International Journal of Geo-Information

MDPI

Article

A Pathfinding Algorithm for Large-Scale Complex Terrain Environments in the Field

Luchao Kui and Xianwen Yu *

School of Transportation, Southeast University, Nanjing 211109, China; 230229452@seu.edu.cn
* Correspondence: yuxianwen@seu.edu.cn

Abstract: Pathfinding for autonomous vehicles in large-scale complex terrain environments is difficult when aiming to balance efficiency and quality. To solve the problem, this paper proposes Hierarchical Path-Finding A* based on Multi-Scale Rectangle, called RHA*, which achieves efficient pathfinding and high path quality for large-scale unequal-weighted maps. Firstly, the original map grid cells were aggregated into fixed-size clusters. Then, an abstract map was constructed by aggregating equal-weighted clusters into rectangular regions of different sizes and calculating the nodes and edges of the regions in advance. Finally, real-time pathfinding was performed based on the abstract map. The experiment showed that the computation time of real-time pathfinding was reduced by 96.64% compared to A* and 20.38% compared to HPA*. The total cost of the generated path deviated no more than 0.05% compared to A*. The deviation value is reduced by 99.2% compared to HPA*. The generated path can be used for autonomous vehicle traveling in off-road environments.

Keywords: pathfinding; RHA* algorithm; multi-scale rectangular region; large-scale map

Citation: Kui, L.; Yu, X. A Pathfinding Algorithm for Large-Scale Complex Terrain Environments in the Field. *ISPRS Int. J. Geo-Inf.* **2024**, *13*, 251. https://doi.org/10.3390/ijgi13070251

Academic Editors: Hartwig H. Hochmair and Wolfgang Kainz

Received: 14 May 2024
Revised: 7 July 2024
Accepted: 10 July 2024
Published: 12 July 2024

1. Introduction

1.1. Background and Significance

In recent years, with the rapid development of mobile robots, artificial intelligence, and other technologies, autonomous vehicles have been gradually applied in military and civil fields such as off-road material transportation, fire support, geological research and investigation, etc. Before carrying out these tasks, it is necessary to provide an optimal path for vehicles and guide them to travel autonomously. The off-road environment presents numerous challenges. The mission area is large, with diverse geological conditions and complex terrain. However, there is a lack of road network data with complete topology. Therefore, it is necessary to develop an algorithm that can quickly generate an optimal path with high access reliability and a short travel time. This has great value for the autonomous vehicle's rapid access in the off-road environment without a road network.

1.2. Literature Review

The ability of an autonomous vehicle in the off-road environment is affected by its own off-road performance and the geographic environment, so it is necessary to comprehensively consider the influence of these two factors when modeling the accessibility of the off-road environment. Wang et al. developed a three-dimensional-like map to represent the off-road environment. This environment model contains not only contour information of obstacles, but also the height information [1]. The vehicle's off-road performance is taken into account when identifying obstacles. Silver et al. studied the cost assessment of unstructured models for complex off-road environments, and provided an environmental cost function for the pathfinding of autonomous vehicles [2]. Bagnell et al. proposed to build an accessibility map based on data from LADAR and satellites, and to generate the travel cost of two-dimensional grids using three-dimensional voxel-related features [3]. Rankin et al. constructed a 3D stereo model of the small-scale off-road environment using

on-board sensors, which can be used for small area pathfinding [4]. Huertas et al. proposed a stereo-based algorithm for tree traversability analysis that considers vehicle speed, mainly for accessibility assessment in wooded environments [5]. Carsten et al. used voxels to construct a geographic graphics environment and provided a three-dimensional capacity map for local area pathfinding [6–8]. Regardless of what method is used, it is necessary to comprehensively consider the impact of various environmental factors on vehicle driving, so as to construct a field accessibility map. This forms the basis for pathfinding in off-road environments.

Pathfinding algorithms can mainly be divided into two categories: sampling algorithms and graph search algorithms.

Sampling algorithms approximate the state space by randomly sampling in the space and then finding paths based on these sampled points. Kavraki et al. first proposed the Probabilistic Road Maps (PRM) algorithm [9], which demonstrated a method for finding high-dimensional spatial paths by randomly sampling and forming a roadmap. LaValle proposed the Rapidly-exploring Random Trees (RRT) algorithm [10], which is a method that constructs a search tree based on random sampling to explore the configuration space and find paths. Karaman et al. proposed the RRT* [11] algorithm based on RRT, which introduces a specific cost function to minimize the connection cost of a new node to its parent node, ensuring a near-optimal path, but it is computationally intensive and inefficient in generating paths. Jordan et al. proposed the Bidirectional Rapidly-Exploring Random Trees (B-RRT*) algorithm [12], which adopts a bidirectional expansion strategy from the start and end points to improve convergence speed. Esposito et al. designed an optimized probabilistic roadmap processing algorithm [13] to simplify the computation required to deal with the number of convex units and nodes in free space, which further improves efficiency. Sampling-based pathfinding algorithms are simple in principle, easy to implement, and do not need to accurately model the obstacles in the environment to achieve pathfinding, but in the off-road large-scale complex terrain environment, the number of sampling points is large, the computational efficiency is low, and it is easy to lead to a planning path which has a large deviation from the actual optimal path.

Graph search algorithms are based on graphical methods to carry out pathfinding, and the classic algorithms include Dijkstra [14], A* [15], Jump Point Search (JPS) [16], JPS+ [17], Hierarchical A* [18] and Hierarchical Path-Finding A* (HPA*) [19].

The Dijkstra algorithm adopts a breadth-first search strategy, searching for the shortest paths between two points in the weighted graph. The A* algorithm is a heuristic search algorithm, and the optimal pathfinding time can be accelerated by reasonably choosing the heuristic function. The JPS algorithm reduces unnecessary node search by finding jump points and substantially improves the pathfinding speed. The JPS+ algorithm proposed by Steve Rabin further improved the pathfinding efficiency by preprocessing and calculating the key jump points in advance. The Dijkstra and A* algorithms ensure that the optimal path is found, but the search efficiency decreases dramatically as the map range expands. JPS is advantageous for its runtime efficiency and reduced memory usage compared to A*, but it is complex to implement and less suitable for non-uniform cost grids. JPS+ builds on JPS's strengths by further reducing runtime complexity through preprocessing, but it incurs higher preprocessing costs and memory usage and is even more complex to implement. However, both methods are not suitable for off-road environments with complex terrain.

In order to improve the efficiency of pathfinding for large-scale, unequal-weighted maps, Holte et al. proposed the Hierarchical A* method, which abstracts the map into clusters of the same size. The midpoints in these clusters are directly connected to form an abstract map. A quadtree-based decomposition [20] of the original map into accessible and obstacle grids of varying sizes was proposed by Samet. This method significantly reduces the number of abstract map nodes and improves the pathfinding efficiency, but the generated paths are of low quality. In order to improve the accuracy of the generated paths, Yahja et al. adopted a framed-quadtree instead of the standard quadtree [21]. This method expands the grid boundaries with cells of the highest resolution, allowing the boundaries

to be traversed in any angular direction, which significantly improves the path quality. The Hierarchical A* and quadtree algorithms significantly improve the efficiency of large-scale map search but result in paths of poorer quality. Framed-quadtree improves the quality of paths by expanding the boundary points, but the significant increase in the number of nodes reduces the pathfinding efficiency.

The HPA* algorithm was proposed by Botea et al. [19]. This method aggregates the original map grids to form multiple layers of abstract maps. Each layer of the abstract maps includes multiple fixed-size clusters, which are connected to each other by symmetric nodes. The optimal paths between the boundary nodes of the clusters are precomputed and cached in a data table. Real-time pathfinding is completed based on the abstract map, and the generated paths within the clusters are refined by checking the table. The HPA* algorithm realizes a significant increase in real-time pathfinding efficiency by aggregating the original maps into several clusters of the same size and precomputing the paths between the nodes at the boundaries of the clusters, solving the problem that algorithms such as A* are unable to adapt to the pathfinding of large unequal-weighted maps. However, in the off-road large-scale complex terrain environment, there are still two problems:

1. The total cost of generating paths is higher than the optimal paths computed by A*, and it cannot be directly applied to vehicles traveling in off-road environments.
2. The terrain in off-road environments has continuity, which means that there are many continuous equal-weighted accessible areas (e.g., grassland). HPA* uses fixed-size clusters to construct the abstract map, which splits these areas into multiple clusters, generating a large number of unnecessary nodes and reducing the pathfinding efficiency in these areas.

1.3. Research Objective

In order to balance the efficiency of real-time pathfinding and the quality of generated paths, this paper proposes a Hierarchical Path-Finding A* based on Multi-Scale Rectangle (RHA*), which improves the HPA* abstract map construction method and aggregates the original grid cells to form rectangular regions of varying sizes. This fully aggregates the equal-weighted areas of the off-road environment, greatly reducing the total cost of generated paths and narrowing the deviation from the optimal path. By optimizing the real-time search algorithm and node selection strategy, the real-time pathfinding efficiency is improved. The RHA* algorithm ensures high performance of the Hierarchical Path-Finding algorithm while significantly improving the quality of the generated paths.

2. Materials and Methods

2.1. RHA Algorithm Architecture Design*

Using the RHA* algorithm to carry out pathfinding in the off-road environment includes two steps: offline preprocessing and real-time pathfinding. The specific process is shown in Figure 1.

1. In the offline preprocessing stage, the original map is abstracted hierarchically using multi-scale rectangular regions. The number of nodes and edges is reduced by optimizing node selection. This creates an abstract map of nodes, edges, and travel costs for real-time pathfinding.
2. In the real-time pathfinding stage, different algorithms are selected to compute the optimal abstract paths based on the given start and end locations, and then the generated paths are refined by looking up the table.

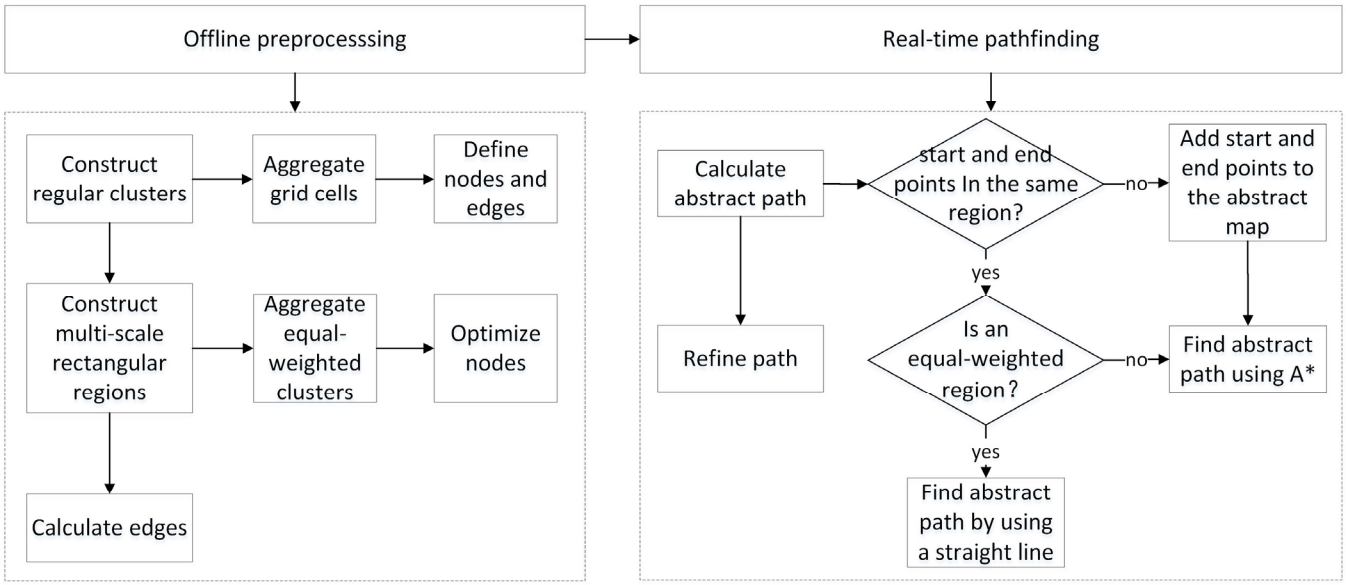

Figure 1. Flowchart of pathfinding in an off-road environment based on the RHA* algorithm.

2.2. Offline Preprocessing

The original map needs offline preprocessing to generate the abstract map, which includes three steps: regular clusters construction, multi-scale rectangular regions construction, and edges calculation. In the following, a 40 × 40 example map is used to detail the entire process of offline preprocessing.

2.2.1. Regular Clusters Construction

Firstly, starting from the upper left corner of the map, $n \times n$ (this paper takes 10×10) neighboring grid cells are aggregated to form a cluster. When the number of original grid cells at the edge of the map is less than $n \times n$, the vacant grid cells are automatically filled using a certain number of inaccessible grids. The example map containing 40×40 grid cells is decomposed into 4×4 clusters. All types of grid cells are involved in the aggregation process. Figure 2 illustrates the aggregation process. Each cluster has a type value that determines whether the adjacent clusters can be further aggregated. The type value is determined by the travel costs of all grid cells within the cluster. When all internal grid cells are inaccessible or there are n ($n \geq 2$) types of travel costs, the cluster type is set to a fixed value and no further aggregation is required. When all internal grid cells have the same travel cost and are accessible, different type values are set according to the travel cost. See Appendix A for details on cluster definition.

Next, the nodes on the cluster boundaries need to be found. A cluster is like a separate room, and the nodes are like doors to the next room. The number and location of the nodes are determined by the channel on the boundary of the neighboring clusters. A channel is defined as the longest accessible segment on the boundary of two neighboring clusters. For each channel, 1–n (this paper takes 3) nodes are determined based on the channel width. If the value of n is too large, the number of nodes will increase significantly and reduce the search efficiency. The gray grids in Figure 3 represent the nodes on the cluster boundary. See Appendix B for details on nodes definition.

Finally, the edges connecting the nodes need to be defined. Two symmetric nodes on the boundary of two adjacent clusters are directly connected to form an inter-edge. The shortest path formed by nodes inside a cluster is called an intra-edge. The gray curves in Figure 3 denote intra-edges, and the blue straight lines denote inter-edges.

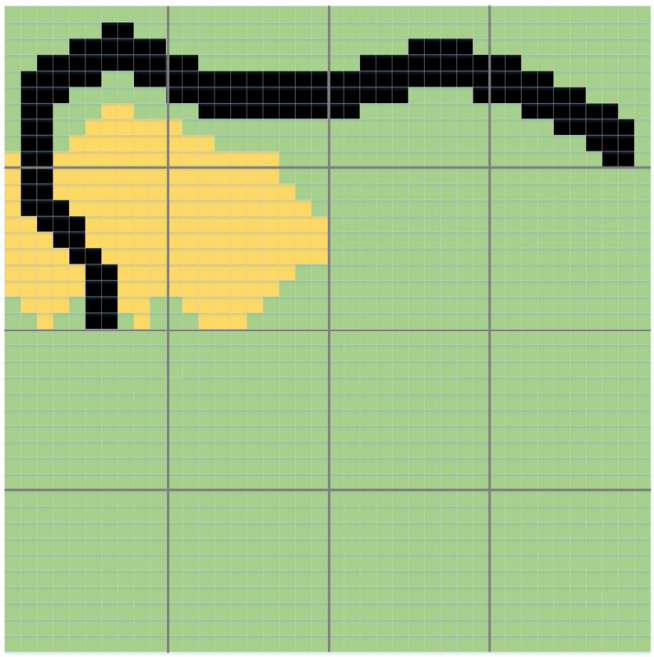

Figure 2. The 40 × 40 example map is decomposed into 4 × 4 clusters by bold black lines, with black grids indicating inaccessible areas, yellow grids indicating high-cost accessible areas, and green grids indicating low-cost accessible areas.

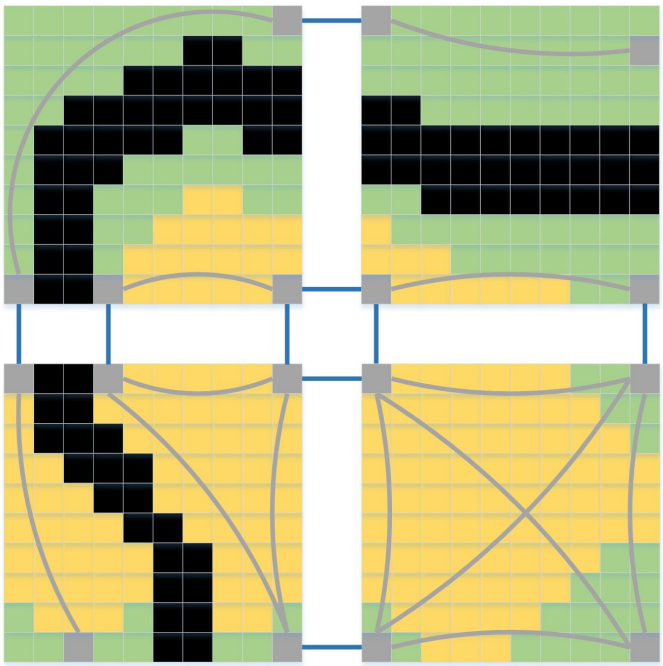

Figure 3. Taking the four clusters in the upper left corner of the example map as an example, the gray grids represent the nodes, the gray curves are the intra-edges, and the blue straight lines are the inter-edges.

2.2.2. Multi-Scale Rectangular Regions Construction

In order to aggregate equal-weighted areas, it is necessary to further aggregate clusters to form multi-scale rectangular regions. For clusters that are inaccessible or contain multiple travel costs, no further aggregation is required. A region containing only these clusters is directly generated, which is called a barrier region or mixed region. Therefore, the

equal-weighted regions must be focused on. An equal-weighted region is a collection of clusters $\{C_1, C_2 \ldots C_n\}$, which needs to satisfy the following conditions:

- All the clusters have the same type:

$$C_1.type = C_2.type = \ldots = C_n.type \quad (1)$$

- One region contains at least one cluster, i.e., $n \geq 1$.
- The interior of a region does not contain any inaccessible area.
- The region is a rectangle.

Starting with the upper left cluster of the map, determine whether each cluster needs to be aggregated. Clusters containing obstacles or multiple travel costs are not aggregated. For clusters that need to be aggregated, this paper adopts a simple and efficient maximum rectangle search method. First, starting from the current cluster to be aggregated (called currentCluster), find the clusters that have the same type value and are continuous. Then, the same search is performed on each subsequent row until the type of the starting cluster differs from currentCluster. The starting cluster for each row search is located directly below the currentCluster. Finally, based on the clusters obtained from the search, find the largest rectangle that can contain the currentCluster. In Figure 4, the gray cluster is the first cluster that needs to be aggregated. Blue clusters are the search start points for each row. Black arrows indicate the direction of the search. The red wireframe is the largest rectangle that can be found, containing a region. See Appendix C for details on region definition and maximum rectangle search method.

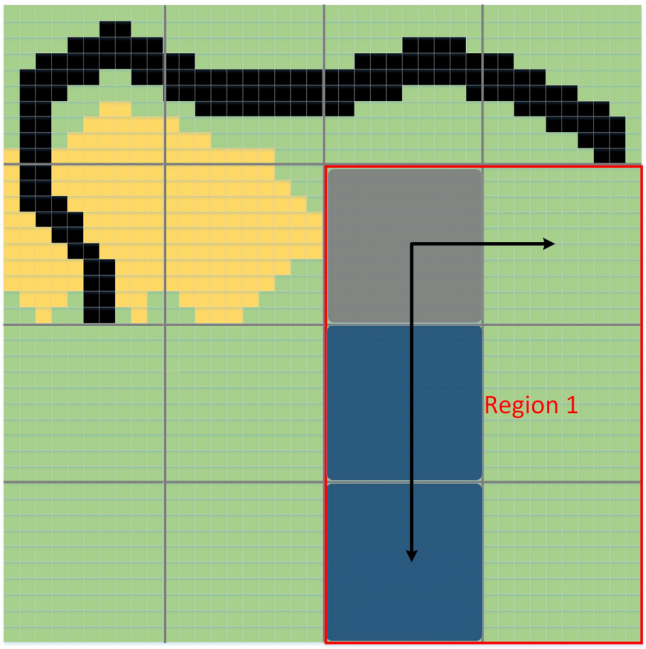

Figure 4. Schematic of cluster aggregation, the gray cluster is the first cluster needs to be aggregated, and the blue ones are the search start points for each row. The red wireframe indicates a region.

An equal-weighted region aggregated from multiple clusters contains many unnecessary nodes that need to be optimized, including the following two situations:

- For nodes that are completely inside the region, just discard them.
- For nodes on the region boundary, they need to be processed according to the number and type of adjacent regions. The basic principle is to discard one of the two adjacent nodes on the region boundary as much as possible while maintaining the complete connectivity of the abstract map. See Appendix D for detailed processing steps.

After optimization, the number of nodes in equal-weighted regions is significantly reduced, but the map connectivity and integrity remain unchanged. In Figure 5, the red nodes need to be discarded, and the gray ones need to be retained.

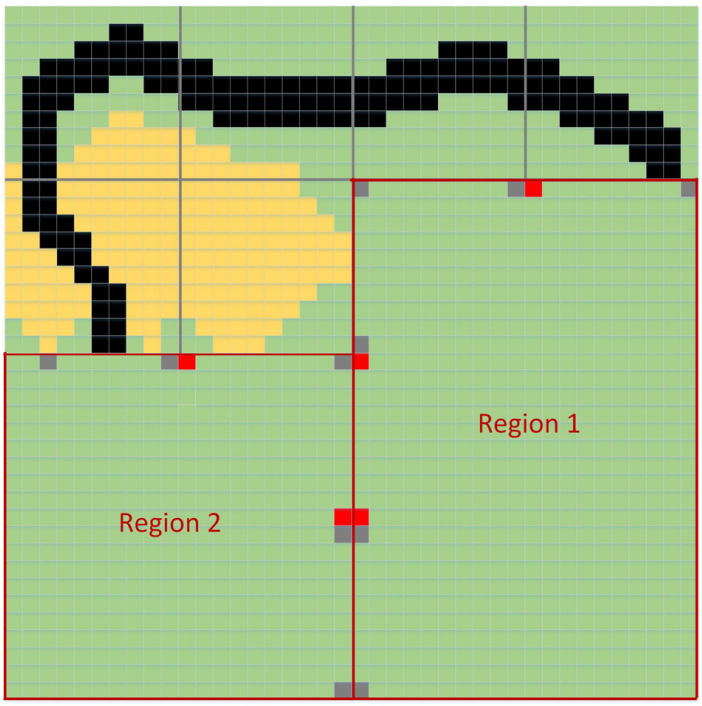

Figure 5. The red nodes need to be discarded after optimization, and the gray ones need to be retained. The red wireframes indicate regions.

2.2.3. Edges Calculation

After completing the construction of regions, edges need to be constructed, including both inter- and intra-edges. The inter-edges are connected by neighboring adjacent nodes, and no additional computation is required. For the unequal-weighted region, which contains only one cluster with a small number of original grid cells, the A* algorithm can be directly used to quickly calculate the optimal path between two nodes. The travel cost of each grid cell that the path passes through is summed to obtain the total cost between the two nodes.

For equal-weighted regions, the number of original grid cells, nodes, and edges contained will be large. According to the Euclidean geometric axioms, the straight-line distance between two points in a rectangular region is the shortest. Therefore, computing edges within the equal-weighted region becomes straightforward. Instead of finding the optimal path based on the original grid cells, the total cost of the edges is computed directly based on the coordinates of the two nodes. Given any two different nodes node1 ($x1$, $y1$), node2 ($x2$, $y2$) on the boundary of the equal-weighted region, as well as the travel cost on the unit grid g, the travel cost between the nodes is obtained by the following equation:

$$cost = \sqrt{(x1 - x2)^2 + (y1 - y2)^2} \times g \tag{2}$$

Figure 6 shows the nodes and edges included in the example map after constructing regions.

After preprocessing, the cost of each edge in the abstract map and its corresponding path on the original map are calculated in advance. The calculation results need to be stored in the local data table and will be used in real-time pathfinding.

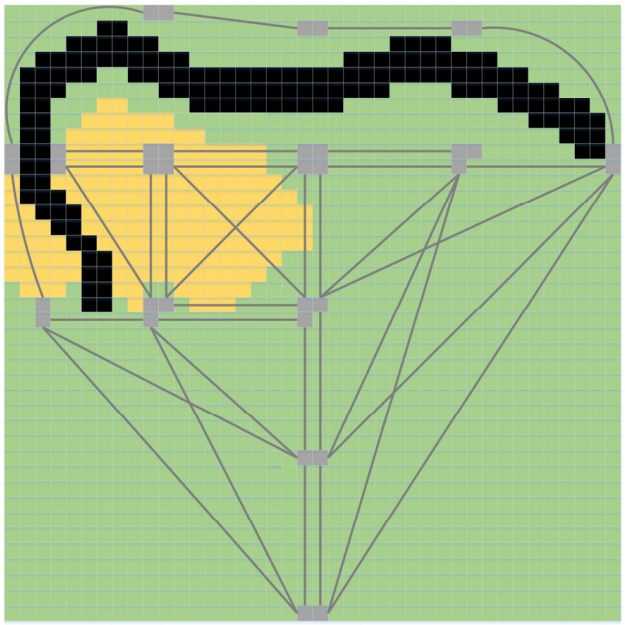

Figure 6. Nodes and edges contained in the example map after decomposing it into regions. The gray grids represent the accessible nodes on the boundaries of the region and the gray lines represent the intra-edges of the region.

2.3. Real-Time Pathfinding

The offline preprocessing obtains an abstract map containing nodes and edges, and the real-time pathfinding is performed based on this abstract map, which mainly consists of two steps: abstract path calculation and path refinement.

2.3.1. Abstract Path Calculation

According to the different positions of the start point S and the end point E added to the abstract map, the calculation process is divided into the following two cases:

1. Points S and E lie in the same equal-weighted region. The equal-weighted region is a rectangular equal-weighted accessible area, so a straight line between S and E is the optimal path. The total cost between S and E is calculated using Equation (2). This situation is more likely to occur when the region's size is larger.
2. Points S and E are not in the same region. S and E need to be added to the abstract map first. For S within the unequally weighted region, use the A* algorithm to compute the edges from S to each node at the boundary of the region and add them to the abstract map. For S within an equal-weighted region, the edges from S to each node are calculated based on the straight-line distance from S to the boundary node and added to the abstract map. The same process is performed for E. Then, based on the complete abstract map, the A* algorithm is used to search for paths starting from S. The search process is performed by using the formula to select the next search node:

$$f(n) = g(n) + h(n) \tag{3}$$

where:

$f(n)$ is the integrated travel cost to the current node to be detected, and the one with the smallest cost is preferred as the next search node; $g(n)$ is the travel cost from the start point along the generated path to the current node to be detected; and $h(n)$ is the valuation function, which is calculated using Euclidean distance.

$$h(n) = \sqrt{(x_n - x_{n-1})^2 + (y_n - y_{n-1})^2} \tag{4}$$

where:

x_n, y_n are the coordinates of the current node, x_{n-1}, y_{n-1} are the coordinates of the previous node.

2.3.2. Path Refinement

Path refinement requires transforming the abstract path into a detailed path based on the original grid cells. The original grid paths corresponding to each unequal-weighted region intra-edge on the abstract path can be retrieved by querying the locally cached data table from Section 2.2.3. Edges within the equal-weighted region contain only two nodes, the start point and the end point, and do not need to be refined. The entire path refinement process can be completed at once, or it can be gradually refined in segments during the vehicle's traveling process as required.

3. Experiment and Results

3.1. Experiment Setup

Benchmark map sets provided by the Mobile AI Lab at the University of Alberta are widely used for testing pathfinding algorithms. This paper selected one of the StarCraft maps named Archipelago as a test, which contains 512 × 512 raw grid cells with three terrains having different travel costs. The experiment uses the standard test set that comes with the map for real-time pathfinding testing. This set includes 2311 test cases, each containing a start point and an end point, and all the test cases are evenly distributed across different areas of the map, which helps to improve the accuracy of the test results. Figure 7 shows the test map.

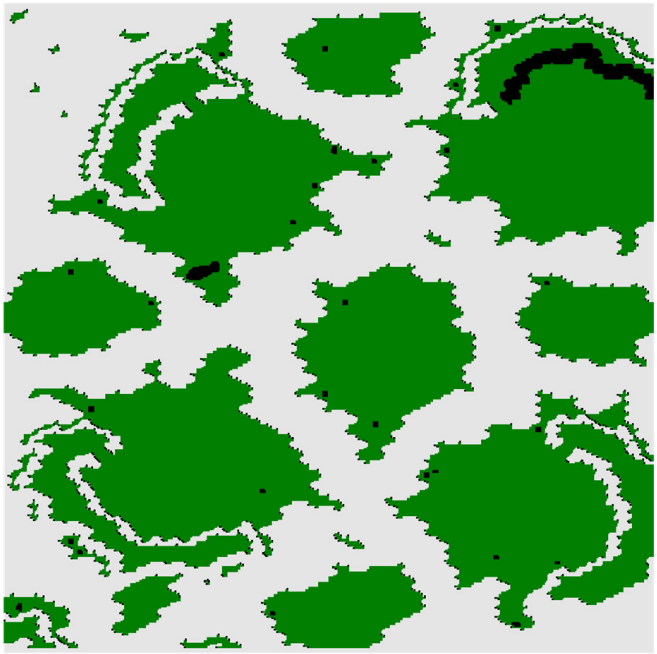

Figure 7. Test map. Green blocks represent vegetation, and black blocks represent obstacles.

The tests were conducted on a 12th Gen Intel(R) Core(TM) i5-1240P 1.70 GHz CPU with 16 GB of RAM. The application was run under a Windows 11 operating system environment.

3.2. Experiment Steps

The experiment consists of two steps: offline preprocessing and real-time pathfinding.

3.2.1. Offline Preprocessing

First, the cluster size in the test program is set to 10 × 10. Then the experimental map data is read for offline preprocessing, which includes regular clusters construction, multi-scale rectangular regions construction, and edges calculation.

After completing the offline preprocessing, an abstract map $G = (V, E, C)$ is formed, where V is the set of all nodes containing a total of 7842 nodes, E is the set of all edges containing a total of 78,994 edges, and C is the set of corresponding travel cost of edges.

3.2.2. Real-Time Pathfinding

The test program reads the standard test set of the experimental map and executes 2311 test cases based on G. Before performing each real-time pathfinding process, the program automatically removes the last addition of the start and end points and their associated edges in G. This ensures the independence and accuracy of each real-time pathfinding test result.

3.3. Result Analysis

Firstly, the real-time pathfinding efficiency of the algorithms is compared. The computation time includes the time for inserting the start and end points into the abstract map, calculating the optimal abstract path, and refining the abstract path. Figure 8 shows a comparison of the average time consumed by the three algorithms: A*, HPA* and RHA*. A* consumes an average of 111.03 ms, HPA* consumes an average of 3.73 ms, and RHA* consumes an average of 2.97 ms. This demonstrates that the improved algorithm proposed in this paper exhibits excellent real-time pathfinding efficiency. The computation time was reduced by 96.64% compared to A* and 20.38% compared to HPA*.

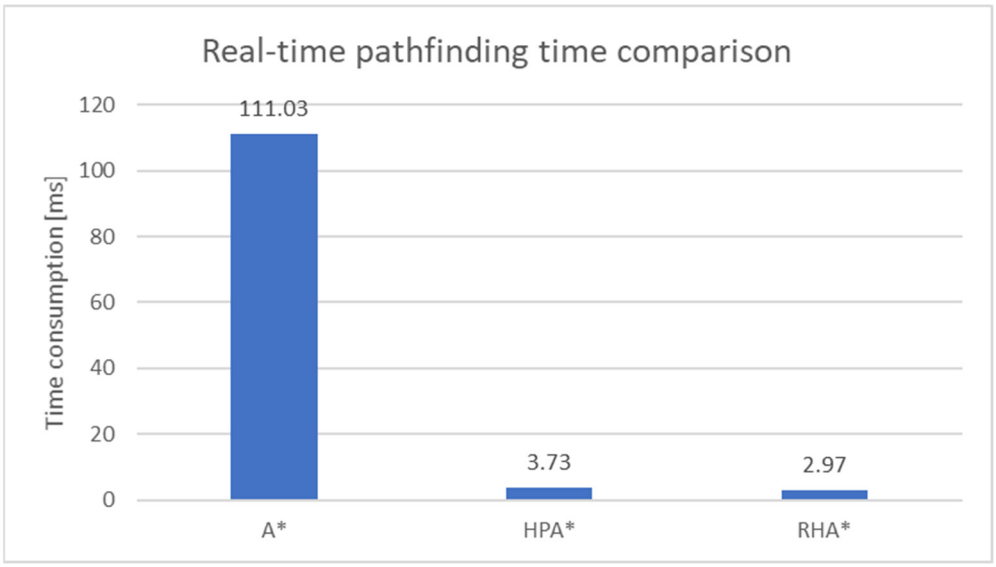

Figure 8. Comparison of real-time pathfinding time consumption.

In terms of the comparison of the quality of the generated paths, it is first defined as:

$$e = \frac{il - ol}{ol} \times 100\% \tag{5}$$

to indicate the deviation value between the total cost of the generated paths and the total cost of the optimal paths computed by A*. Where il represents the total cost of the generated path using HPA* or RHA*, and ol represents the total cost of the optimal path using A*. This paper performed a total of 2311 real-time pathfinding tests, each with a different start point and end point, evenly distributed across different areas of the map. Table 1 presents the average travel cost of paths generated by HPA*, RHA* and A*. It can be seen that the

average deviation between the total cost of paths generated by RHA* and the optimal paths generated by A* is only 0.033%, which is significantly better than 4.12% of HPA*.

Table 1. Comparison of A*, HPA* and RHA* path quality.

	A*	HPA*	RHA*
Total cost of paths	399.5	415.94	399.63
Deviation e	-	4.12%	0.033%

From Figure 8 and Table 1, it can be seen that the RHA* algorithm achieves both real-time pathfinding efficiency and generated path quality. While ensuring high efficiency of real-time pathfinding, it also significantly reduces the deviation between the total cost of the generated path and the optimal path generated by A*.

In the experimental test set, the start and end points are different for each test. However, in actual applications, there may be multiple searches for driving paths from different start points to the same end point. In this case, the process of inserting the end point into the abstract map only needs to be calculated once, and the result can be cached and reused. This can further improve the efficiency of real-time pathfinding.

4. Discussion

4.1. Optimization of Real-Time Pathfinding Efficiency

The HPA* algorithm uses fixed-size clusters to construct an abstract map. In a large-scale off-road environment, this will split continuous equal-weighted areas (such as large grassland, flat bare land, etc.) into multiple clusters, generating a large number of unnecessary nodes. The RHA* algorithm proposed in this paper uses regions of different sizes to construct an abstract map. Fixed-size equal-weighted accessible clusters are further aggregated to form larger rectangular regions. Nodes inside these regions are removed directly, while boundary nodes are optimized as described in Section 2.2.2. This significantly reduces the number of nodes in the abstract map.

Before processing, the experimental map has 262,144 nodes. After HPA* preprocessing, it has 12,088 nodes, and after RHA* preprocessing, it has 7842 nodes. This shows that the number of nodes in the RHA* algorithm is reduced by 35.13% compared to HPA* and by 97.01% compared to the original map. The shortest paths between nodes on the region boundaries are computed offline in advance and stored locally. No further computation for regions is required in the real-time pathfinding stage. The reduction in the number of abstract map nodes and the precomputation of the shortest path within the regions enhance the real-time pathfinding efficiency of RHA*.

4.2. Improvement in the Quality of Generated Paths

In wide off-road environments, the quality of the generated paths is equally important. As shown in Figure 8 and Table 1, the paths generated by RHA* exhibit significantly higher quality compared to those generated by HPA*. Such differences are mainly reflected in the offline map preprocessing stage. HPA* decomposes the original map into multiple clusters of a fixed size. Consequently, each equally-weighted accessible cluster generates at least one node on each boundary. When performing pathfinding in large areas of equal weight, the generated path will consist of nodes at the boundaries of multiple clusters. Such a path is often a fold line.

In contrast, the RHA* algorithm maximally aggregates adjacent equal-weighted accessible clusters during preprocessing to form equal-weighted regions. Nodes within these regions are discarded, and the shortest path between nodes on the region boundary is a straight line. When the start point, end point and access conditions are the consistent, the travel cost of a straight-line path will be significantly lower than that of a fold-line path. Thus, when a generated path traverses both non-equal-weighted and equal-weighted areas, the segment passing through the equal-weighted area consistently improves in quality.

This fundamental difference explains why RHA* produces higher-quality paths compared to HPA*.

5. Conclusions

In this paper, the RHA* algorithm is proposed for the off-road environment pathfinding problem of autonomous vehicles. The algorithm adopts an abstract map construction method based on multi-scale rectangular regions. First, the original map grid cells are aggregated to form multiple fixed-size clusters. Then, unlike HPA*, the equal-weighted clusters are further aggregated to form rectangular regions of varying sizes. The shortest paths between nodes at the boundaries of the regions are computed offline in advance, and the results are stored locally. Finally, real-time pathfinding based on the abstract map is performed to obtain the abstract paths. The abstract paths are refined into final paths based on the original map grid by querying the local data table. The algorithm achieves complete aggregation of equal-weighted areas in the field, enhancing real-time pathfinding efficiency. Simultaneously, it effectively addresses the challenge where the total travel cost of paths generated based on the abstraction map significantly exceeds the optimal paths computed by A*.

The experimental benchmark map used in this paper is provided by the Mobile AI Laboratory at the University of Alberta. The experiment shows that the RHA* algorithm proposed in this paper not only ensures high efficiency in real-time pathfinding, but also significantly reduces the deviation between the total cost of the generated path and the optimal path generated by A*. The generated path can be directly used for unmanned vehicles to travel in the off-road environment, solving the problem of real-time pathfinding in large-scale complex terrain environments in the field. The RHA* algorithm is designed for raster maps. When applied to other types of maps, further improvements to the algorithm may be necessary.

Author Contributions: Formal analysis, Luchao Kui; investigation, Luchao Kui; methodology, Luchao Kui and Xianwen Yu; software, Luchao Kui; supervision, Xianwen Yu; writing—original draft, Luchao Kui; writing—review and editing, Luchao Kui and Xianwen Yu. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by National Key R&D Program of China (grant number 2023YFB3907103).

Data Availability Statement: The experiment data can be found in https://movingai.com/benchmarks/sc1/index.html (accessed on 26 April 2024). The original experiment code can be found in https://github.com/sophie1987/my_pathfinding (accessed on 26 April 2024).

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A

The data structure definition of the cluster mainly contains boundary, width, height, type, nodes, position and regionID. Boundary contains the maximum and minimum x and y values of the four corners of the cluster. Width and height represent the width and height of the cluster which are typically defined as n during cluster generation, though they may be smaller at the edges of each row/column. Type is defined by all grid cells contained in the cluster. Type equals -1 means inaccessible or mixed areas (i.e., contains both inaccessible and accessible areas, or contains multiple grid cells with different costs), requiring no further clustering. Clusters with the same cost have the same type value. Nodes indicate the set of accessible nodes on the cluster boundary. Position denotes the cluster's coordinates in the abstract map, represented by two unsigned integers, x and y. RegionID identifies the region to which the cluster belongs, with a default value of -1 indicating that the cluster has not yet been aggregated.

Appendix B

When two neighboring clusters are not inter-connectable, the number of channels is 0. In the worst case, where an obstacle appears every other grid on the boundary of a cluster, the number of channels is $n/2$ and n is the number of grids on the boundary. For each channel, 1–3 pairs of passage nodes are defined, depending on the channel width. When the channel width is less than $W1$ (in this paper, $W1$ is set to 6), only the center of the channel is used as a node; when the width of the channel is greater than or equal to $W1$ and less than $W2$ (this paper takes $W2$ as 15), the two endpoints on both sides of the channel serve as nodes. When the width of the channel is greater than or equal to $W2$, nodes include the endpoints on both sides and the center of the channel.

Appendix C

The data structure of a region mainly includes id, clusters, type, leftTopCoord, rightBottomCoord, and nodes. Clusters represent the set of all clusters contained in a region. Type is the type of the region. The leftTopCoord and rightBottomCoord represent the position of the upper-left grid and the lower-right grid of the region, respectively, through which the range of the region and the positions of all the internal clusters and grid cells can be derived. Nodes are the set of nodes at the boundary of the region.

The detailed processing steps of the maximum rectangle search method are as follows:

1. Judge whether currentCluster.regionID is equal to -1: Yes means the cluster is not aggregated, and go to the second step of judgment; No means the cluster has been aggregated, and jump out of the loop to traverse the next cluster.
2. Judge whether currentCluster.type is greater than 0: Yes means that the cluster is an equal-weighted accessible area, and go to the third step; No means that the cluster is an inaccessible area or a mixed area, so there is no need to aggregate, so jump out of the loop and traverse the next cluster.
3. From the current position of currentCluster, keep finding the neighboring clusters to the right, and judge whether the types are the same, until we find the clusters with different types. This gives the maximum number of neighboring clusters of the same type in the row and is recorded in the maxWidth array. Use the same method to iterate over the next row until the maximum number of adjacent same-type clusters in a row is 0 to stop iterating and go to step four.
4. Starting from the current row of currentCluster, expand downward. Based on the maxWidth array which records the maximum number of neighboring same-type clusters in each row, and the number of expansion rows, derive the maximum rectangular area that can be formed when expanding to each row. Select the rectangular area with the largest area as the currentRegion corresponding to the current cluster.
5. Set the regionID of other clusters contained in currentRegion to the id of currentRegion, and iteratively traverse the next cluster.

Appendix D

For the two adjacent nodes, node1 and node2, on the boundary of an equal-weighted region A, the following two processing cases are adopted:

- If the nodes node3 and node4, which are symmetric to node1 and node2, are in the same equal-weighted region B, node1 and node3 are deleted directly, or node2 and node4 are deleted.
- If the nodes node3 and node4 symmetric to node1 and node2 are in two regions B and C, respectively, node1 (or node2) is deleted and an edge is added between node2 (or node1) and node3 to maintain the integrity of the abstract map.

References

1. Wang, Y.; Cao, W. A global path planning method for mobile robot based on a three-dimensional-like map. *Robotica* **2014**, *32*, 611–624. [CrossRef]
2. Silver, D.; Bagnell, J.A.; Stentz, A. Learning from demonstration for autonomous navigation in complex unstructured terrain. *Int. J. Robot. Res.* **2010**, *29*, 1565–1592. [CrossRef]
3. Bagnell, J.A.; Bradley, D.; Silver, D.; Sofman, B. Learning for Autonomous Navigation. *IEEE Robot. Autom. Mag.* **2010**, *17*, 74–84. [CrossRef]
4. Rankin, A.L.; Huertas, A.; Matthies, L.H. Stereo-vision-based terrain mapping for off-road autonomous navigation. In *Unmanned Systems Technology XI*; SPIE: Orlando, FL, USA, 2009; Volume 7332, pp. 253–269.
5. Huertas, A.; Matthies, L.; Rankin, A. Stereo-based tree traversability analysis for autonomous off-road navigation. In Proceedings of the Seventh IEEE Workshops on Application of Computer Vision, Washington, DC, USA, 5–7 January 2005; Volume 1, pp. 210–217.
6. Carsten, J.; Ferguson, D.; Stentz, A. 3D Field D*: Improved path planning and replanning through three dimensions. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Beijing, China, 9–15 October 2006; pp. 3381–3386.
7. Lacaze, A.; Murphy, K.; DelGiono, M. Autonomous mobility for the Demo III experimental unmanned vehicles. In Proceedings of the AUVSI Conference, Orlando, FL, USA, 8–12 July 2002.
8. Kelly, A.; Stentz, A.; Amidi, O.; Bode, M.; Bradley, D.; Diaz-Calderon, A.; Happold, M.; Herman, H.; Mandelbaum, R.; Pilarski, T.; et al. Toward reliable off road autonomous vehicles operating in challenging environments. *Int. J. Robot. Res.* **2006**, *25*, 449–483. [CrossRef]
9. Kavraki, L.E.; Svestka, P.; Latombe, J.C.; Overmars, M.H. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Autom.* **2006**, *12*, 566–580. [CrossRef]
10. LaValle, S.M. Rapidly-Exploring Random Tree: A New Tool for Path Planning. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Leuven, Belgium, 16–20 May 1998; Volume 3.
11. Karaman, S.; Frazzoli, E. Incremental sampling-based algorithms for optimal motion planning. *arXiv* **2011**, 1105–1186. [CrossRef]
12. Jordan, M.; Perez, A. Optimal Bidirectional Rapidly-Exploring Random Trees. Computer Science and Artificial Intelligence Laboratory Technical Report 2013. Available online: https://dspace.mit.edu/handle/1721.1/79884 (accessed on 26 April 2024).
13. Esposito, J.M.; Wright, J.N. Matrix completion as a post-processing technique for probabilistic roadmaps. *Int. J. Robot. Res.* **2019**, *38*, 388–400. [CrossRef]
14. Dijkstra, E.W. A note on two problems in connexion with graphs. *Numer. Math.* **1959**, *1*, 269–271. [CrossRef]
15. Hart, P.E.; Nilsson, N.J.; Raphael, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.* **1968**, *4*, 2829. [CrossRef]
16. Harabor, D.; Grastien, A. Online graph pruning for pathfinding on grid maps. In Proceedings of the AAAI Conference on Artificial Intelligence, San Francisco, CA, USA, 7–11 August 2011; Volume 25, pp. 1114–1119.
17. Rabin, S. A* Speed Optimizations. In *Game Programming Gems*; Deloura, M., Ed.; Charles River Media: Needham, MA, USA, 2000; pp. 272–287.
18. Holte, R.C.; Perez, M.B.; Zimmer, R.M.; MacDonald, A.J. Hierarchical A*: Searching abstraction hierarchies efficiently. In Proceedings of the AAAI Conference on Artificial Intelligence, Portland, OR, USA, 4–8 August 1996. Available online: https://aaai.org/papers/079-aaai96-079-hierarchical-a-searching-abstraction-hierarchies-efficiently (accessed on 26 April 2024).
19. Botea, A.; Müller, M.; Schaeffer, J. Near-Optimal Hierarchical Pathfinding. *J. Game Dev.* **2004**, *1*, 1–30.
20. Samet, H. An Overview of Quadtrees, Octrees, and Related Hierarchical Data Structures. *Theor. Found. Comput. Graph. CAD* **1988**, *F40*, 51–68.
21. Yahja, A.; Stentz, A.; Singh, S.; Brumitt, B.L. Framed-quadtree path planning for mobile robots operating in sparse environments. In Proceedings of the 1998 IEEE International Conference on Robotics and Automation, Leuven, Belgium, 16–20 May 1998; Volume 1, pp. 650–655.