A Fast Bi-Directional A* Algorithm Based on Quad-Tree Decomposition and Hierarchical Map

ZHOU YIJUN, XI JIADONG, AND LUO CHEN

School of Mechanical Engineering, Southeast University, Nanjing 211189, China

Corresponding author: Luo Chen (chenluo@seu.edu.cn)

This work was supported in part by the Six Talent Peaks Project in Jiangsu Province under Grant GDZB-002.

ABSTRACT Although the popular path-planning algorithms based on graph-search such as Dijkstra and A* guarantee to find the optimal path, the search time will increase rapidly with the growth of map size. Those traditional graph-search based algorithms ignore the distribution of obstacles, causing much time wasted on exploring unrelated regions. A fast bidirectional-A* algorithm based on hierarchical map (QH-A*) was proposed in this paper with high real-time performance. QH-A* consists of two parts: offline pre-processing and online search. The offline pre-processing of QH-A* comprises map decomposition and construction of hierarchical map. In the map decomposition step, a modified quad-tree decomposition method was used to recursively divide the initial map into different regions. Then boundary nodes of each region were extracted. During the construction of hierarchical map, a kind of hierarchical map was defined and precomputation was done to construct the hierarchical map. In the online-search step, a revised bidirectional-A* specific to hierarchical map was proposed to finish the path-planning task. Experiments and theories proved that QH-A* can always find the shortest path. In addition, the search area required by QH-A* was 70%-80% less than that of A*, which verified high real-time performance of QH-A*.

INDEX TERMS Quad-tree decomposition, bi-directional A*, hierarchical map, real-time path planning.

I. INTRODUCTION

In the field of real-time strategy games and mobile robots, path planning is a basic but important problem, as well as a popular direction of research. An effective path planning algorithm ensures that the feasible path can be found. In most cases, the shortest planned path is required to improve the efficiency of the robot. In addition, the algorithm should not consume too much computing resources nor occupy too much memory.

The classical methods to solve the shortest path problem include Dijkstra [1] and A* [2] algorithms. Dijkstra algorithm is famous for its stability and optimality. A* algorithm improves Dijkstra algorithm by introducing a heuristic function, which estimates the true distance between the searched node and the target node. This goal-oriented technology can effectively speed up the search process and find the shortest path faster. The A* algorithm has become the mainstream algorithm for path planning due to its excellent performance. However, A* algorithm is still not efficient enough as the size of map grows, some technologies has been proposed to speed up A* algorithm.

New technologies can be divided into two categories: with and without pre-processing. Methods without preprocessing is mainly a variant of the A* algorithm, such as LPA*, Theta-A*, Bi-A*. LPA* [4] is an incremental variant of A* algorithm, it is designed to solve multiple source-target path planning problem, LPA* restores the corresponding path and distance information while solving each pair of source-target point paths, and uses them in subsequent path planning. The efficiency of the algorithm increases significantly as the algorithm progresses.

Theta-A* expands the search direction compared to traditional A*, eliminates the angle restriction, and takes shorter time to plan a path. Bi-A* [5] introduced a bi-directional search technology based on the A* algorithm, that is, searching simultaneously from the source and the target at the same time, thereby efficiency is improved. In addition, the combination of A* algorithm and sliding window algorithm [6] can efficiently avoid obstacles.

The key technology of those methods with pre-processing is the compression and extraction of information, such as TDA*, "Jump Point", Swamp, HEPV, Multi-level Graph.
TDA* pre-computes and stores the shortest distance between any two points on the graph, and uses its value as the heuristic function of A*, which can accurately find the shortest path, but its huge amount of pre-calculation and storage requirements limit its applications. A differential TDA* method [9] was proposed. It only needs to calculate the distance between a part of the source-target pairs, then the distance between all the other pairs can be calculated from the combination of these distances. This differential version reduces the amount of calculation and memory requirement. “Jump Point” [6] method was proposed by Danie and Alban. “Jump Point” are special nodes in the graph which has limited directions of expanding, the authors proved that any path can be connected by several such nodes. In practical applications, A* algorithm will spend a lot of time exploring those locally but not globally optimal areas. The idea of Swamp algorithm [7] is to find these “redundant” areas before searching, and only search these areas when the algorithm starts or ends in these areas, so that the search algorithm only searches these areas under limited conditions, avoiding falling into “redundant” areas. Jing et al. [11] proposed the HEPV method considering the huge data and high real-time requirements of the intelligent transportation system, which is divided into two parts: offline and online. The offline part uses a plane segmentation algorithm to extract map information layer by layer to form a hierarchical map. Every time a layer is extracted, the number of nodes on the hierarchical map is greatly reduced, and the coding method in [11] is used to encode and store the map information. Sung won et al. further developed the hierarchical map theory and proposed Multi-Level Graph method [12], which systematically explained how to generate a hierarchical map from the original map. The hierarchical map condenses information of the original map, and the search efficiency on the hierarchical map is higher. For more algorithms based on hierarchical maps, see literature [13]–[17].

Algorithms that rely on preprocessing are usually better in real-time performance than algorithms without preprocessing. However, many methods with preprocessing have disadvantages such as long calculation time and large memory space required. Algorithms that rely on preprocessing are usually better in real-time performance than algorithms without preprocessing. However, many preprocessing methods have disadvantages such as long calculation time and large memory space required. In this paper, we proposed QH-A* (Bi-directional A* algorithm based on quad-tree decomposition and hierarchical map). QH-A* applies the hierarchical map theory to grid maps, uses an improved quad-tree decomposition algorithm to decompose the map into different regions, extracts the boundary nodes of the regions, builds hierarchical map; A bi-directional A* algorithm is complemented on the built hierarchical map. Experiments have proved that compared to the A* algorithm, QH-A* can effectively reduce the search area and improve the search efficiency, and the amount of pre-calculation required is also controlled within a reasonable range. The flowchart of the QH-A* algorithm is shown in Figure 1. Previous work of other authors are listed in the dashed box, in the bold box are our original work.

FIGURE 1. Flow chart of QH-A* algorithm.

II. PROBLEM DEFINITION
Given a graph \( G = (V, E, C) \), \( V \) is the set of nodes, \( E \) is the set of edges between nodes, and \( C \) is the corresponding cost of traveling through each edge. A path \( p \) is formed as an ordered sequence of nodes, \( p = < n_1, n_2, \ldots, n_k > \). The shortest path \( sp \) which has \( |sp| = \min{|p|} \).

Grid map is a widely used kind of graph, which can be generated by cell decomposition method. In a grid map, nodes are represented by neighboring grids, the cost between each neighboring grid is always a unit value. A grid always has value 0 means free to explore, otherwise it’s occupied by obstacle.

III. OFFLINE PREPROCESSING
Online preprocessing of QH-A* includes two steps: firstly, we use map decomposition method to divide the original map into several regions. Secondly, we define and construct a so-called “hierarchical map” based on the decomposition. All of these two steps will be explained in detail in this section.

A. MAP DECOMPOSITION METHOD
1) QUAD-TREE AND FRAMED QUAD-TREE DECOMPOSITION
Quad-tree decomposition [17] recursively divides the map into four sub-maps of same size (shown in Figure 2), this method is widely used in image processing such as collision detection and GPS. Combined with path planning algorithms, the center node of each sub-map are chosen to replace all the nodes inside, which makes rapid path planning possible. However, the quality of the path obtained by this method is not high.

The framed quad-tree decomposition [18] addressed the suboptimal problem by adding grids of the highest resolution around the boundary of region. A comparison of quad-tree and framed quad-tree decomposition is shown in Figure 3. However, the framed quad-tree decomposition suffers from
enlarged amount of nodes, so we modified the original method by controlling the process of division according to obstacle density within a sub-region, if the obstacle density is lesser than a threshold, no further division will happen.

2) TREE STRUCTURE OF MAP DECOMPOSITION
The improved quad-tree decomposition recursively divides the original map into four sub-maps of same size, which naturally forms a tree structure: each sub-map is a node of the tree, one parent node can have four children nodes. We call this kind of tree as decomposition tree.

Definition 1: A region \( r = (x, y, \text{level}) \) is a tuple, in which \( x, y \) represent the x/y coordinate of the top-left node in the area. For example, the original map has level 0, and it generates four sub-maps of level 1.

The set of all the areas is denoted as \( R \), if there is a grid map of size \( n \times n \), and \( n = 2^j, j \in \mathbb{N}^+ \), then size of an area with level \( L \) is \( L \times L, L = n \times 2 \). Level.

Definition 2: Each area (if permitted) can be divided into four sub-areas of same size, mark this area as their common parent, define a map from an area to its parent area \( f: R \rightarrow R \), the parent area of the original map is itself. Given a region \( r \), \( f(r) \) is denoted as the first order parent of \( r \), \( f^2(r) = f(f(r)) \) as the second order parent of \( r \), and so on. The tree structure is shown in Figure 4, and each area is a node of the tree.

As each area is a node of the decomposition tree, and nodes of a tree have the so-called smallest common parent node, we can define the smallest common parent area similarly.

Definition 3: Given two nodes \( x \) and \( y \), their smallest common parent area is the smallest area that contains both of these two nodes, denoted as:

\[
mcr(x, y) = \{ r_m | x, y \in r, \text{level}_m = \max(\text{level of } r) \}
\]

An example of smallest common parent area is shown in Figure 5. The source/target nodes are marked in yellow, and the smallest common parent area is indicated with the blue dash box.

3) EXTRACT BOUNDARY NODES
The last step of the map decomposition method is to extract boundary nodes of each region. Final result of the decomposition method applied to a sample map is shown in Figure 6.

B. CONSTRUCTION OF HIERARCHICAL MAP
The original map is a graph \( G = (V, E, C) \), a hierarchical map is also a graph, which can be denoted as \( G^h = (V^h, E^h, C^h) \). The node set \( V^h \) is the set of boundary nodes extracted in the map decomposition step. Denote the boundary nodes set of a region \( r \) as \( B(r) \), we have \( V^h = \bigcup_{r \in R} B(r) \).

In order to determine edge set \( E^h \) of a hierarchical map, we first divide the elements in \( E^h \) into two categories: level-edge and intra-edge, then we will explain why these two kinds of edge are necessary to find the optimal path.

1) LEVEL-EDGE
Definition 4: Given a region \( r \) and its boundary node set \( B(r) \), parent region \( f(r) \), boundary node set of its parent...
region \( B(f(r)) \). The source node of a level-edge on the area \( r \) is the element in \( B(r) \), and the target node is the element in \( B(f(r)) \). In addition, there is at least one feasible path on the original graph \( G \) from that source node to the target node. The length of a level-edge is defined as the length of the shortest path among all such paths. Formal definition of level-edge is:

\[
LE|r = \{ (\text{start, dest}) | \text{start} \in B(r) \land \text{dest} \in B(f(r)), \exists p | G = < \text{start}, \ldots \text{dest} > \}
\]

\[
\text{length}(LE|r) = \min \{ |p| \}
\]

\( p \) represents paths in the original graph \( G \).

**Algorithm 1 Compute_Level_Edge**

**Input:** \( R \ f \)  
**Output:** level_edge  

1: for region \( \in R \) do  
2: father_region \( = f(\text{region}) \)  
3: \( Bf = B(father\_\text{region}) \)  
4: for node \( \in Bf \) do  
5: for other_node \( \in Bf \) do  
6: \( < \text{path, distance}> = \text{compute\_path\_reduced} \) (node,other_node,father_region)  
7: if path exists then  
8: new_edge.start \( \leftarrow \) node  
9: new_edge.end \( \leftarrow \) other_node  
10: new_edge.path \( \leftarrow \) path  
11: new_edge.distance \( \leftarrow \) distance  
12: Level_edge.add(new_edge)  
13: end if  
14: end for  
15: end for  
16: return level_edge

**Algorithm 2 Compute_inter_edge**

**Input:** \( R \) f map  
**Output:** Inter_edge  

1: for current_region \( \in R \) do  
2: for region \( \in R \) do  
3: if \( f(\text{region}) = \text{current\_region} \) then  
4: Add region to child_region  
5: end if  
6: end for  
7: for node \( \in B(\text{child\_region}) \) do  
8: for other_node \( \in B(\text{child\_region}) \) do  
9: \( < \text{path, distance}> = \text{compute\_shortest\_path} \) (node,other_node)  
10: if path exists then  
11: new_edge.start \( \leftarrow \) node  
12: new_edge.end \( \leftarrow \) other_node  
13: new_edge.path \( \leftarrow \) path  
14: new_edge.distance \( \leftarrow \) distance  
15: Inter_edge.add(new_edge)  
16: end if  
17: end for  
18: end for  
19: end for  
20: return inter_edge

**FIGURE 6.** The smallest common parent area.

**FIGURE 7.** Example of level-edge, red grids represent boundary nodes of certain region, blue grids represent boundary nodes of its parent region.

2) **INTRA-EDGE**

**Definition 5:** Given a region \( r \) (represented by tuple), denote set of its child region (if exists) as \( cr = \{ r' | f(r') = r, r' \in R \} \). \( B(cr) \) refers to set of boundary nodes of \( cr \), both the source node and the target node of an intra-edge on region \( r \) is the element of \( B(cr) \). There is at least one feasible path from the start node of the edge to the end node of the edge on the original graph. The length of an intra-edge is equal to the length of the shortest path among all such paths.

\[
IE|r = \{ (\text{start, dest}) | \text{start} \in B(cr) \land \text{dest} \in B(cr), \exists p | G = < \text{start}, \ldots \text{dest} > \}
\]

\[
\text{length}(IE|r) = \min \{ |p| \}
\]

Let \( E^h = IE|r \cup \bigcup_{r \in R} LE|r \). A complete description of the hierarchical map can be given by combining the relevant definitions in the previous section.

3) **ROLE OF LEVEL-EDGE AND INTRA-EDGE**

According to the definition above, level-edge is the set of edge which connects one region’s nodes and its parent region’s nodes, whereas intra-edge is the set of edge which connects nodes of the same region. We will prove that any path in the original map can be represented by a combination of intra-edge and level-edge in the hierarchical map.
among which we have:

\[ p = p_1 \cup p_m \cup p_d \]

Lemma 1: Given a source node “start” and a target node “dest” (short for “destination”), denote \( sr \) as the region of source node, \( ls \) is the level of \( sr \), denote \( dr \) as the region of target node, \( ld \) is the level of \( dr \). If \( sr \neq dr \) and there exists at least one path from source node to target node, then any such path \( p = < n_1, n_2, \ldots, n_k >, n_1 = start, n_k = dest \) must have the following form:

\[ p_1 = \langle start, \ldots, n_1, \ldots, n_k \rangle, \quad n_i \in B(\hat{f}(sr)) \]

\[ s = lca - ls + 1 \]

\[ p_d = \langle dest, \ldots, n_0, \ldots, n_1, \ldots, n_d \rangle, \quad n_i \in B(\hat{f}(dr)) \]

\[ d = lca - ld + 1 \]

\[ p_m = \langle n_0, \ldots, n_k \rangle, \quad n_i \in B(mcr) \]

Proof: On the other hand, robot starts from the start region \( sr \), pass through the first order parent of \( sr \), and repeat this process until it reaches the smallest common region \( mcr \). On the other hand, robot passes through the smallest common region to one of its child region, and continues to move until it finally reaches the \( dest \) region.

IV. ONLINE SEARCH

The task of online search is to apply a search algorithm to the hierarchical map to obtain the final path. In this section we first give a brief review of the traditional A* algorithm, then we propose a new kind of bi-directional A* algorithm based on our hierarchical map. Finally, we prove the optimality of this method, and analyze the time complexity of this search method.

4) BASIC METHOD

A* uses an evaluation function \( d(u) = g(u) + h(u) \) to determine which node to expand at first, the node with the minimum \( d(v) \) has the highest priority. \( g(u) \) represents distance from start node to node \( u \) and \( h(u) \) represents the actual distance from \( u \) to \( dest \) node involving evaluation of an unknown distance. We can make sure that A* algorithm finds the optimal path as long as \( h(u) \) is less than the true distance. A common heuristic function is \( L_1 = |x_1 - x_2| + |y_1 - y_2| \).

The bi-directional A* algorithm consists of forward search and reverse search. Taking forward search as an example, forward search maintains an openlist and a closelist. At the very beginning \( g(u) = \infty, \forall u \in V \), let \( g(start) = 0 \) in the initialization step.

1) Set the node with minimum \( d \)-value as current node and add it to openlist.

2) Find all the neighboring nodes of current note and denote them as neighbor(current), add all nodes \( \notin \) openlist or closelist to openlist.

3) Update distance function:

\[ g(u) = g(current) + C^h(current, u) \]

4) Add current node to closelist

Repeat procedure 1-4 until \( dest \) node is in the closelist, then \( d(dest) \) is the length of the shortest path. For bi-directional A* algorithm, the forward search and the reverse search each maintain a closelist, denoted as forward_list and backward_list. The algorithm will terminate if \( forward_list \cap backward_list = \omega \) and the shortest path length is \( g_f(\omega) + g_b(\omega) \) at that time. A bi-directional search procedure is shown in Figure 9.

VOLUME 9, 2021
Proof: Denote the original map as $G$, given a destination, the corresponding hierarchical map is $G^h$. Denote the shortest path obtained in $G$ as $SP(G)$, and the one obtained in $G^h$ as $SP(G^h)$. It is equal to prove that $SP(G)$ and $SP(G^h)$ have the same length. On the one hand, path obtained in $G^h$ is still one path in $G$, thus we have $\text{length}(SP(G^h)) \geq \text{length}(SP(G))$. On the other hand, according to Theory1, any path in $G$ can be divided into three parts: $P(G) = P_s + P_m + P_d$. According to the optimal theory, $SP(G) = SP_s + SP_m + SP_d$, i.e., the shortest path is the sum of the three shortest paths. These three paths can be composed entirely of edge set in the hierarchical map: $E^h$, any shortest path in $G$ can be divided into three shortest paths in $G^h$. Thus we have $\text{length}(SP(G)) \geq \text{length}(SP(G^h))$. In summary, we have $\text{length}(SP(G)) = \text{length}(SP(G^h))$.

6) TIME COMPLEXITY

The time complexity of bi-directional $A^*$ has the same order of magnitude as $A^*$. Given a graph $G = (V, E)$, it is easy to quantify the time complexity of $A^*$. This section aims to evaluate the time complexity of $A^*$ search on hierarchical map. The time complexity of $A^*$ depends on the data structure used. Given a graph $G = (V, E)$, the time complexity of $A^*$ search using the Fibonacci heap is $O((|E| + |V| \log |V|))$, where $|E|$ and $|V|$ are the size of edge set and node set, respectively.

Lemma 3: The time complexity of $A^*$ search on the hierarchical map generated by QH-A* is $O(n^2)$. 

Proof: The hierarchical map used by QH-A* uses the boundary nodes of each region as the basic unit of search. For a region of size $L \times L$, the number of boundary nodes is at most $4(L-1)$. Without loss of generality, suppose the size of the map is $n \times n$, $n = 2^l$. The minimum area size is specified as $2 \times 2$, and the generated hierarchical map level is up to $j-1$, $G_{j-1}^h = (V_{j-1}^h, E_{j-1}^h)$. Given a source node and a target node, in the worst case, the smallest common region of the two is the original map, and the maximum number of nodes to be searched is $2 \sum_{i=1}^{j-1} 4(n/2^i - 1) = \sum_{i=1}^{j-1} 2 \times 3^i - \sum_{i=1}^{j-1} 2^i$. Remove repeatedly searched nodes of number $\sum_{i=2}^{j-1} 2 \times n/2^i - 1$, the total number of nodes is at most $|V^h| = \sum_{i=1}^{j-1} 2^{i-1} \times n - \sum_{i=1}^{j-1} 2^i - \sum_{i=1}^{j} 2 \times n/2^i - 1$.

$$\sum_{i=1}^{j-1} 2^{i-1} \times n = 2j \sum_{i=1}^{j-1} 2^{i-1} = 2^{j+1} - 2 = 2^j - 1 = n - 1 \quad (1)$$

$$\sum_{i=2}^{j-1} 2 \times n/2^i - 1 = 2 \sum_{i=2}^{j} 2^{i-1} - 1 = n - 4 \quad (2)$$

Thus total number of node satisfies $|V^h| < 8n - 16 - n + 1 - n + 4 < 6n$. The size of $E^h$ is the sum of intra-edge and level-edge with the total number $|E^h| < C_n^2 \sim O(n^2)$. Time complexity is:

$$O(n^2 + n \log n) = O(n^2) \quad (4)$$

It is easy to prove that the time complexity of the $A^*$ search on the original map $G$ is $O(n^4 \log n)$, so search on the hierarchical map generated by QH-A* has the advantage in terms of time complexity.

V. EXPERIMENTS

In this section experiments were conducted to test the performance of QH-A*, two performance indices were chosen: (a) preprocessing time, since a hierarchical map was proposed to reduce the problem of enlarged number of nodes, comparison was made between preprocessing amount on the original map and hierarchical map (b) search space, the most intuitive demonstration of efficiency.

A. PREPROCESSING TIME

The time consumed by the map decomposition algorithm is negligible. The main time consumption comes from calculation of the level-edge and intra-edge, which involves storing some shortest paths between nodes in advance. This section lists the preprocessing methods used in existing algorithms for comparison with the preprocessing method used by QH-A*. For TDA* [10], the shortest path between any two nodes on the map is pre-calculated and used in the heuristic function of $A^*$. For a map of size $n \times n$, the time complexity is $O(n^4 \log n)$. Another different version of TDA* predetermines $k$ “landmark” nodes to reduce the amount of calculation, the time complexity is $O(n^3 \log n)$. For HPA* [22], the algorithm uses the simplest way of abstraction. It divides the map into square regions of same size, and extracts boundary nodes of each region. In order to avoid large amount of pre-computation, the HPA* algorithm discards many of the boundary nodes, and then calculates the shortest distance between the rest, thus the final path is not optimal. m-A* [24] uses the PFR-RDP method to decompose the original map into several areas, and calculates multi-scale information in the pre-processing step, the authors proposed an up-bottom fusion algorithm to compute shortest paths, which has the time complexity of $O(n^3)$.

QH-A* method also has the map decomposition step as m-A* and other similar algorithms, however, preprocessing is done on the hierarchical map instead of the original map. The amount of preprocessing is proportional to the size of edge set, that is, $\propto |E^h|$. If the node size $|V^h| = n$, in the worst case, the shortest paths between each pair of nodes need to be calculated with total number of $C_n^2$. For comparison, one can compare the amount of calculation based on the worst case. The number of edges needs to be calculated in QH-A* is the sum of level-edge and intra-edge, and we define the ratio:

$$\rho = \frac{|\text{level - edge}| + |\text{intra - edge}|}{C_n^2} \times 100\% \quad (5)$$
TABLE 1. Pre-calculation on map of size 32 × 32.

| Boundaries | 32x32          |
|------------|----------------|
| 32 nodes   | 325            |
| 32 Level-  | 330            |
| 32 Intra-  | 308            |
| 32 Total   | 317            |
| 32 Worst-case ratio | 307 |
| 32          | 311            |
| 32          | 335            |

TABLE 2. Pre-calculation on map of size 64 × 64.

| Boundaries | 64x64          |
|------------|----------------|
| 64 nodes   | 853            |
| 64 Level-  | 873            |
| 64 Intra-  | 787            |
| 64 Total   | 715            |
| 64 Worst-case ratio | 837 |
| 64          | 530            |
| 64          | 779            |
| 64          | 739            |
| 64          | 1129           |

Experiments are carried out on 10 randomly generated maps with size of 16 × 16, 32 × 32, and 64 × 64. The obstacle density is set to be 20%. Tables 1-2 record the ratio of the pre-calculation using the QH-A* algorithm to the worst-case scenario on each map (the smaller the ratio, the smaller the pre-calculation amount).

The data results show that the amount of pre-calculation of QH-A* is significantly reduced compared to existing methods. According to the trend information of the box chart, the reduction effect of the amount of pre-calculation is more pronounced when the map size is large. QH-A* Algorithms perform better when processing large maps.

B. SEARCH SPACE

In order to evaluate the performance of the algorithm, set the density of obstacles to 20% and 30% on a 64 × 64 and a 128 × 128 map, respectively; we randomly generate 10 sets of source-target pairs, and further compare the A* algorithm and the QH-A* algorithm. Taking into account that the running time of the algorithm is related to the data structure used and the details of the programming implementation, the time consumed by the algorithm is not directly recorded here, but the number of nodes searched by the algorithm to compare the A* algorithm and the QH-A* algorithm. The smaller the number of searched nodes, the higher the efficiency of the algorithm. The experimental results are shown in Table 3.

More intuitively, we perform both A* and QH-A* on map of 64 × 64 and 128 × 128, as shown in Figure 10 and Figure 11. final path is marked in blue and actual search area is marked in green.
| TABLE 3. Algorithm efficiency on maps of different size with different obstacle density. |
|-----------------------------------------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|
| A*                                           | 105           | 53            | 30            | 1326          | 1050          | 119           | 85            | 303           | 56            | 1358          |
| QH-A*                                        | 21            | 8             | 7             | 199           | 111           | 21            | 17            | 40            | 11            | 246           |
| ratio                                        | 20%           | 15.1%         | 23.3%         | 15%           | 10.6%         | 17.6%         | 20%           | 13.2%         | 19.6%         | 17.9%         |

| 64x64 obstacle density:30%                   |               |               |               |               |               |               |               |               |               |               |
| A*                                           | 516           | 997           | 1085          | 389           | 1518          | 470           | 714           | 882           | 635           | 1969          |
| QH-A*                                        | 118           | 129           | 279           | 59            | 154           | 105           | 134           | 161           | 102           | 228           |
| ratio                                        | 22.9%         | 12.9%         | 25.7%         | 15.1%         | 10.1%         | 22.1%         | 18.7%         | 18.2%         | 16.1%         | 11.6%         |

| 128x128 obstacle density:20%                 |               |               |               |               |               |               |               |               |               |               |
| A*                                           | 261           | 2497          | 1906          | 1814          | 2148          | 2343          | 1406          | 113           | 1290          | 203           |
| QH-A*                                        | 38            | 269           | 374           | 353           | 377           | 574           | 296           | 31            | 367           | 52            |
| ratio                                        | 14.5%         | 10.8%         | 19.6%         | 19.5%         | 17.5%         | 24.4%         | 24.1%         | 27.4%         | 28.4%         | 25.6%         |

From these experiments, it is clear that the QH-A algorithm has greatly reduced the search area. Compared with the unmodified A algorithm, the search area has been reduced by more than 70%.

VI. CONCLUSION

In this paper, a new algorithm (termed QH-A) has been proposed for path planning in a complex environment. Our main contribution lies in (a) a kind of hierarchical map which reduces amount of preprocessing. (b) a bi-directional A algorithm specific to hierarchical map. The original map is decomposed through an modified quad-tree decomposition method, which takes into account the obstacle distribution information. Calculating the shortest paths between nodes on a map is the most time-consuming part of any preprocessing methods. Thus the proposed QH-A algorithm introduces hierarchical map through defining the concept of level-edge and intra-edge. This helps to reduce the amount of pre-calculation. On path searches, QH-A adopts the bi-directional A algorithm, which fits the “level up and down” characteristics of the newly built hierarchical map, and further improves the search efficiency.

Since the performance of an algorithm is partially affected by its application scenarios, we want to mention that QH-A algorithm is proposed for grid maps, and cannot be directly applied to other types of maps. The preprocessing step of QH-A takes time and memory to calculate and store related information.

We also want to mention that quad-tree based decomposition may not be the most suitable method as preprocessing, there has been literatures about how to efficiently perform clustering of graphs, such as methods applicable to network, for example,

A variational Bayesian framework for complex network [26], higher-order structures for clustering for biological networks [27], [28], a fuzzy clustering method based on content relevance and linked structures [29]. We are also interested in graph clustering methods which achieves in other applications.

REFERENCES

[1] E. D. Dijkstra, “A note on two problems in connexion with graphs,” Numerische Mathematik, vol. 1, no. 1, pp. 269–271, 1959.
[2] 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., vol. 4, no. 37, pp. 28–29, Jul. 1972.
[3] D. Ferguson and A. Stentz, “Field D*: An interpolation-based path planner and replanner,” in Robotics Research, Berlin, Germany: Springer 2004, pp. 239–253.
[4] S. Koenig and M. Likhachev, “Incremental A∗,” in Proc. Neural Inf. Process. Syst., 2001, vol. 57, no. 1, pp. 1539–1546.
[5] L. Sint and D. de Champeaux, “An improved bidirectional heuristic search algorithm,” J. ACM, vol. 24, no. 2, pp. 177–191, Apr. 1977.
[6] D. Harabor and A. Grastien, “Online graph pruning for pathfinding on grid maps,” in Proc. AAAI Conf. Artif. Intell., 2011, pp. 1114–1119.
[7] N. Pochter, A. Zohar, and J. S. Rosenschein, “Search space reduction using swamp hierarchies,” in Proc. 3rd Annu. Symp. Combinat. Search (SOCS), Atlanta, GA, USA, Jul. 2010, pp. 156–160.
[8] A. Felner and N. R. Sturtevant, “Abstraction-based heuristics with true distance computations,” DBLP, 2009.
[9] N. R. Sturtevant, A. Felner, M. Barrer, and J. Schaeffer, “Memory-based heuristics for explicit state spaces,” in Proc. IJCAI 21st Int. Joint Conf. Intell. Intell. Pasadena, CA, USA: Morgan Kaufmann, Jul. 2009, pp. 609–614.
[10] N. R. Sturtevant, “Memory-efficient abstractions for pathfinding,” in Proc. 3rd Intell. Interact. Digit. Entertainment Conf., Stanford, CA, USA, Jun. 2007, pp. 1–6.
[11] N. Jing, Y.-W. Huang, and E. A. Rundensteiner, “Hierarchical encoded path views for path query processing: An optimal model and its performance evaluation,” IEEE Trans. Knowl. Data Eng., vol. 10, no. 3, pp. 409–432, May/Jun. 1998.
[12] “Valentin Avram road traffic optimization by using an intelligent vehicle-highway system (IVHS),” in Proc. 14th Int. Conf. Control Syst. Comput. Sci. (CSCS), 2003.
[13] S. Jung and S. Pramanik, “An efficient path computation model for hierarchically structured topographical road maps,” IEEE Trans. Knowl. Data Eng., vol. 14, no. 5, pp. 1029–1046, Sep. 2002.
[14] F. Schulz, D. Wagner, and C. Zaroliagis, “Using multi-level graphs for timetable information in railway systems,” in Proc. Alenex Algorithm Eng. Exp. Int. Workshop, vol. 2409, 2002, pp. 43–59.
[15] M. Holzer, F. Schulz, D. Wagner, and T. Willhalm, “Combining speed-up techniques for shortest-path computations,” ACM J. Exp. Algorithmics, vol. 10, pp. 269–284, Dec. 2005.
ZHOU YIJUN was born in Jiangsu, China, in 1975. He is currently a Senior Engineer and a master’s Supervisor with the School of Mechanical Engineering, Southeast University. He has long been engaged in research on advanced manufacturing equipment, robot design and control, computer vision theory and application, precision online inspection and intelligent instrumentation, and automatic optical inspection (AOI). He has published 13 articles in domestic and foreign journals as the first author, authorized three national invention patents as the first inventor, and participated in the compilation of four monographs.

XI JIADONG was born in Jiangsu, China, in 1997. He received the bachelor’s degree from Southeast University, in 2019, where he is currently pursuing the master’s degree. His main research interests include robot design method and control algorithm, and simultaneous localization and mapping.

LUO CHEN was born in China, in 1980. She received the bachelor’s and master’s degrees from Southeast University, and the Ph.D. degree from Shanghai Jiao Tong University, China. She is currently with the School of Mechanical Engineering, Southeast University. Her main research interest includes computer vision, especially 3-D measurement based on machine vision and tolerance modeling.  

---

[16] M. Holzer, F. Schulz, and T. Willhalm, “Combining speed-up techniques for shortest-path computations,” in Proc. Int. Workshop Exp. Efficient Algorithms. Berlin, Germany: Springer, 2004, pp. 269–284.

[17] M. Holzer, F. Schulz, and D. Wagner, “Engineering multilevel overlay graphs for shortest-path queries,” ACM J. Exp. Algorithmics, vol. 13, p. 5, Feb. 2009.

[18] A. Yahja, A. Stentz, S. Singh, and B. L. Brutmann, “Framed-quadtree path planning for mobile robots operating in sparse environments,” in Proc. IEEE Int. Conf. Robot. Autom., May 1998, pp. 650–655.

[19] L. Weyten and W. De Pauw, “Quad list quad trees: A geometrical data structure with improved performance for large region queries,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 8, no. 3, pp. 229–233, Mar. 1989.

[20] O. Souissi, R. Benatitallah, D. Duvivier, A. Artiba, N. Belanger, and P. Feyzeau, “Path planning: A 2013 survey,” in Proc. Int. Conf. Ind. Eng. Syst. Manage. (IESM), Oct. 2013, pp. 1–8.

[21] R. C. Holte, T. Mkadmi, R. M. Zimmer, and A. J. MacDonald, “Speeding up problem solving by abstraction: A graph oriented approach,” Artif. Intell., vol. 85, nos. 1–2, pp. 321–361, Aug. 1996.

[22] J. Jung, I. Kwon, D. Goo, and D. Lee, “Safe pathfinding using abstract hierarchical graph and influence map,” in Proc. IEEE 27th Int. Conf. Tools with Artif. Intell. (ICTAI), Nov. 2015, pp. 860–865.

[23] A. Botea, M. Müller, and J. Schaeffer, “Near optimal hierarchical pathfinding,” J. Game Develop., vol. 1, no. 1, pp. 7–28, 2004.

[24] E. Galceran and M. Carreras, “A survey on coverage path planning for robotics,” Robot. Auto. Syst., vol. 61, no. 12, pp. 1258–1276, Dec. 2013.

[25] Y. Lu, X. Huo, and P. Tsotras, “A beamlet-based graph structure for path planning using multiscale information,” IEEE Trans. Autom. Control, vol. 57, no. 5, pp. 1166–1178, May 2012.

[26] L. Hu, K. C. C. Chan, X. Yuan, and S. Xiong, “A variational Bayesian framework for cluster analysis in a complex network,” IEEE Trans. Knowl. Data Eng., vol. 32, no. 11, pp. 2115–2128, Nov. 2020.

[27] L. Hu, J. Zhang, X. Pan, H. Yan, and Z.-H. You, “HiSCF: Leveraging higher-order structures for clustering analysis in biological networks,” Bioinformatics, vol. 37, no. 4, pp. 542–550, May 2021.

[28] L. Hu, X. Pan, H. Yan, P. Hu, and T. He, “Exploiting higher-order patterns for community detection in attributed graphs,” Integr. Comput.-Aided Eng., vol. 28, no. 2, pp. 207–218, Mar. 2021.

[29] L. Hu and K. C. C. Chan, “Fuzzy clustering in a complex network based on content relevance and link structures,” IEEE Trans. Fuzzy Syst., vol. 24, no. 2, pp. 456–470, Apr. 2016.