that the connectivity of each region will not be destroyed.

Lemma 2. *If the start point and the goal point are in padding blocks, the block decomposition procedure will recover their connectivity.*

Assuming that there is a path in the original map, so the start point and the end point are both located in the free space and connected with each other. Now, considering that after applying the FC algorithm, the start point is in the padding area and the goal point is still in free space. The adjacency relationship of Fig. 2(b) and *BAT* is represented as a graph structure in Fig. 4. For b_6 and b_7, their linked lists $B(6)$ and $B(7)$ are empty linked lists, which are also directly connected to the free space. So, when the graph traversal algorithm is executed on the *BAT*, the traverse terminated if there is no subsequent node, which means the method finds a path from the start vertex to free space.

The composition of FC algorithm and block decomposition can maintain the connections among the start point, free area and goal point, so the existence of the optimal path is guaranteed.

5.2. Decompose an established block

Path planning needs not to decompose the blocks that have already been processed, but this approach is feasible. First of all, the gravity direction should be recorded when the block is established. And then, the reverse process of Algorithm 2 should be executed along the gravity direction. Finally, the $T_b(b)$ of the block is obtained, and the exclusion list is established according to the previous method. Other block digestion in contact with this block is similar and will not be described again. The effectiveness of this method will be demonstrated in the experiments.

There is another situation should be considered – if the start point or the goal point of the robot in the map is fixed, the start information and the goal information can be added into the evaluation procedure before building blocks. For example, when judging the upper grid cell, the growing procedure will stop not only when the obstacles are encountered but also when the upper cell is the start point or the goal point.

6. Experiments and Results

Experiments have been carried out on several maps with irregular concave obstacles. In Section 6.1, we present the main information and explanations for the block association table and then give a block disintegrated demonstration. Subsequently, in Section 6.2, we demonstrate the effect of FC algorithm in dealing with the same lab map built by the SLAM under different top judgment strategies. In the end, we compare the effect of A* and APF in path planning on the map processed by FC algorithm with that one on the map that does not processed by FC algorithm.

6.1. Demonstration of padding the concave region

Figure 5(b) is the map which processed by FC algorithm in one direction. Due to the limitation of gravity direction and Condition 2, some concave regions that can be filled in other directions cannot be filled at this time. Figure 5(c) is the applying result of FC algorithm in four different directions

https://doi.org/10.1017/S0263574720000818 Published online by Cambridge University Press

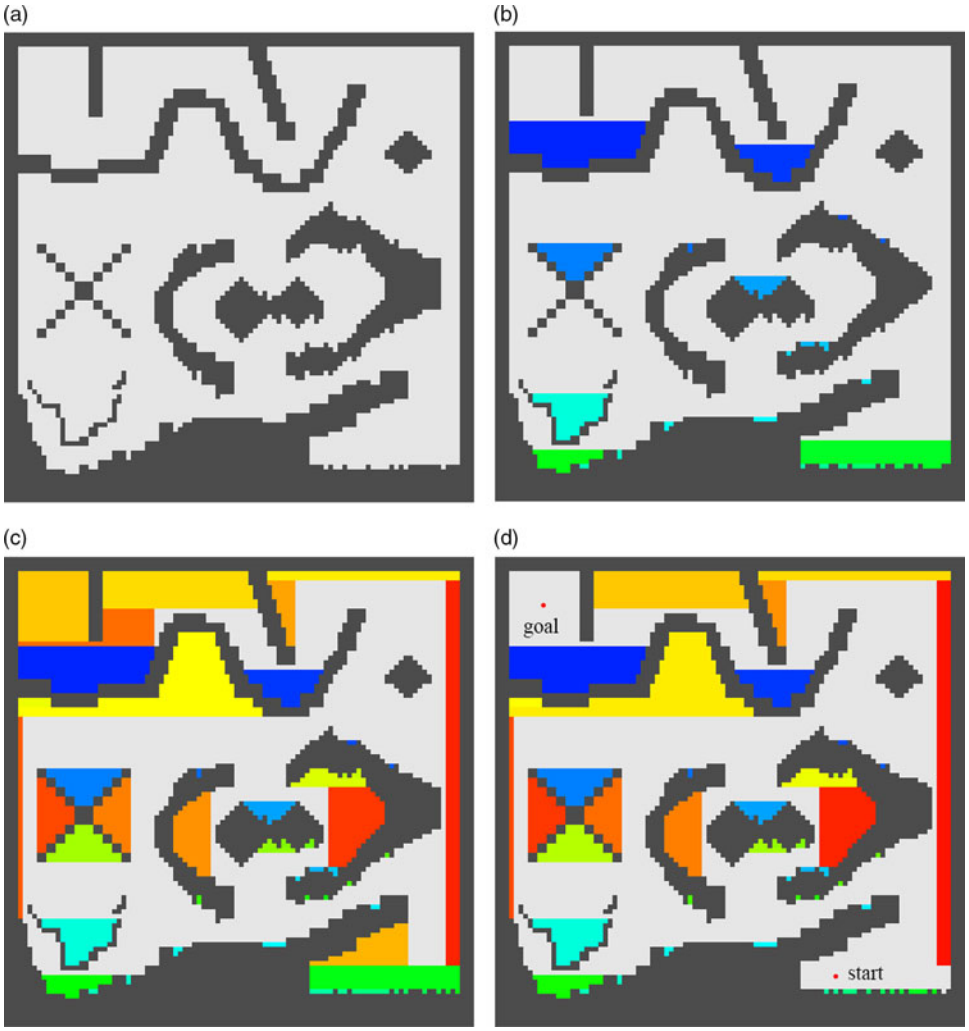


Fig. 5. The blank area in the figure indicates the passable path, the black area represents the original obstacles and the colored grid cells are the area filled by the FC algorithm. Different colors represent different blocks. And the blue–green–red gradient indicates the area that is filled sequentially during the running process of the FC algorithm. (a) A map with many concave areas and irregular obstacle boundaries. (b) Processed by FC algorithm in one direction. (c) Processed by FC algorithm in four directions. (d) Decomposition of block.

repeatedly. It can be seen clearly that most of the concave areas are filled. The remaining blank areas are much smaller, and these free spaces basically form a pathway. Running the path planning algorithm on the processed map will greatly reduce invalid searches. There are 54 blocks in the map. These blocks have different sizes and shapes, which are related to the area filled by each block. Please note that the obstacles in the map are unstructured and their edges are extremely irregular, but FC algorithm can still deal with these areas well.

In the block association table, 21 head nodes have rear nodes, which means that a total of 21 blocks are regarded as obstacles by the blocks built later, while the remaining blocks are independent of each other. In the horizontal green block at the lower right corner of the figure, its lower edge is composed of several individual small blocks. They have the same color because they are constructed sequentially. These jagged obstacle edges tend to cause small blocks, leading to the increase of the number of head nodes in block association table that has rear nodes. This situation can be seen everywhere on the map. It is precisely because blocks are related to each other and build upon each other, and that the FC algorithm completely is insensitive to the edges of objects. In the block association table, the maximum number of rear nodes in the head node is 2, so there will not be too many queries in the process of block disintegration.

https://doi.org/10.1017/S0263574720000818 Published online by Cambridge University Press

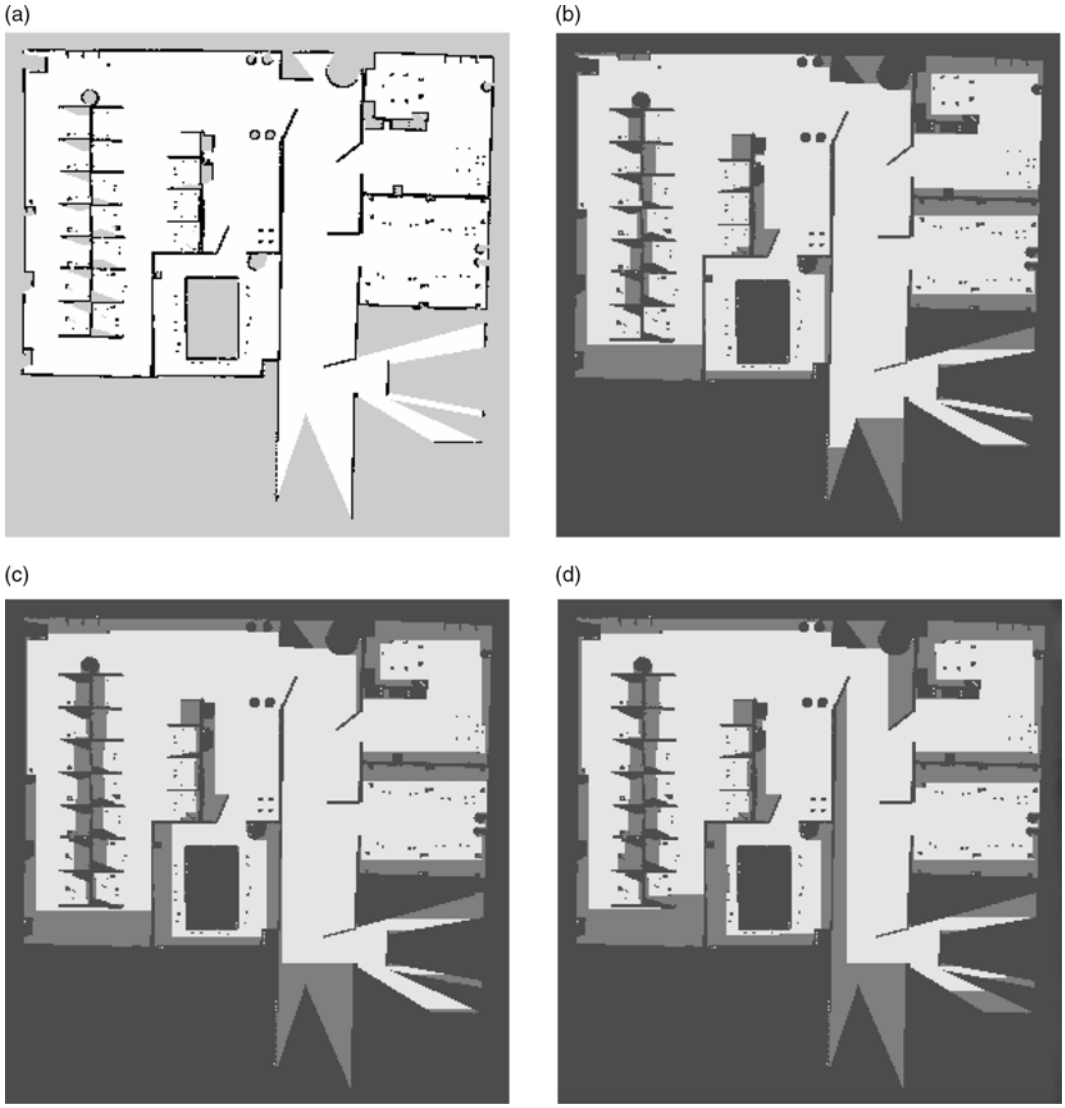


Fig. 6. Demo of real-world map. (a) Lab map constructed by GMapping algorithm. (b) Zero tolerance top check. (c) Sampling from uniformly distributed. (d) Less sampling from uniformly distributed.

The start point and the goal point are marked in Fig. 5(d). Obviously, the blocks where they locate have been removed, and all the blocks blocking the path have also been removed. As we stated in Section 4, the actual path planning does not need to remove blocks blocking the path, just find them and create an exclusion list. The purpose of what we have done here is just to demonstrate this process.

6.2. FC algorithm on real-world map and different strategy of top judgment

Figure 6(a) shows the occupied grid-based map of lab constructed with GMapping algorithm.[29] When the type of cell is converted according to Section 3.1, the occupied threshold is 0.65 and the free threshold is 0.196. In Fig. 6(a), the scatter points are mostly composed of table legs, chair legs, sundries and so on. Similar to the topological composition and multi-resolution grid-based maps, these scatter points can seriously affect the effect of FC algorithm. In Fig. 6(b), as long as there is an obstacle at the top, the expansion will be stopped. Figure 6(c) and (d) uses the method of sampling to obtain a better filling effect than in Fig. 6(b). Among them, Fig. 6(d) has a smaller sampling size with preset ratio 25%, and Fig. 6(c) has a preset ratio 50%. It is clear that the former fills better than the latter, but the small sampling size leads to overfilling and blocks the passage in the lower left

https://doi.org/10.1017/S0263574720000818 Published online by Cambridge University Press

Table I. Multiple times path planning and its cost.

	Expt.	Goal 1	Goal 2	Goal 3	Goal 4	Total
Preprocessing	(a)	0.326 s	0.0 s	0.0 s	0.0 s	0.326s
	(b)	0.081 s	0.116 s	0.049 s	0.917 s	1.163 s
	(c)	278	367	202	1407	2254
No preprocessing	(a)	0.0 s	0.0 s	0.0 s	0.0 s	0.0 s
	(b)	0.082 s	0.200 s	0.057 s	1.915 s	2.254 s
	(c)	278	510	217	2146	3151

(a) Time cost of FC algorithm.
(b) Time cost of A* algorithm.
(c) The close list length of A*.

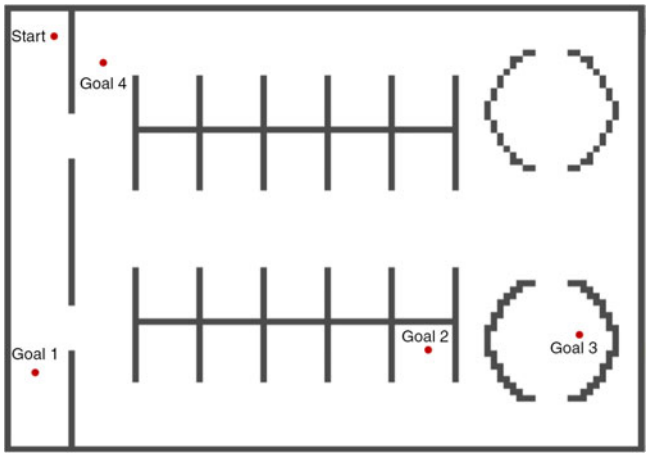


Fig. 7. Simplified office map with one starting point and several target points.

corner of the map. This is a typical example of improper padding. If using a low sampling ratio, like Fig. 6(d), it is better to set the padding area as a high consumption area.

6.3. The effect of FC algorithm on path planning

The experiment is run with Ubuntu16.04, Python 3.6 and Intel i7-7700 CPU. Only the original A* algorithm is used. The heuristic function[1] $h(x)$ is the Euclidean distance, and in each step of the planning, the robot can move around eight grid cells.

Figure 7 is a multi-target point map, showing the target points that the mobile robot sequentially passes during the entire navigation process. As shown in Table I, after the map is preprocessed by FC algorithm, the robot spends 1.418 s in the whole process, while the robot without preprocessing spends 2.254 s in the whole process. In the single-step path planning, the cost time between goal 3 and goal 4 is the largest. The path planning on the unprocessed map takes 1.915 s, and the close list has a total of 2146 nodes. After processing by FC algorithm, the path planning algorithm takes 0.917s, and the close list has 1407 nodes. The number of nodes in the close list is less than two times of the difference, but the time consumption is more than two times. This is because in the A* algorithm, it needs to repeatedly look for the node with the minimal $f(x)$ in the open list, which involves sorting. The additional nodes added by concave region not only increase the number of nodes traversed but also cause extra time for sorting.

When a large number of concave regions are encountered in the process of path planning, as shown in Fig. 8, the time cost for path planning will increase sharply. As shown in Table II, A* runs on the both maps, and the time difference was nearly three times. We use FC algorithm in Fig. 8(b), and the concave areas in the figure are filled. In addition, we compare the differences between artificial potential field method and Artificial Potential Field (APF) method with virtual obstacle[26] in dealing with concave areas. In Table II, the time consumption of APF method and Artificial Potential Field with Virtual Obstacle (APFVO) in path planning on preprocessed map are almost the same,

https://doi.org/10.1017/S0263574720000818 Published online by Cambridge University Press

Table II. Worst case path planning.

	FC	A*	APF	APFVO
Preprocessing	0.328 s	1.332 s	2.436 s	2.437 s
No preprocessing	0 s	4.003 s	Local minimum	6.704 s

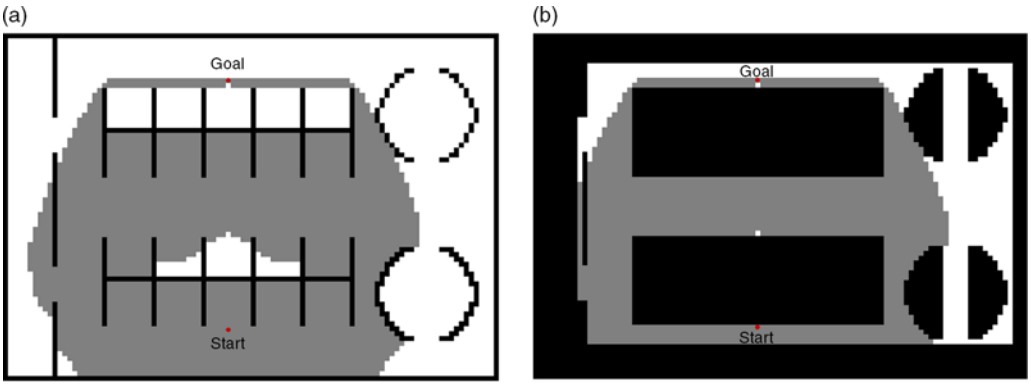


Fig. 8. Comparison of office environment with preprocessing and without preprocessing, the gray area in figure is the node traversed during the running of the A* algorithm and stored in the close list. (a) The unprocessed map and (b) the map processed by FC algorithm.

because all areas in the map that may cause local minimum have padded, and APFVO has no effect. On the unprocessed map, the original APF algorithm shows local minimum and the planning fails. Although APFVO completes the navigation, it takes a lot of time, because the concave obstacle area in the map causes a large number of local minimum points. In the calculation of gradient descent, it is necessary to repeatedly use the virtual obstacle to escape the minimum point. Besides that, the path obtained by APFVO is not the optimal path, but it can help the robot move along the edge of obstacles. So basically, the FC algorithm is aimed to avoid the invalid search encountering in the path planning algorithm in advance. In contrast, the processing time in the FC algorithm is very short, so preprocessing seems to be always a good choice, let alone the situation to run repeatedly on the same map.

Although we only adopt the A* algorithm as the path planning algorithm, the preprocessing process in this paper has the same effect on all path planning algorithms.

7. Conclusion and Future Work

In this paper, we propose a novel approach to deal with the concave regions in 2D grid-based map, called FC method, which is easy to comprehend and its efficiency is very well. The algorithm is insensitive to the shape of the obstacles. As shown in the experiments, the FC algorithm can fill the concave regions well in the map, greatly improving the efficiency of the path planning algorithm. By building block association table and exclusion list, the map needs not to reprocess when the start point and the goal point are changed, so the FC algorithm is a good choice when we need to repeatedly perform path planning on a map.

We mainly explain the concept and basic implementation of FC algorithm in this paper. In fact, there are still some further improvements of the algorithm, so that it can adapt to more environments and improve its performance, such as combining the processes of several gravitational directions and merging small blocks with larger adjacent blocks to further simplify the complexity. In addition, we can also execute FC algorithm in four diagonal directions, which can build more blocks and block more areas. FC algorithm can be used repeatedly, which can cover more missed areas. In some maps, even a path structure close to the topology map can be directly formed after such processing, but this is not stable, and more design work should be done in the future.

We hope this method can be extended to methods such as navigation mesh in the future and can be combined with topological map and other methods to greatly simplify the map structure, or it can build the topology map through blocks alone as the basic unit.

https://doi.org/10.1017/S0263574720000818 Published online by Cambridge University Press

Acknowledgements
The authors thank Yunfei Chen, Siyuan Gong, Jingting Qiu, Xian Li, Ge Ding and Sichen Pan for discussing the content of the paper and providing valuable insights. The authors thank Professor Ge Yang for his help in the translation and writing of the paper.

Funding
The work is financially supported by the National Natural Science Foundation of China (61502118).

References

1. P. E. Hart, N. J. Nilsson and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Trans. Syst. Sci. Cybern.* **4**(2), 28–29 (1972).
2. J. J. Kuffner and S. M. Lavalle, "RRT-Connect: An Efficient Approach to Single-Query Path Planning," *IEEE International Conference on Robotics and Automation* **4**, 995–1001 (2002).
3. N. M. Amatoc and L. K. Dale, "OBPRM: An Obstacle-Based PRM for 3D Workspaces," *Found of Robotics* (1998).
4. E. Rimon and D. E. Koditschek, "Exact robot navigation using artificial potential functions," *IEEE Trans. Robot. Autom.* **8**(5), 501–518 (1992).
5. M. Hoy, A. S. Matveev and A. V. Savkin, "Algorithms for collision-free navigation of mobile robots in complex cluttered environments: a survey," *Robotica* **33**(3), 463–497 (2015).
6. S. Thrun, "Learning metric-topological maps for indoor mobile robot navigation," *Artif. Intell.* **99**(1), 21–71 (1998).
7. J. Carsten, A. Rankin, D. Ferguson and A. Stentz, "Global Path Planning on Board the Mars Exploration Rovers," *IEEE Aerospace Conference* (2007) pp. 1–11.
8. M. Wang and J. N. K. Liu, "Fuzzy logic-based real-time robot navigation in unknown environment with dead ends," *Robot. Auto. Syst.* **56**(7), 625–643 (2008).
9. M. D. Berg, M. V. Kreveld, M. Overmars and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications* (Springer Publishing Company, Incorporated, 2000).
10. R. H. T. Chan, P. K. S. Tam and D. N. K. Leung, "Robot Navigation in Unknown Terrains via Multi-resolution Grid Maps," *1991 International Conference on Industrial Electronics, Control and Instrumentation* (1991) pp. 1138–1143.
11. E. Einhorn, C. Schröter and H. Gross, "Finding the Adequate Resolution for Grid Mapping - Cell Sizes Locally Adapting on-the-Fly," *2011 IEEE International Conference on Robotics and Automation* (2011) pp. 1843–1848.
12. Y. Chen, W. Shuai and X. Chen, "A Probabilistic, Variable-Resolution and Effective Quadtree Representation for Mapping of Large Environments," *2015 International Conference on Advanced Robotics (ICAR)* (2015) pp. 605–610.
13. S. Kambhampati and L. S. Davis, "Multiresolution path planning for mobile robots," *IEEE J. Robot. Autom.* **3**(2), 135–145 (1986).
14. R. A. Finkel and J. L. Bentley, "Quad trees a data structure for retrieval on composite keys," *Acta Informatica* **1**(2), 1–9 (1974).
15. M. Liu, F. Colas and L. Oth, "Incremental topological segmentation for semi-structured environments using discretized GVG," *Auto. Robots* **38**(2), 143–160 (2015).
16. E. G. Tsardoulias, A. T. Serafi, M. N. Panourgia, A. Papazoglou and L. Petrou, "Construction of minimized topological graphs on occupancy grid maps based on GVD and sensor coverage information," *J. Intell. Robot. Syst.* **75**(3), 457–474 (2013).
17. A. K. Jain, M. N. Murty and P. J. Flynn, "Data clustering: A review," *ACM Comput. Surv.* **31**(3), 264–323 (1999).
18. Y. Zhou, S. Yu, R. Sun, Y. Sun and L. Sun, "Topological Segmentation for Indoor Environments from Grid Maps Using an Improved NJW Algorithm," *2017 IEEE International Conference on Information and Automation (ICIA)* (2017) pp. 142–147.
19. M. Liu, F. Colas and R. Siegwart, "Regional Topological Segmentation Based on Mutual Information Graphs," *2011 IEEE International Conference on Robotics and Automation* (2011) pp. 3269–3274.
20. B. Kaleci, Ç. M. Senler, O. Parlaktuna and U. Gürel, "Constructing Topological Map from Metric Map Using Spectral Clustering," *2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)* (2015) pp. 139–145.
21. M. Kallmann, "Shortest Paths with Arbitrary Clearance from Navigation Meshes," *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2010) pp. 159–168.
22. C. Barber, D. P. Dobkin and H. Huhdanpaa, "The quickhull algorithm for convex hulls," *ACM Trans. Math. Software* **22**(4), 469–483 (1996).
23. R. L. Graham, "A reevaluation of an efficient algorithm for determining the convex hull of a finite planar set," *Inf. Process. Lett.* **7**(1), 53–55 (1978).
24. F. P. Preparata and S. J. Hong, "Convex hulls of finite sets of points in two and three dimensions," *Commun. ACM* **20**(2), 87–93 (1977).

https://doi.org/10.1017/S0263574720000818 Published online by Cambridge University Press

25. M. G. Park and M. C. Lee, "Experimental Evaluation of Robot Path Planning by Artificial Potential Field Approach with Simulated Annealing," *Proceedings of the 41st SICE Annual Conference. SICE 2002* **4**, pp. 2190–2195, (2002).
26. M. C. Lee and M. G. Park, "Artificial Potential Field Based Path Planning for Mobile Robots Using a Virtual Obstacle Concept," *Proceedings 2003 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM 2003)*. **2**, 735–740 (2003).
27. C. Zhang, C. Hu, J. Feng, Z. Liu, Y. Zhou and Z. Zhang, "A self-heuristic ant-based method for path planning of unmanned aerial vehicle in complex 3-D space with dense U-type obstacles," *IEEE Access.* **7**, 150775–150791 (2019).
28. H. Chen, Y. Ji and L. Niu, "Reinforcement learning path planning algorithm based on obstacle area expansion strategy," *Intell. Serv. Robot.* **13**(2), 289–297 (2020).
29. G. Grisetti, C. Stachniss and W. Burgard, "Improved techniques for grid mapping with Rao-Blackwellized particle filters," *IEEE Trans. Robot.* **23**(2), 34–46 (2007).

https://doi.org/10.1017/S0263574720000818 Published online by Cambridge University Press