Intelligent Service Robotics (2022) 15:289–306
https://doi.org/10.1007/s11370-022-00416-8

ORIGINAL RESEARCH PAPER

Auto-splitting D* lite path planning for large disaster area

Shin-nyeong Heo[1] · Jiaheng Chen[1] · Yu-chi Liao[1] · Hee-hyol Lee[1]

Received: 6 July 2021 / Accepted: 14 February 2022 / Published online: 16 March 2022

Abstract
This research introduces a new path planning method for rescue robots in a dynamic and partially known area when the robots are performing tasks in a large area. The path planning of the rescue robots that move in the dynamic area is introduced to solve the issue of unnecessary areas, which are the disadvantages of the existing D*-based algorithms. This paper proposes a method to eliminate unnecessary expanded nodes of the dynamic and partially known environment by segmenting a map with an auto-clustering algorithm, which is able to achieve a faster execution time than conventional algorithms. Furthermore, to show the effectiveness of the proposed algorithms, an expected value of re-planned nodes in the dynamic and partially known area is introduced using a probability-based approach.

Keywords Global path planning · Dynamic environment · Partially known area · Large-scale area · D*-based algorithm · Auto-clustering algorithm

1 Introduction

Due to the recent surge in disaster-level large-scale disasters caused by climate change, rescue robots and UAVs (unmanned aerial vehicle) have been developed to assist first responders. The safety of the people in danger and the first responders should be most considered. Robots and UAVs have been strengthened with many technologies to help the first responders to protect their safety. The UAVs are developed to capture images from wide areas, and send the image and location information to the robots immediately. In general, the rescue robots are designed to perform global path planning tasks and local path planning tasks simultaneously. The essential element of the rescue robots mission is the time needed from a starting point to its destination, which could possibly be most reduced by the global path planning. Therefore, an efficient global path planning method is needed to support rescue robots in a large disaster area.

In this paper, the research target is to design a global path planning algorithm for rescue robots in a large disaster area. The environments are generally dynamic and partially known areas, which is why global path planning is adopted. The global path planning algorithm is capable of giving more information in the dynamic and partially known area than static and known area. Furthermore, this information increases the survival rate of those people in danger. It means that the path planning algorithm requires a shorter path and a faster execution time. The shorter path can decrease the arrival time to the destination. Also, the faster execution time for updating the path improves the performance of the algorithm when the environment is dynamic.

Figure 1 illustrates this research background. The first responders are in charge of controlling the rescue robots and disaster area management. The rescue robots are mobile robots with multiple sensors such as a robot with drones, a robot with robot arms, and a robot specialized for transportation. The first responders are distributed to the spot by trucks equipped with servers, including several robots on board. When the search and rescue process begins, the drone in the robots will scan the disaster area. Later on, the drone's image information is sent back to the server for the first responders to give instructions based on the provided information.

✉ Shin-nyeong Heo
snheo@akane.waseda.jp

Jiaheng Chen
chenjia@akane.waseda.jp

Yu-chi Liao
yukiandyoki@moegi.waseda.jp

Hee-hyol Lee
hlee@waseda.jp

[1] Graduate school of Information, Production and Systems, Waseda University, 2-7 Hibikino, Wakamatsu Ward, Kitakyushu Fukuoka 808-0135, Japan

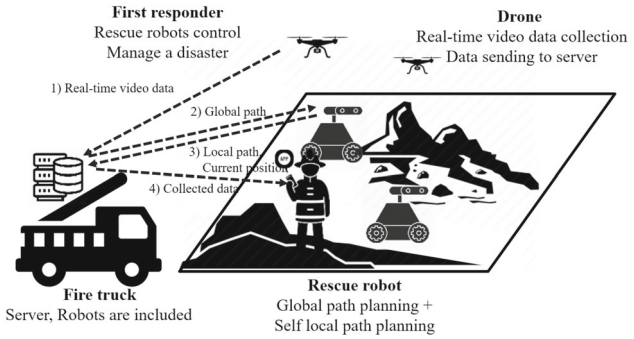


Fig. 1 Background (flow of data represents in the order of 1) to 4))

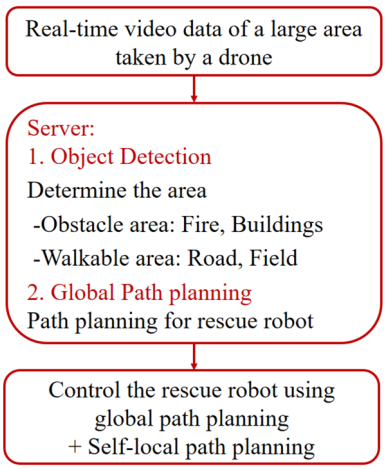


Fig. 2 Flow of research overview (robots are composed in the order of data flow in Fig. 1, and this research focuses on 2. Global path planning. A typical example of 1. Object detection is the use of a you only look once (YOLO) algorithm

Figure 2 shows the flow of the research. Most of the algorithms are processed on the server. The server sends the information of the planned path based on the robot's current location along with the obstacle information provided by the UAVs to the rescue robot. Then, the robot follows the global planned path with an implemented automatic control system. This paper consists of five parts to show the effectiveness of a proposed global path planning algorithm. Section 2 describes the current issues of previous global path planning algorithms in the dynamic and partially known area. Also, auto-clustering methods are summarized to select a better method to apply with the proposed global path planning algorithm, which are solutions to the current issues. Section 3 describes details of the proposed algorithm, and the performance is analyzed through a time complexity method in special cases. Section 4 shows the simulation of special cases, algorithm performances, large city map simulations and large rural map simulations. Section 5 describes the conclusions of this paper.

2 Related work

Previous research on the global path planning algorithms in the dynamic and partially known area for large areas is described below. The common target of the algorithms is to achieve a shorter path and a faster execution time. The dynamic global path planning algorithms are divided into several representative categories based on the characteristics of the algorithm. Grid-based algorithms, typically D* and D* lite algorithms, have an advantage in dynamic environments due to their fast performance. Nevertheless, their performance decreases significantly as a map grows in size [1–4]. In order to maintain fast computational performance, methods to reduce expanded nodes have been proposed in several ways. A typical method is a weighted method [5,6]. However, the weighted methods can sometimes perform worse than the traditional algorithms [7]. In many situations, there is a general trend where a higher weight in the weighted cost method leads to a faster search. Nevertheless, there are also circumstances where a higher weight leads to a slower search. The weighted search is fast if there is a strong correlation between the estimated cost of getting to a goal node from current node and the number of nodes between current node and the nearest goal measured in edge count. In general, the weighted cost reduces the number of expanded nodes, but in situations where the costs are too highly weighted, the number of expanded nodes cannot be reduced due to the reason that the map becomes similar to a free-cost map. Hence, the performance will be the same or even worse than the traditional algorithm. Therefore, it is necessary that the grid-based algorithms require to reduce expanded nodes in other ways. Sampling-based algorithms, typically RRT and RRT* algorithms, are less capable of performing tasks in dynamic environments, but have better performance as the map grows in size [8–11]. The traditional approaches such as the potential fields and their related algorithms are not suitable for the dynamic and partially known environment, and the execution time is not fast enough [12–16]. Discrete optimization algorithms such as particle swarm optimization are used in both global and local path planning methods. Nonetheless, most algorithms can only deal with static obstacles. Their performance for moving obstacles is not sufficient enough to perform dynamic path planning in the large area [17–19]. Now, the current trend is to use sampling-based algorithms along with discrete optimization algorithms, and the machine learning method with the grid-based algorithm. [20–22] However, as the calculation performance of various chipsets has increased rapidly due to the development of the semiconductor industry, the research aimed at fast global path planning in large areas is actively underway. Consequently, the grid-based global path planning algorithms are being considered again to obtain high speed in the dynamic and partially known area. The grid-based global path plan-

ning algorithms work well for the dynamic and partially known areas, yet large dimension area processing is always the weak point. Especially, unnecessary dynamic areas exist in D* based path planning [23–26].

To reduce the unnecessary areas, which means unnecessarily expanded nodes, of the D* based path planning, this research proposes a way to pre-split a map using an Auto-splitting D* lite algorithm denoted as AS-D* lite. The AS-D* lite algorithm uses an automatic clustering algorithm that calculates the optimal map segmentation automatically using the information of the obstacle's location.

On the other hand, how to split an area automatically is just as important as removing unnecessary nodes for path planning. For splitting the map automatically, an automatic clustering method is commonly used. The automatic clustering methods are divided into different types such as centroid-based, connectivity-based, and density-based. The centroid-based automatic clustering is appropriate for determining the number of clusters for unlabeled data [27,28]. The connectivity-based automatic clustering is suitable for hierarchical clustering [29,30]. The density-based automatic clustering applies autonomous machine learning techniques and is widely used for finding clusters of any arbitrary shape, not only spheres [31,32]. However, the centroid-clustering method is considered for splitting the maps through the path planning conditions, which are large disaster areas including dynamic and partially known, to achieve a fully automatic splitting method with a shorter calculation time. Nevertheless, there are some issues in the previous centroid-based automatic clustering methods with the grid-based global path planning algorithms for the dynamic environments. The centroid-based automatic clustering methods usually require a long processing time. This is a common issue that occurs in these algorithms. A k-means clustering and a mean-shift clustering-based algorithms are typical centroid-based algorithms. The k-means clustering-based algorithm takes a long time to determine the k-value. This means that the grid-based path planning algorithm takes a long time to achieve an optimal effect by classifying a map based on the obstacles in the map [33]. The mean-shift clustering-based algorithms also have similar issues such as the repeated calculations. The repeated calculation carried out to find a center point rather than the k-value [34,35].

To shorten the processing time for determining the k-value comparing with the other methods, a gap statistics is applied to find an optimal k-value, which is a splitting number, for the k-means algorithms. Moreover, a Voronoi algorithm is combined with the k-means algorithm to draw a segmented map automatically with the optimal k-value.

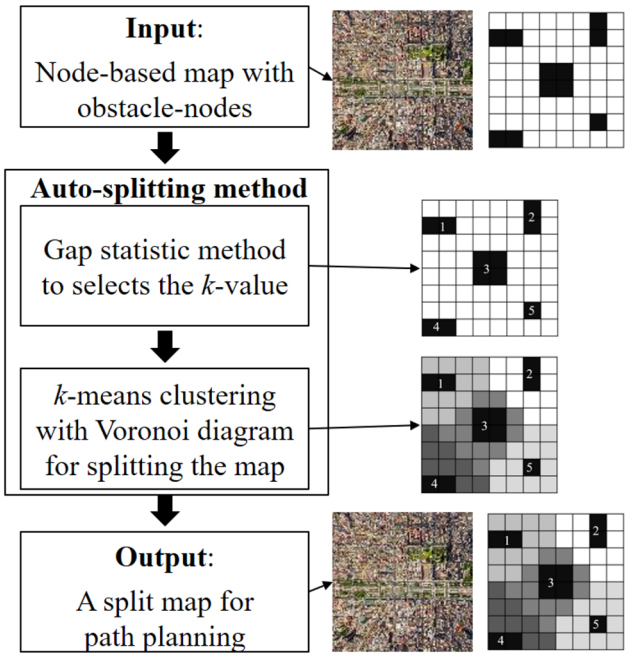


Fig. 3 Overview of auto-splitting method

3 Auto-splitting D* lite

3.1 Auto-splitting method for auto-clustering and drawing

As explained in Sect. 2, it is necessary to pre-divide a map to eliminate unnecessary areas when planning the path with the D* based algorithm on the large map. Accordingly, the automatic clustering method is used as a way to separate the map in this paper. However, the previous centroid-based automatic clustering methods require a long processing time [33,36]. Especially, the k-means clustering-based algorithm takes a long time to determine the k-value optimally. Therefore, the gap statistics is used to obtain the optimal k-value, which is the splitting number, for the k-means algorithms to achieve a shorter processing time comparing with all the other methods. The Voronoi diagram is combined with the k-means algorithm to draw a splitting map automatically with the optimal k-value.

Figure 3 shows the overall flow for the auto-splitting method. The auto-splitting method proceeds in the following steps:

Input: node-based map with obstacle nodes.
Step 1. Select the k-value by the gap statistic.
Step 2. Divide the map by the k-means clustering algorithm with the Voronoi diagram.
Output: a split map for path planning.

The information of the map retrieved by a drone is used as the input, and the grid map is generated according to the location of the obstacles. With the information collected, the optimal k-value can be found by the gap statistics. The details are described in Sect. 3.1.1. Then, the map is split by the lines where the Voronoi circles meet each other. Detailed descriptions are introduced in Sect. 3.1.2.

3.1.1 Gap statistics

The gap statistics method is applied for selecting the k-value before the k-means algorithm splits the map. There are other automatic k-means clustering methods such as combining a Silhouette coefficient [37] with the k-means algorithm. However, most of the previous methods require a long calculation time to determine the k-value [33,36]. Furthermore, determining the k-value with learning algorithms for complete automation tends to make the proceeding time longer [38]. Centroid clustering methods such as mean-shift clustering-based methods also require longer calculation time because of the centroid finding issue, which means repeated calculations.

The selected method based on the gap statistics is discussed by Tibshirani et al. [39] for estimating the number of clusters in a set of data. In this paper, the gap statistics is used to find an optimal k-value for the k-means algorithm with rapid calculation in the preprocessing step. It should be noted that the first step in the Auto-Splitting D* lite (AS-D* lite) algorithm is operated under a static environment. This method shows a faster processing time than the k-means with the Silhouette method, which has been frequently used.

List of symbols in gap statistics

n	Number of node (x-axis)
m	Number of node (y-axis)
N	Total number of obstacle node
(x, y)	Position of each node
G	Set of ground nodes
O	Set of obstacle nodes
R	Set of randomized obstacle nodes
$o_i(x, y)$	Element of obstacle nodes
(x', y')	Randomized position of each nodes
$r_i^d(x', y')$	Element of randomized obstacle nodes
M	Sample size determination
Z	Z score
EBM	Error bound for the mean
W_k	Within-cluster sum of squares using obstacle nodes
W_{kb}^*	Within-cluster sum of squares using randomized nodes
sd_k	Standard deviation
s_k	Simulation error

Let us consider an n[node]$\times m$[node] map having a total of N obstacle nodes. The ground node is represented as $G = \{(x, y)|x = 1, 2, \ldots, n, y = 1, 2, \ldots, m\}$. A set of obstacle nodes O is a subset, which belongs to the set of ground nodes G. The obstacle nodes O is denoted as $O = \{o_i(x, y)|i = 1, 2, \ldots, N\}$. Also, let us consider a set of randomized obstacle nodes R having a total of B patterns, which is represented as $R = \{r_i^d(x', y')|i = 1, 2, \ldots, N, d = 1, 2, \ldots, B\}$. The gap statistics is used to determine an optimal k-value compared to each pattern. The position of random obstacle nodes are represented as (x', y') by randomizing function $rand()$. The random obstacle nodes are limited based on the obstacle nodes as given by

$$\begin{cases} \min\limits_x(o_i(x, y)) < r_i^d(x', y') < \max\limits_x(o_i(x, y)) \\ \min\limits_y(o_i(x, y)) < r_i^d(x', y') < \max\limits_y(o_i(x, y)) \end{cases}. \tag{1}$$

The number of total patterns B is recommended to simplify the calculation process using a function floor() with the sample size determination M as given by

$$M \triangleq \left(\frac{Z\sigma}{\text{EBM}/2}\right)^2 \tag{2}$$

$$\text{floor}(M) = \max\{B \in \text{Integer}|B \leq M\} \tag{3}$$

where Z is a standard z-score, and EBM is the error bound for a population mean. For the calculation example, if a randomized data set is assumed to follow $N(0, 1)$ when the desired confidence interval is 90%, then the z-score is 1.645. When the desired margin of error is $\pm 0.5\%$, then the EBM is 1. In this situation, the confidence interval is $90\% \pm 0.5\%$, then the value of floor(M) is floor(10.82)=10 from Eq. (2). Other distributions, such as a uniform distribution, are possible to apply for the confidence interval. Depends on the distribution, the standard deviation changes the B value.

The gap statistics $Gap(k)$ for k-value is defined as

$$\text{Gap}\,(k) \triangleq \left(\frac{1}{B}\right)\sum_{b=1}^{B}\log\left(W_{kb}^*\right) - \log\left(W_k\right) \tag{4}$$

where W_k is calculated by the obstacle nodes, and W_{kb}^* is calculated by the randomized nodes in the map, respectively. The standard deviation sd_k is calculated by

$$sd_k = \left[\left(\frac{1}{B}\right)\sum_{b=1}^{B}\left\{\log\left(W_{kb}^*\right) - \left(\frac{1}{B}\right)\sum_{b=1}^{B}\log\left(W_{kb}^*\right)\right\}^2\right]^{\frac{1}{2}} \tag{5}$$

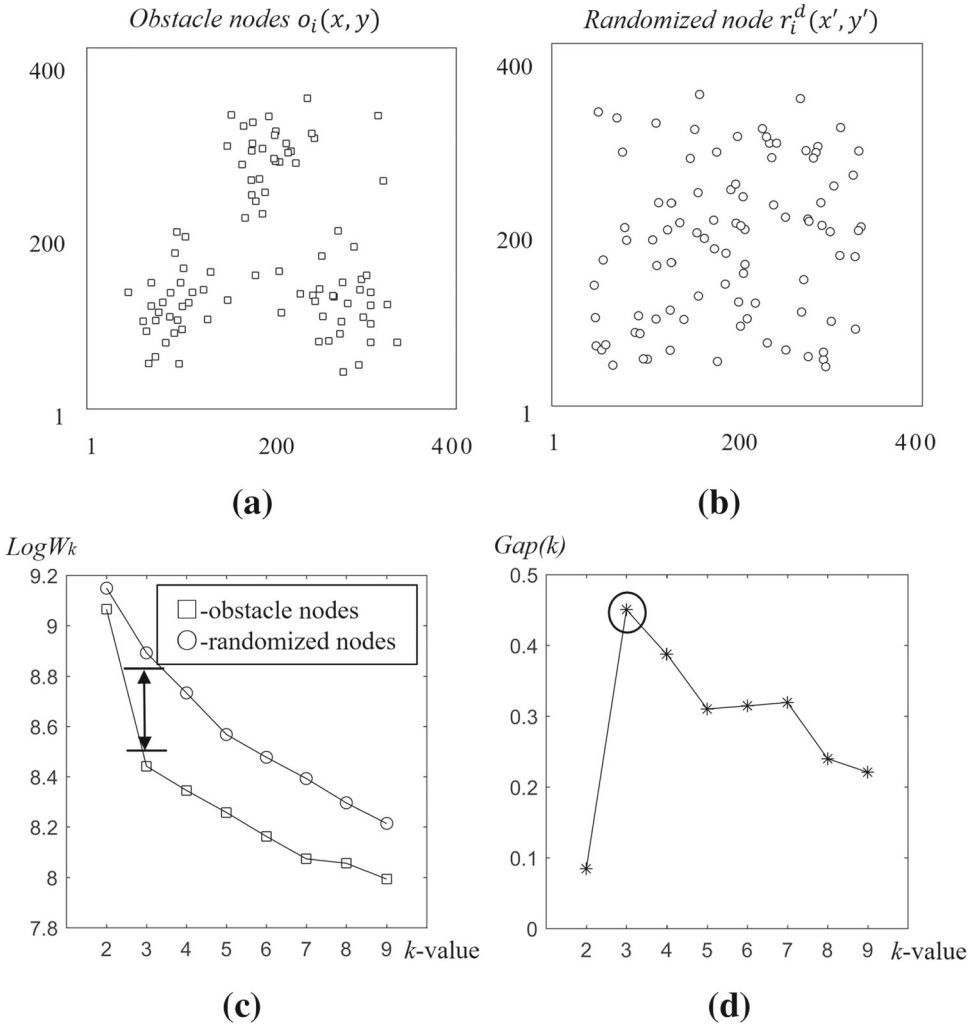


Fig. 4 Procedure of gap statistics. **a** Detecting the position of obstacle nodes, **b** generating the location of the random node based on the position of the obstacle nodes, **c** calculating within-cluster sum of squares, **d** calculating the $Gap(k)$ and using method to select k-value

The simulation error s_k is calculated by the standard deviation sd_k and the value B such as

$$s_k = sd_k\sqrt{(1 + 1/B)}. \tag{6}$$

The optimal k-value is the smallest k-value which satisfies the equation:

$$\text{Gap}\,(k) \geq \text{Gap}\,(k + 1) - s_{k+1}. \tag{7}$$

Figure 4 shows the working principle of the gap statistics. From Fig. 4d, the best k-value is indicated as $k = 3$. The gap statistics is applied for selecting the best k-value before using the k-means clustering algorithm combined with the Voronoi diagram to split a large map.

3.1.2 Combined *k*-means clustering with Voronoi diagram

The k-means clustering algorithm is combined with a Voronoi diagram for splitting the large map, and determines the splitting line automatically. Also, the auto-splitting method needs to initialize the positions of the cluster centers and the initial k-value. The k-means algorithm is not faster than heuristic algorithms. However, the AS-D* lite determines an optimal k-value more conveniently than the heuristic algorithms [36,40]. The AS-D* lite is briefly represented in three steps as follows:

Step 1. Input every cluster center from the k-means clustering.
Step 2. Draw the Voronoi diagram based on the cluster centers.
Step 3. Draw the splitting line after the Voronoi diagram is finished.

List of symbols in *k*-means with Voronoi

CR	Set of cluster regions
G	Set of ground nodes
O^s	Subset elements of obstacle nodes in each cluster center

N	Total number of obstacle nodes
N_s	Total number of elements belong to O^s
d_{js}	Distance between obstacle nodes and cluster centers
$o_j^s(x, y)$	Position of obstacle nodes
$c_s(x, y)$	Position of each cluster center
c_s	Set of cluster center
d_s	Distance between ground node and cluster centers
W_k	Within-cluster sum of squares using obstacle nodes

The k-means clustering is a method for separating data into k sets, and is applied for partitioning obstacles into k-regions on a two-dimensional map. Then, the map is partitioned into k-regions, which is the same as the k-value used in the k-means clustering algorithm. The map is composed of obstacle nodes and ground nodes. The position data of obstacle nodes needs to be collected from the map, and an initial k-value needs to be set before applying the k-means clustering method. Then, the two-dimensional coordinate of each obstacle node in the map is stored. The initial k-value is set as $k_{\text{initial}} \geq 2$.

Also, the initial k-value represents the lower limit of an expected number of clusters. The Voronoi diagram is applied to classify each ground node in the map to one of the cluster regions. The large map is divided into k cluster regions, and the nodes which have the same distance to two different regions are defined as the splitting lines.

The cluster regions are represented as CR, and each cluster region is represented as $CR = \{CR_s | s = 1, 2, \ldots, k\}$, where $G = \bigcup_{s=1}^{k} CR_s$. Each cluster center is denoted as $c_s(x, y)$ in the cluster region CR_s. The obstacle nodes in each cluster region are represented as $O^s = \{o_j^s | j = 1, 2, \ldots, N_s\}$, for example $\{o_1^s, o_2^s, \ldots, o_{N_s}^s\}$ in CR_s. Therefore, the O^s has the total of N_s obstacle nodes. The total number of obstacle nodes is equal to the sum of the number of nodes in each cluster region $N = \sum_{s=1}^{k} N_s$. To draw the Voronoi diagram, a distance d_{js} between obstacle nodes $o_j^s(x, y)$ and cluster centers $c_s(x, y)$ in the cluster region CR_s are defined as

$$d_{js} = |o_j^s(x, y) - c_s(x, y)|^2 \tag{8}$$

A value of the within-cluster sum of squares W_k for each cluster region CR_s is defined as

$$W_k = \sum_{s=1}^{k} \frac{1}{2N_s} \sum_{o_j^s, c_s \in CR_s} d_{js}. \tag{9}$$

The initial position of each cluster center is set randomly. A final position of each cluster center is set by

$$\underset{c_s}{\text{argmin}}\, W_k. \tag{10}$$

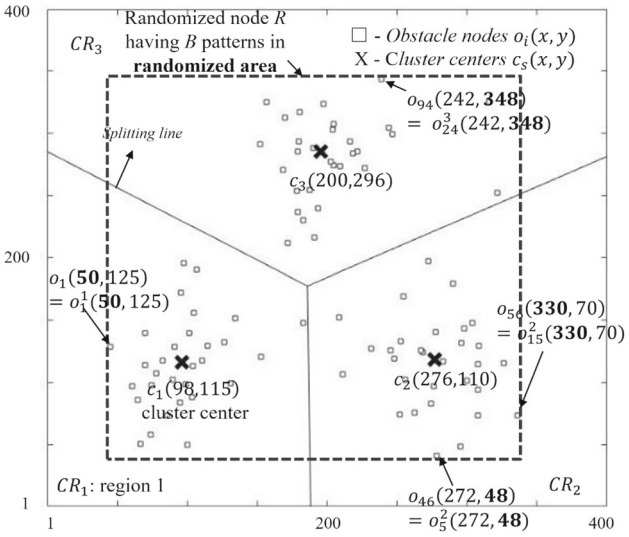


Fig. 5 Splitting map by k-means with Voronoi diagram. Step 1: Make the randomized area to find the k-means clustering ; Step 2: Find the best k-value ; Step 3: Input c_1, c_2, c_3 from the k-means clustering ; Step 4: Divide the cluster region CR using the distance between $o_j^s(x, y)$ and $c_s(x, y)$; Step 5: Calculate and draw the Voronoi diagram using Eq. (12)

According to the position of cluster centers, the map can be divided into a few regions using the gap statistics method. To divide the splitting line of each separate region, the Voronoi diagram is applied. The Voronoi diagram is a data structure investigated extensively in the domain of computational geometry. Originally, the Voronoi diagram characterizes the regions of proximity for a set of sites in a plane where the distance is defined by the Euclidean distance [36]. In this paper, the Voronoi diagram is applied for grouping every point into the nearest cluster region.

For the Voronoi diagram, the ground nodes and the cluster center of the map selected by the k-means algorithm are required. The distance between a ground node (x, y) and the cluster center $c_s(x, y)$ in the cluster regions CR_s are defined as

$$d_s = |(x, y) - c_s(x, y)|^2. \tag{11}$$

Then, the regions are divided by the splitting lines based on the condition given as

$$d_s = d_{s'},\ c_s \neq c_{s'}. \tag{12}$$

After applying the k-means clustering algorithm with the Voronoi diagram, the large map can be split into k-regions with splitting lines shown on the map. An example of the k-means clustering combined with the Voronoi diagram splitting a map is shown in Fig. 5.

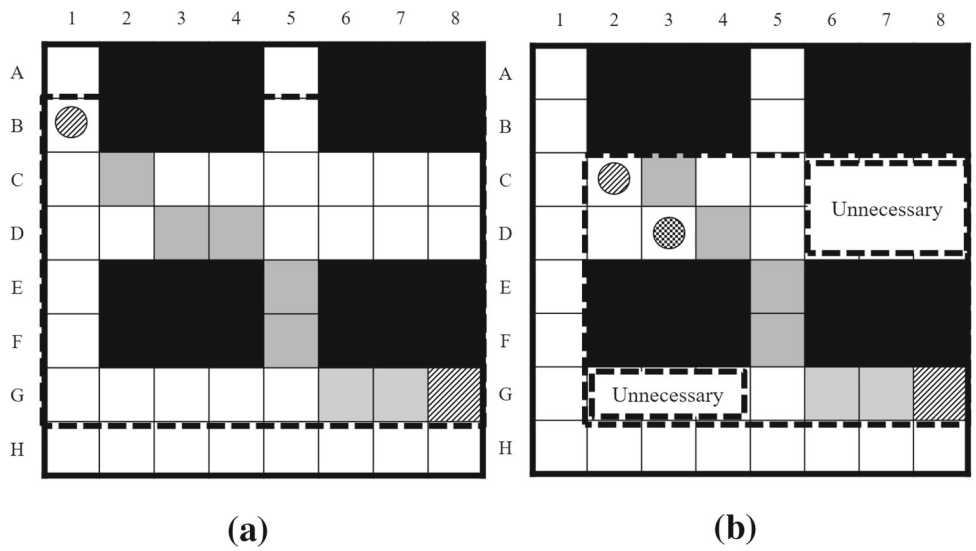


Fig. 6 Unnecessary dynamic areas. **a** Expanded area in traditional D* lite, **b** unnecessary area while dynamic path planning runs

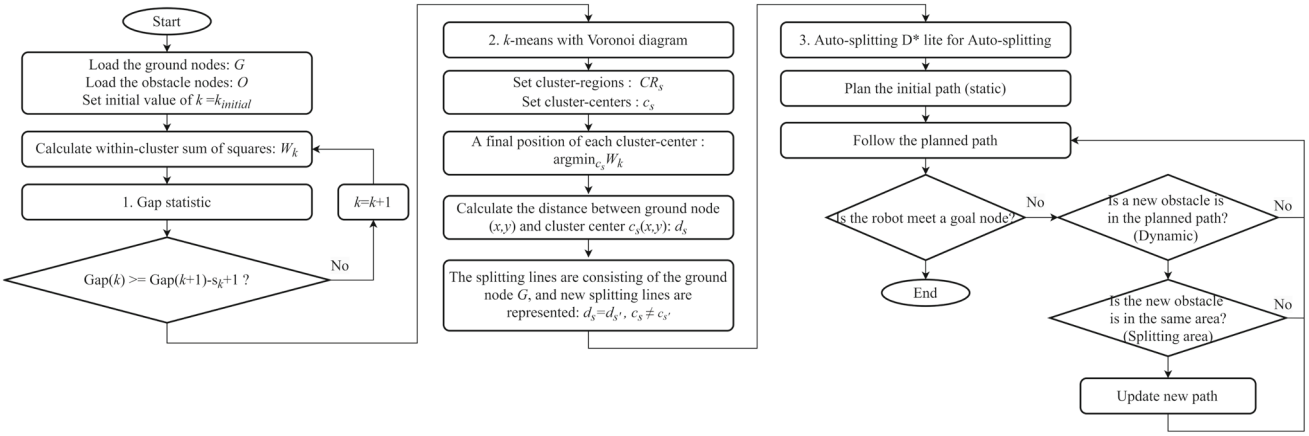


Fig. 7 Flowchart of auto-splitting D* lite

3.2 Auto-splitting D* lite

The issues of the D* based path planning algorithms containing unnecessary areas are mentioned in the previous sections. To reduce the unnecessary areas of the D* based path planning algorithms, this paper proposes a way to pre-split the map using the AS-D* lite. The AS-D* lite adopts an automatic clustering algorithm that automatically calculates the optimal map segmentation based on the location information of obstacles.

The unnecessary areas are defined as the areas that are not required for the path planning algorithm when calculating from a current node to a target node. Let us have a close look at Fig. 6, it can be seen that C6, C7, C8, D6, D7, D8, G2, G3, and G4 are the unnecessary areas. To reduce these unnecessary areas, the method of generating the splitting map in advance when calculating the path is described in Fig. 7. It is represented in three steps as follows:

Step 1. Plan the shortest path in an initial map state. (static path planning)
Step 2. Split the large map using the auto-splitting method. (auto-splitting method)
Step 3. Update the path if the map changes in the splitting area where the robot exists. (dynamic path planning to avoid moving obstacle) Else, keep the path. (dynamic path planning to reduce unnecessary areas)

Figure 8 shows an example of re-planned nodes by traditional D* lite, while Fig. 9 shows the re-planned nodes by the AS-D* lite accordingly. Circle with slashed-area in Fig. 8a represents a current position, Node with slashed-area represents a goal node (D5), and Circle with checked-flag (D3) represents the moving obstacle (D3→ C4 → B5), respectively. Also, Black nodes represent the obstacle nodes (C1, C2, and C3), White nodes represent the movable nodes, and Gray nodes represent the planned nodes, respectively.

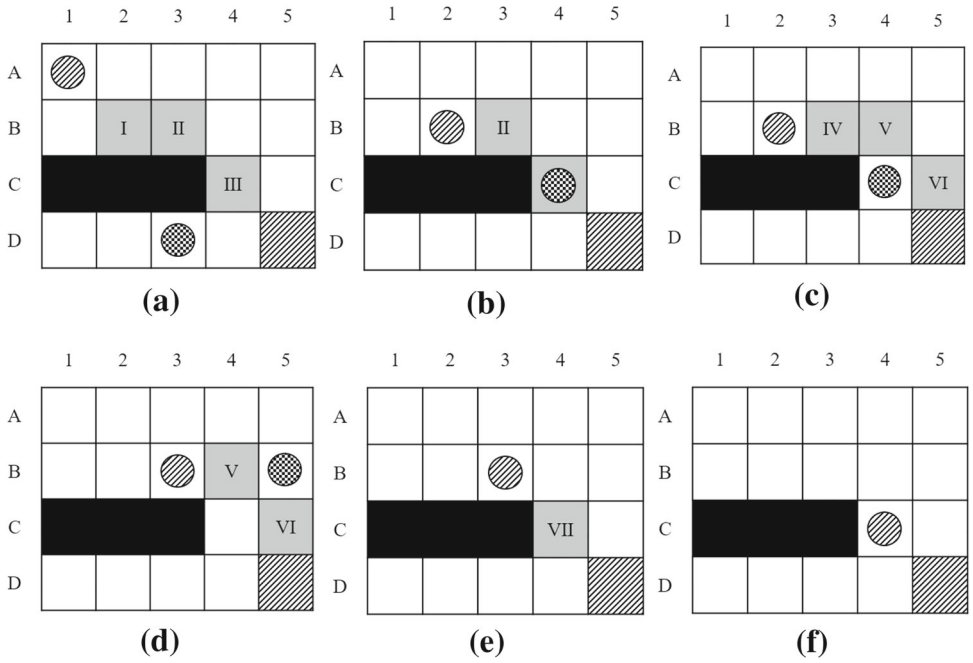


Fig. 8 Re-planned nodes by traditional D* lite. **a** Planned path (S-I-II-III-G), **b** dynamic obstacle block the path (II-III-G), **c** new path generated (IV-V-VI-G), **d** dynamic obstacle escape the path, **e** new path generated (VII-G), **f** find the goal (**g**)

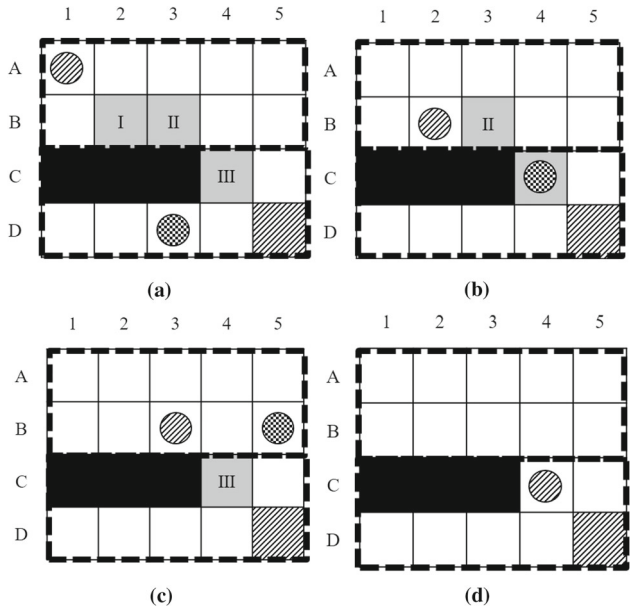


Fig. 9 Re-planned nodes by auto-splitting D* lite. **a** Planned path (S-I-II-III-G), **b** dynamic obstacle block the path (II-III-G), **c** unnecessary node reducement (III-G), **d** moved to next area

As shown in Fig. 9a, the rescue robot(A1) is in area 1 and moving to the goal(D5). At this point, the planned path is A1, B2, B3, C4, and D5, and the moving obstacle(D5) appears in the area 2. Figure 9b shows the second step of the rescue robot(B2), and it remains in the area 1 and is moving to the goal(D5). The remaining planned path at this step is B2, B3, C4, and D5. The moving obstacle(C4) is currently moving in the area 2 and moves to the planned path B3, C4, and D5, but the AS-D* lite algorithm does not re-plan the path because the rescue robot and the moving obstacle are not existing in the same area. Figure 9c shows the third step of the rescue robot(B3), and it still remains in the area 1 and is moving to the goal(D5). The remaining planned path is C4 and D5. The moving obstacle(B3) is now in the area 1, but it is not blocking the path. Therefore, the algorithm does not re-plan the path. Figure 9d shows the fourth step of the rescue robot(C4). The rescue robot enters the area 2 and is moving to the goal(D5). The only remaining node of the planned path is D5. Now, the moving obstacle has disappeared. The goal and planned path remained in the same area. The AS-D* lite shows a significant difference in the total number of re-planned nodes(RPN) comparing to the traditional D* lite. Here, RPN is defined as the number of re-planned nodes. Then, the 7 re-planned nodes(RPN) are generated in the traditional D* lite in total, but only 3 re-planned nodes(RPN) are generated in the AS-D* lite in the case of Figs. 8 and 9.

3.3 Analysis of auto-splitting D* lite

The proposed AS-D* lite is an algorithm that removes unnecessary areas with the auto-clustered methods based on the obstacles' locations. Therefore, theoretical analysis such as the time complexity of the AS-D* lite is discussed. In general, it is difficult to calculate the time complexity of the AS-D* lite algorithm. First, the auto-clustering algorithm re-calculates when the static map updates globally. However, the path planning algorithm re-plans whenever the dynamic obstacle changes. Secondly, the number of obstacle nodes is

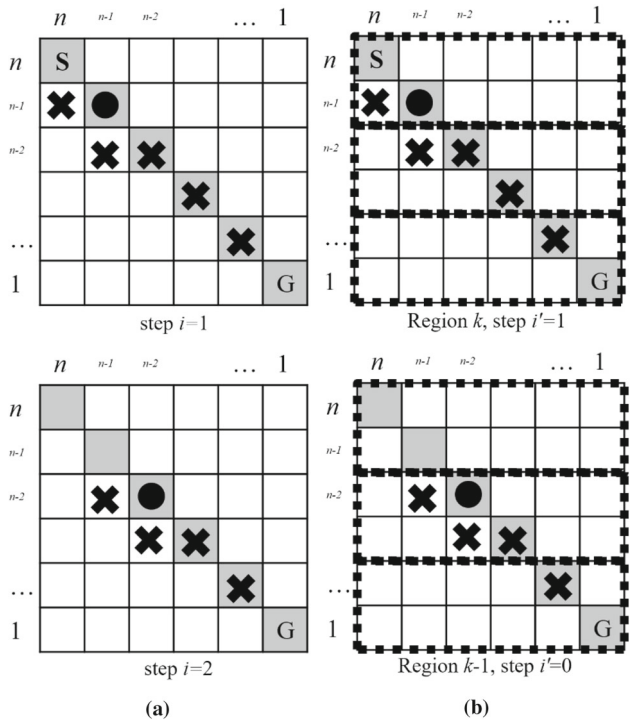


Fig. 10 Number of re-planned node(RPN) in $n \times n$ nodes. **a** Re-planned nodes of Traditional D* lite until step $i = n - 1$, **b** re-planned nodes of AS-D* lite until step $i' = m - 1$ (m represents the number of the planned nodes in region k)

a unit for the auto-clustering, but the number of grids n in a map is another unit for the time complexity in path planning. Therefore, an expected value for the number of re-planned nodes E(RPN) is newly defined to represent as the calculation time that varies according to the map size and k-value. E(RPN) is introduced to analyze the traditional D* lite and the AS-D* lite using the total number of re-planned nodes (RPN) as shown in Fig. 10.

In Fig. 10, S node represents a start node (x_n, y_n), G node represents a goal node (x_1, y_1), Gray nodes represents planned path nodes and Black circle represents an obstacle node. Especially, the obstacle node that positioned in planned path node means the obstacle blocks the planned path. Continuously, X marks represent re-planned node r, and Dashed lines represent k-region $(1, 2, \ldots, j, \ldots, k)$ which is the same number of k-value. Finally, $i\,(i = 0, 1, 2, \ldots, n-1)$ is the number of step for traditional D* lite, and i' is for AS-D* lite.

For the traditional D* lite:

Let us consider a square map of n[node]$\times n$[node], a start node (x_n, y_n) and a goal node (x_1, y_1) in the map. Also, a robot moves from the start node to the goal-node along the planned path nodes as

$$(x_n, y_n), (x_{n-1}, y_{n-1}), (x_{n-2}, y_{n-2}), \ldots, (x_1, y_1)$$

Table 1 Procedure of E(RPN) in traditional D* lite

T_n	(x_n, y_n) to (x_1, y_1)
$R(T_n)$	n
$P(T_n)$	$\frac{n}{n^2}$
$E(T_n)$	$\frac{n}{n^2} \cdot n$
T_{n-i}	(x_{n-i}, y_{n-i}) to (x_1, y_1)
$R(T_{n-i})$	$n - i$
$P(T_{n-i})$	$\frac{n-i}{n^2}$
$E(T_{n-i})$	$\frac{n-i}{n^2} \cdot n - i$
T_1	(x_1, y_1) to (x_1, y_1)
$R(T_1)$	1
$P(T_1)$	$\frac{1}{n^2}$
$E(T_1)$	$\frac{1}{n^2} \cdot 1$

The robot moves one node in every step. Assume a random obstacle node with $U(1, n^2)$ appears anywhere on the map.

List of symbols for E(RPN) - AS-D* lite

r	Number of the re-planned nodes
i	Number of steps. $(i = 0, 1, 2, \ldots, n - 1)$
T	Event of the planned path being re-planned if the random obstacle appears on any planned path node that blocks the path.
T_n	Event T happens from (x_n, y_n) to (x_1, y_1)
$R(T_n)$	Number of the re-planned nodes when T_n occurs
$P(T_n)$	Probability of a re-planned path generated

If the robot is moving on the planned path and the random obstacle appears on any planned path node that blocks the path, then the traditional D* lite generates $r(r = n, n - 1, \ldots, 1)$ re-planned nodes. The location of the robot (x_n, y_n) changes to the (x_{n-1}, y_{n-1}). After one step, the previous node (x_n, y_n) is deleted from the planned path nodes. When the robot doesn't have any movement, then the number of step is $i = 0$; when the robot moves from (x_n, y_n) to (x_{n-1}, y_{n-1}), the number of step is $i = 1$; and the maximum number of steps represents as $i = n - 1$. For the traditional D* lite, the expected value for the number of the re-planned nodes E(RPN) is represented as

$$\begin{aligned} E\,(\text{RPN}) &= \sum_{i=0}^{n-1} P\,(T_{n-i})\,R\,(T_{n-i}) \\ &= \sum_{i=0}^{n-1} \frac{1}{n^2}(n-i)^2. \end{aligned} \tag{13}$$

Table 1 shows the outlines of the entire process for computing E(RPN) in the Traditional D* lite.

For the Auto-Splitting D* lite:

Let us consider the square map which is split into k-regions by the AS-D* lite algorithm. Also, assume each k-region $(1, 2, \ldots, j, \ldots, k)$ contains the same number of planned path nodes n/k (natural number). Here, k is a number of regions split by the k-value, and $j(j = k, k-1, \ldots, 1)$ are any area among the regions automatically split by k. Then, the planned path nodes starting from k-region through j-region to the final region 1 are represented as

$$\begin{aligned}
&\left(x_{\frac{nk}{k}},\ y_{\frac{nk}{k}}\right), \left(x_{\frac{nk}{k}-1},\ y_{\frac{nk}{k}-1}\right), \left(x_{\frac{nk}{k}-2},\ y_{\frac{nk}{k}-2}\right),\\
&\quad \cdots, \left(x_{\frac{nk}{k}-(\frac{n}{k}-1)},\ y_{\frac{nk}{k}-(\frac{n}{k}-1)}\right),\\
&\quad \cdots, \left(x_{\frac{nj}{k}},\ y_{\frac{nj}{k}}\right), \left(x_{\frac{nj}{k}-1},\ y_{\frac{nj}{k}-1}\right), \left(x_{\frac{nj}{k}-2},\ y_{\frac{nj}{k}-2}\right),\\
&\quad \cdots, \left(x_{\frac{nj}{k}-(\frac{n}{k}-1)},\ y_{\frac{nj}{k}-(\frac{n}{k}-1)}\right),\\
&\quad \cdots, (x_1,\ y_1).
\end{aligned}$$

List of symbols for E(RPN) - AS-D* lite

r	Number of the re-planned nodes in each region
m	Number of the planned nodes in region k
i'	Number of steps in each region. $(i' = 0, 1, 2, \ldots, m-1)$
A	Event of the planned path being re-planned if the random obstacle appears on any planned path node that blocks the path
A_{km}	Event A happens from (x_{km}, y_{km}) to (x_1, y_1) in region k
$R(A_{km})$	Number of the re-planned nodes when A_{km} occurs
$P(A_{km})$	Probability of a re-planned path generated

If the robot is moving on the planned path in the region j, and the random obstacle appears on any planned path node that blocks the path in the region j, then the AS-D* lite generates the number of re-planned nodes $r(r = n, n-1, \ldots, 1)$. The node of the robot (x_{km}, y_{km}) changes to the (x_{km-1}, y_{km-1}) in the region k. After one step, the previous node (x_{km}, y_{km}) is deleted from planned path nodes. For the AS-D* lite, the expected value for the number of the re-planned nodes E(RPN) is represented as

$$\begin{aligned}
E\ (\mathrm{RPN}) &= \sum_{i'=0}^{m-1}\sum_{j=1}^{k} P\left(A_{km-i'}\right) R\left(A_{km-i'}\right)\\
&= \sum_{i'=0}^{m-1}\sum_{j=1}^{k} \frac{1}{n^2}\left(m-i'\right)\left(jm-i'\right).
\end{aligned} \tag{14}$$

Table 2 shows the outlines of the entire process for computing E(RPN) in the AS-D* lite. Not only area j but also area k should be calculated and added. The E(RPN) in different map sizes n for Eqs. (13) and (14) are shown in Fig. 11. The x-axis shows the size of the map (n[node]×n[node]) and the y-axis shows the expected value for the number of the re-planned nodes E(RPN).

Table 2 Procedure of E(RPN) in AS-D* lite

A_{km}	First node to last node in region k
$R(A_{km})$	n
$P(A_{km})$	$\frac{n/k}{n^2}$
$E(A_{km})$	$\frac{n/k}{n^2} \cdot n$
$A_{km-i'}$	i' step of node to last node in region k
$R(A_{km-i'})$	$\frac{(nk)}{k} - i'$
$P(A_{km-i'})$	$\frac{(n/k)-i'}{n^2}$
$E(A_{km-i'})$	$\frac{(n/k)-i'}{n^2} \cdot (\frac{nk}{k} - i')$
A_{11}	Last node to last node in region k
$R(A_{11})$	1
$P(A_{11})$	$\frac{1}{n^2}$
$E(A_{11})$	$\frac{1}{n^2} \cdot 1$

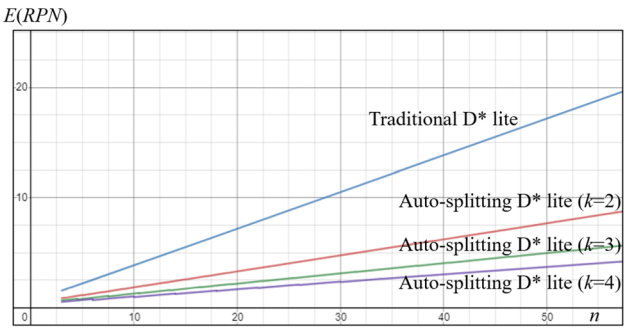


Fig. 11 E(RPN) in different map sizes n for Eqs. (13) and (14)

3.4 Time complexity of auto-splitting D* lite

The calculation time of the grid-based path planning algorithms is based on the expansion of the nodes collected on the open and closed list by the non-descending order. Therefore, the data structure and input arrays are more important than the structure of the algorithm. The time complexity is as follows if we focus on one-dimensional node movement. D* lite algorithms run like an A* algorithms, so the time complexity of the first iteration is $O(n)$ (where n is the total number of nodes in graphs) in the worst case. For the path updates, the number of obstacles that have been updated can be much more than A* in the worst case. However, this research takes place on a two-dimensional map rather than a one-dimensional graph. Accordingly, the time complexity increases to $O(n^2)$, and this is also the reason why the binary heap sorting $O(nlog(n))$ is basically used to reduce

the time complexity. Most of the searching algorithm's typical approach is to use a binary heap if the structure is the same as this research [41]. This research also uses the binary heap, so a Big-O notation for time complexity is the same. Based on the Big-O notation, the time complexity of D* lite is $O(nlog(n))$, where n is the total number of the nodes in the map in the best case. However, the AS-D* lite path planning algorithm's best cases are represented by $1/k$ node-expansion, and it is influenced by the total amount of the data. The number of basic operations of the whole algorithm can be expressed as:

$$\begin{aligned} T_{\text{total}} &= T_{\text{pre}} + T_{\text{main}} + T_{\text{post}} \\ T_{\text{main}} &= 1/k\ O(nlog(n)) \end{aligned} \tag{15}$$

where n is the number of nodes in the map; T_{pre} is a constant number of basic operations before entering the main loop; T_{post} is a constant number of basic operations after the main while-loop. T_{main} is $1/k\ O(nlog(n))$. It can be written that the time complexity of the AS-D* lite path planning algorithm is expressed as $O(nlog(n))$ in worst cases, and the dominant is $1/k$ compared to traditional D* lite theoretically. The time complexity of heuristic search algorithms such as BFS (Breadth-first search), Dijkstra's, and A* changes depending on the sorting methods [42,43]. So it is better to compare the expected value for the number of re-planned nodes and the average update nodes in different map sizes as shown in Figs. 12 and 13. The details are described in the simulation section.

To analyze the time complexity of the preprocessing part T_{pre} of the map segmentation, it is necessary to understand the theoretical time complexity of the k-means clustering and the time complexity of other methods comparable to the k-means. The k-means based clustering methods such as the k-means, k-modes [44], and k-medoids [45] are compared, and the density-based mean-shift algorithm [46] is compared as follows.

To analyze the time complexity of the k-means based algorithm according to this paper, a few variables c, k, and i need to be introduced beforehand. The variable c represents the number of obstacle nodes in the k-means based algorithm. Note that we used the commonly expressed variable n in the T_{main} part, while the number of obstacle c is used instead of n in the T_{pre} part. As the size of the map increases, the number of obstacles also increases, so variable c is adjusted. k represents the number of regions and is fixed to analyze the time complexity according to the size of the map. i is the iterations until convergence, and it is set with the maximum iterations in this study. As an important condition, $k << c$ and $i << c$ are required.

Firstly, the k-means-based clustering methods have $O(cki)$ time complexity. In this paper, the k-means aims to partition the number of obstacles c into k-regions in which each

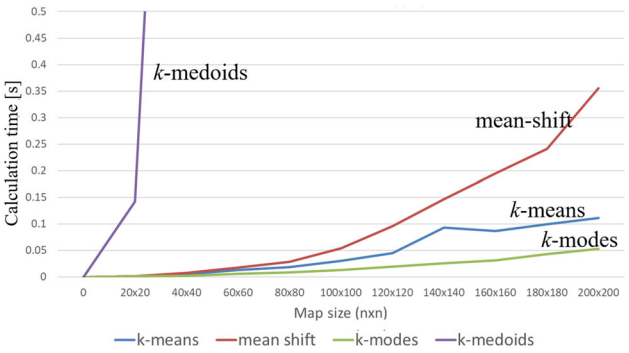


Fig. 12 Real-time performance (T_{pre})

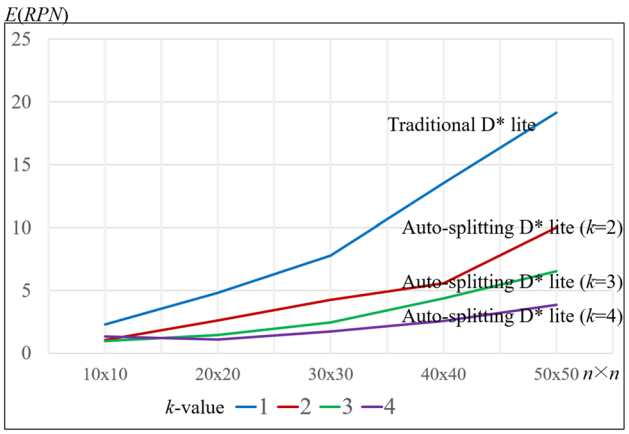


Fig. 13 E(RPN) in different map sizes n

obstacle belongs to the regions with the nearest mean (cluster centers or cluster centroid). The increment of obstacles can be interpreted in the same way as the map size increase. For accurate analysis with other algorithms, an increase in the c value is focused, k=10 is fixed, and the maximum iterations are set to 100. Therefore, it is predictable that the time complexity increases with c.

Secondly, the k-modes-based clustering methods also has $O(cki)$ time complexity. The k-modes is an extended method of the k-means but using the nearest modes (most frequent value) instead of the nearest mean. Using the nearest modes in the k-modes has advantages to categorical data types. Compared with k-mean, the information of obstacle's position is assumed as categorical data. All the values of c,k and i are the same as the k-means.

Thirdly, a representative algorithm of the k-medoids-based clustering called PAM (Partitioning around medoids) has $O(ik(c-k)^2)$ time complexity [47]. In contrast to the k-means clustering algorithms, the k-medoids clustering algorithms choose actual data points as centers (medoids or exemplars). Thereby, it allows for greater interpretability of the cluster centers than in the k-means algorithms, where the center of a cluster is not necessarily one of the input data points. All the remaining medoids $(c-k)$ aimed to find the

set of medoids that has the minimum cost function. The loop would be k for looping through all the medoids $(c-k)$. Then it will be $(c-k)$ to loop through all the non-medoid data points. Then $(c-k)$ again for choosing the random medoid. Therefore, the k-medoids have $O(ik(c-k)^2)$ time complexity if the iteration is i. The analysis focuses on the increase in c value, k=10 is fixed as the other methods. However, the maximum iterations are set to 10 because of the slow convergence.

Finally, the mean-shift-based clustering methods have $O(c^2 i)$ time complexity. It clustering algorithm is a mode-seeking algorithm. The mean-shift is a hill-climbing algorithm that involves shifting kernels iteratively to a higher density region until convergence. Therefore, a bandwidth of the kernel is used to simulate random samples. It means, instead of c, the bandwidth would increase when the map size changes. At every iteration, the kernel is shifted to the centroid or the mean of the points within it. In the case of mean-shift algorithms, the time complexity can be expressed in the same way as the k-means algorithms, but due to the characteristic of the mean-shift algorithms, c increases only when the bandwidth of the kernel increases. Therefore, the bandwidth of the kernel is adjusted in this paper. The bandwidth according to the map size is as follows: $20 \times 20 - 1.8, 40 \times 40 - 3.6, 60 \times 60 - 5.4, 80 \times 80 - 7.2, 100 \times 100 - 9, 120 \times 120 - 10.8, 140 \times 140 - 12.6, 160 \times 160 - 14.4, 180 \times 180 - 16.2, 200 \times 200 - 18$.

Figure 12 shows the simulation of real-time performance for the T_{pre}. Random obstacles are set to 25% of the map. Therefore, c is $0.25n^2$, but the mean-shift uses the bandwidth. The important point is that even if c is $0.25n^2$, the actual c and n should not be substituted. Since this represents the order in the time complexity, T_{main} and T_{pre} are separated. The result shows that the changes in the calculation time are almost the same as the theoretical time complexity. In terms of performance, the k-modes algorithms are the fastest, and the k-medoids algorithms are judged to be difficult to use. In terms of the scalability of the algorithm, the k-means algorithm has the best balance. Therefore, the k-means algorithm is selected. The k-modes algorithm is also possible to be selected due to its advantages for data types with categorical variables.

In this section, the reason for the algorithm selection is explained through time complexity analysis in various situations of T_{main} and T_{pre}. In addition to the clustering algorithm, T_{pre} includes a method for determining the k-value and separating the map. Since calculations are added according to the time complexity of these clustering algorithms, the order follows the higher order of the clustering algorithm. The real-time performance for the entire T_{pre} is shown in Sect. 4.1. Auto-splitting method for Auto-clustering and drawing.

Table 3 Calculation time of auto-splitting method [s]

Map size	k-means	mean-shift	k-modes	k-medoids
20×20	0.0014	0.0015	0.0011	0.1419
40×40	0.0049	0.0075	0.0027	1.9937
60×60	0.0128	0.0176	0.0060	9.7364
80×80	0.0184	0.0290	0.0085	31.2622
100×100	0.0303	0.0537	0.0131	77.6559
120×120	0.0453	0.0958	0.0192	162.7345
140×140	0.0927	0.1467	0.0259	296.2298
160×160	0.0870	0.1949	0.0311	509.5957
180×180	0.0994	0.2408	0.0427	798.9193
200×200	0.1115	0.3558	0.0534	1227.0020

Table 4 Calculation time of auto-splitting method

Method	Calculation time
Silhouette coefficient approach with k-means	11.47[s]
Gap statistics approach with k-means	2.50[s]

4 Simulations

4.1 Auto-splitting method for auto-clustering and drawing

Simulation for the AS-D* lite algorithm is carried out in the Unity 3D environment. Simulation PC has a performance of CPU: i7-7700HQ, GPU: GTX 1060, RAM: 16GB. In the AS-D* lite, the auto-clustering methods are able to segment the maps automatically. The ways to determine the k-value are compared with the auto-clustering methods such as the Silhouette coefficient and the gap statistics. All algorithms are applied with the Voronoi diagram. The simulation condition of the gap statistics and the Silhouette coefficient is given at 40,000 nodes, which will be the same as the city map simulation shown later.

Theoretically, the calculation time itself is not large if excluding the clustering part, but the total calculation process takes a long time in the simulation due to the reason that graphical expressions are used in the map. As shown in Table 3, the gap statistics are running with the k-means algorithm, so the time complexity is the same. However, the Silhouette coefficient has higher order than the time complexity of the k-means algorithm. Therefore, it takes more time to calculate. From Tables 3 to 4, the k-means and k-modes algorithms have the lowest time complexity. The k-means algorithms use the nearest mean and have advantages for numerical variables. The k-modes algorithms use the nearest modes (most frequent) and have advantages for categorical variables. Both k-means based clustering algorithms are fast for larger maps. We decide to apply the k-means algo-

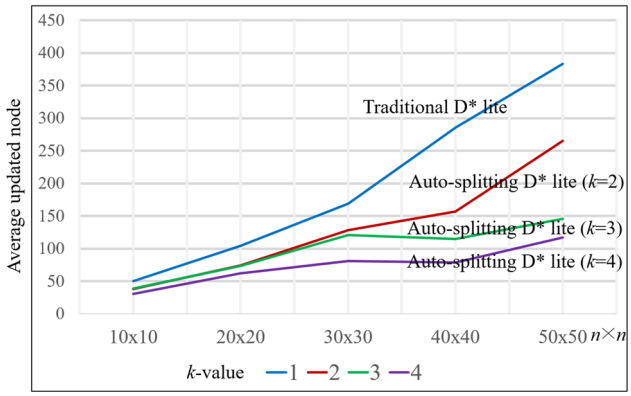
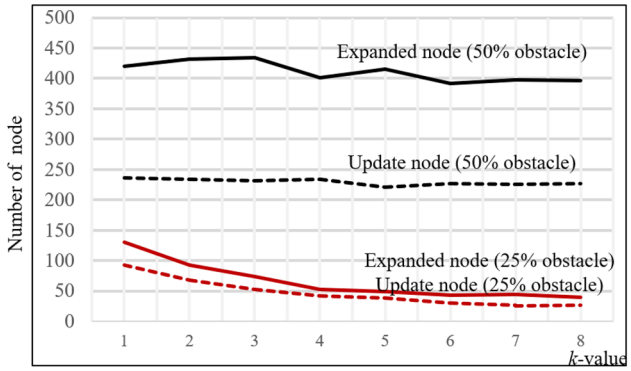


Fig. 14 Average updated node in different map sizes



(k=1 : Traditional D* lite, $k \geq 2$: AS-D* lite)

Fig. 15 Number of expanded node and updated node for different ratios of random obstacles

rithm because of the scalability it has. Moreover, there is not much significant difference between the calculation speed of k-modes algorithms and k-means algorithms when the gap statistics are applied. When the gap statistics is applied, the k-value can be determined about 80% faster than the other traditional methods. The algorithm that determines the k-value only works once until the background map is completely changed. Moreover, this simulation focuses on the preprocessing part, so only the calculation time is analyzed as shown in Table 4. The performance of the AS-D* lite is evaluated using the expected value for the number of the re-planned nodes E(RPN).

4.2 Auto-splitting D* lite

4.2.1 Simulation for special case of auto-splitting D* lite

The following simulations are a validation test using E(RPN). Performance of the traditional D* lite and the AS-D* lite under the condition of k=2,3, and 4 is compared with each other. The total re-planned nodes for each case n are calculated. The special case of the AS-D* lite assuming each k-region $(1, 2, \ldots, j, \ldots, k)$ has the same number of planned path nodes as $m = n/k$ (natural number) described in Sect. 3.3. The simulation conditions are set to satisfy the natural numbers as follows: 50×50 is replaced by 51×51 or 52×52 when k=3 or 4, respectively; 40×40 is replaced by 42×42 when k=3; 30×30 is replaced by 32×32 when k=4; 20×20 is replaced by 21×21 when k=3; 10×10 is replaced by 12×12 when k=3 and 4. All simulations are conducted 5000 times under the same conditions as shown in Fig. 11 to validate the expected value for the number of the re-planned nodes E(RPN).

Figure 13 shows E(RPN) in different map sizes using simulations. The overall error rate is around 10% comparing to Fig. 11. Figure 14 shows the average updated nodes in different map sizes. The updated node contains 8 surrounded nodes of the current node. Also, the updated nodes are possible to overlap with each other if the re-planned paths are created multiple times. Unlike E(RPN), the average number of updated nodes is re-calculated when any value in the open-list of the D* lite algorithms changes [2].

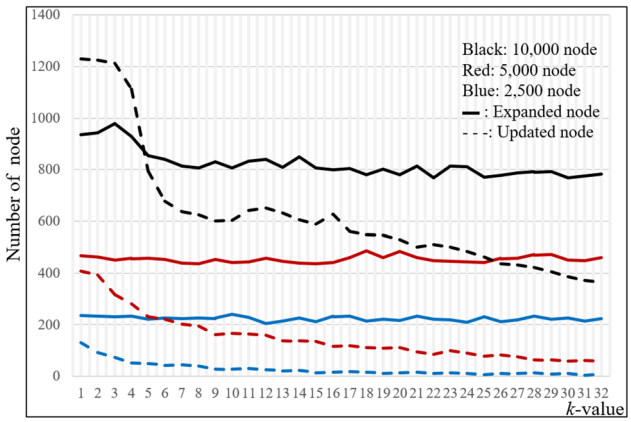
4.2.2 Simulation for different ratios of random obstacles

Figure 15 shows simulation results with different amounts of the obstacles in the same map size. The simulation shows the difference in the reduction rate of the updated nodes resulting from the change in the value for the number of regions k in a map of 2500[node] for 3 cases: random static obstacles, two random dynamic obstacles, and a random start position. In particular, the random static obstacles refer to the density of obstacles on the map. The density of the obstacles on the map is set to 25% to represent as a rural area, and 50% is set to represent as a city area. Each simulation is repeated 500 times, where region k=1 shows the traditional D* lite and region $k \geq 2$ shows the AS-D* lite.

Black lines represent the simulations in an environment with 50% of static obstacles, and Red lines represent the simulations in an environment with 25% of static obstacles. Solid lines show the expanded node, and Dotted lines show the updated node. As shown in Fig. 15, the updated node and the expanded node are reduced by 50%, when the density of the random obstacle is reduced by 50%. Also, large sizes of the map are simulated to show the efficiency of the AS-D* lite.

4.2.3 Simulation for different map sizes

Figure 16 shows a simulation in the map of different sizes with the same proportion of random obstacles. The maps of 2500[node], 5000[node], and 10,000[node] are compared to each other as shown in Fig. 17. The random static obstacles are set at 25% for all three maps, but the number of random dynamic obstacles is set at 2, 4, and 8 for the three maps,



(k=1 : Traditional D* lite, $k \geq 2$: AS-D* lite)

Fig. 16 Number of expanded node and updated node for different map sizes

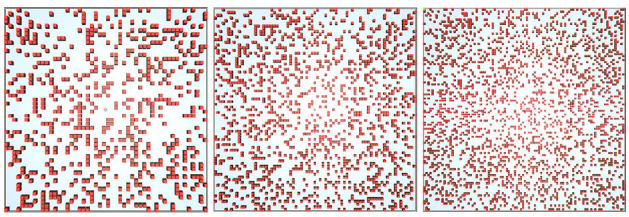


Fig. 17 Node number of test map size. **a** 2500 [node], **b** approximately 5000 [node] (exactly 5041 [node]), **c** 10,000 [node]

respectively. Each simulation is repeated 500 times. The k-value is set from 1 to 32 for showing the difference in the number of expanded nodes and the updated node.

The overall result can be seen that the performance of the AS-D* lite algorithm is better as the map size increases. The results of the AS-D* lite algorithm shows that number of region k=7 or 8 is automatically selected on 2500[node], k=4 on 5000[node], and k=5 or 6 on 10,000[node]. The k-value varies greatly depending on the random obstacle of the reference data set.

4.2.4 City map simulation with fire (Scenario 1)

This simulation is carried out on a city map for a larger environment near Kokura Castle in Kitakyushu, Japan. Image data of 2048×2048[pixel] taken at a height of 1[km] is selected. The actual map has an axial distance of 1.2[km] and a map size of 40,000[node] is constructed. The size of each node is 6[m]×6[m]. The size of one node was calculated based on the area of the two-lane road in Japan. The width of the lane on the Japanese road is a general two-lane road which is commonly seen, and it is approximately 3.0[m] wide per lane, 3.25[m] on the main road and 3.5 – 3.75[m] on the highway. The minimum possible range of global path

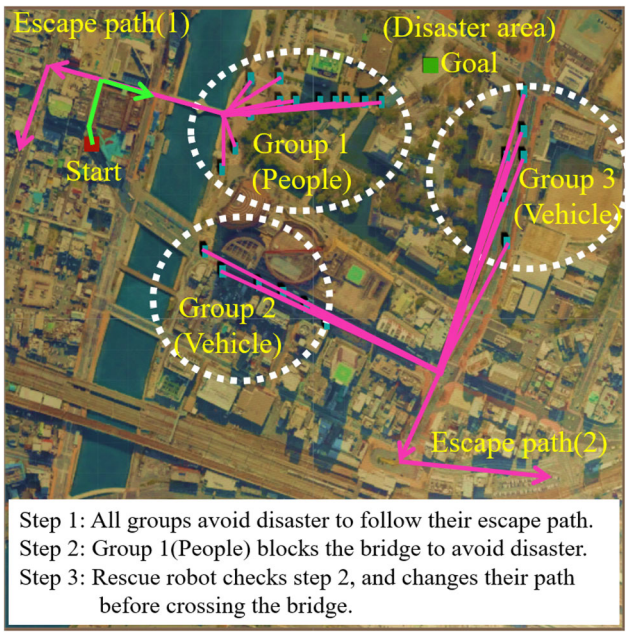


Fig. 18 Scenario 1 of rescue robot path planning in large city map

planning was set. Also, it is possible with a smaller node size on a larger map.

Simulations of scenario 1 are shown in Fig. 18. Red spot is a starting point and Green is a goal point. Assuming that a fire or disaster occurred around the goal point, three dense groups move to two main escape points. At this point, group 1 is following the escape path (1); group 2 and group 3 are following the escape path (2). The rescue robot is finding the shortest path to the goal. The group 1 is escaping through a bridge, so the rescue robot re-calculates another path for dynamic obstacles. For the robot, the components of groups 1, 2, and 3 become dynamic obstacles. Figure 19a shows how the AS-D* lite splits the map. As a result, the k-value is selected as k=11 or 12. Compared to the traditional D* lite algorithm, the proposed AS-D* lite algorithm shows 18% reduction of the expanded nodes and 58% reduction of the updated nodes simulated in the city map case. The reduction of the expanded nodes means the reduction of unnecessary nodes, and the reduction of updated nodes means the reduction of re-planned nodes in the AS-D* lite algorithm. Figure 19b indicates a path of the robot according to the scenario.

Unlike the ideal environment, the number of updated nodes and ratio of expanded nodes does not decrease as much as the ideal environment because of the high ratio of obstacles and non-uniform obstacle density in the actual environment. The number of updated nodes is possible to be reduced by $1/k$ in the theoretical best case.

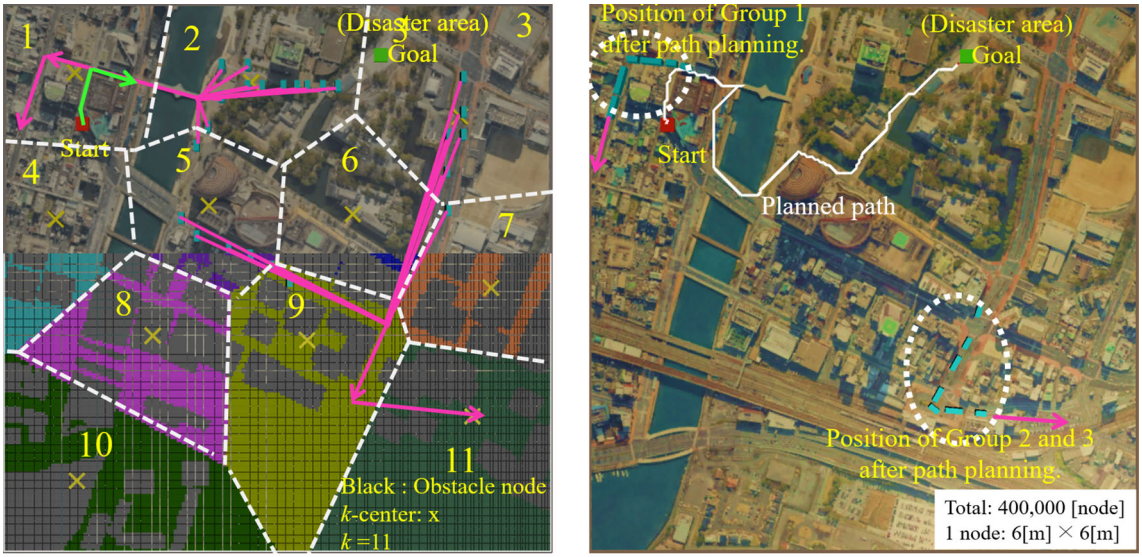


Fig. 19 Path planning in city map. **a** Auto-splitting map (k=11 is automatically generated), **b** planned path

4.2.5 Rural map simulation with landslide (Scenario 2)

Second simulation is carried out on a rural map for a larger environment near Nishimuro District in Wakayama-city, Japan. Image data of 2048×2048[pixel] taken at the height of 0.5[km] is selected. The actual map has an axial distance of 0.4[km], and a map size of 40,000[node] is constructed. The size of each node is 1[m]×1[m].

The scenario for rural maps is as follows:

Step 1. A rescue robot executes dynamic global path planning from the start position to the goal.
Step 2. A landslide occurs while the rescue robot is moving, blocking road 1 and road 2 at the same time.

Unlike the dynamic global path planning of the city map, the surrounding terrains are also dangerous in the case of landslides. Therefore, most of the environments (mountains, buildings, and rivers) except for roads are set as obstacles in this case. Each obstacle is given a base weight of 1, and each non-obstacle is given a base weight of 0. Moreover, the weight is separated into five different levels according to the potential danger each environment may cause. Figure 20b demonstrates how the different levels are given to every area on the map. Level 1 to level 5 represent from the most dangerous to the least dangerous. Level 1 stands for adding 50% of the obstacle's base weight, whereas level 5 stands for adding 10% of the obstacle's base weight. This weight doesn't exceed 1.5 times compared to the existing cost of the obstacle. Also, there are fields adjacent to landslides or lower ground which is also possible to be given a weight. Theoretically, the cost of obstacle nodes in traditional D* lite is infinite and is set by the user. However, since infinity cannot be used in actual applications, it was necessary to properly adjust the cost.

Simulations of scenario 2 are shown in Fig. 20a. The red spot is the starting point, and the green spot is the goal point. Assuming that landslides occurred on the mountain, the blue areas moved in the direction of the arrow and blocked the road. The rescue robot initially plans to travel in the shortest distance to get to the goal point, but primarily changes the path due to the influence of the landslide. The robot has encountered a landslide once again while moving on the first modified path, so it changed to the second modified path and arrived at the goal point. Figure 21a shows how the AS-D* lite splits the map. As a result, the k-value is selected as k=9 or 10. Compared to the traditional D* lite algorithm, the proposed AS-D* lite algorithm shows 11% reduction of the expanded nodes and 46% reduction of the updated nodes simulated in the rural map case. Figure 21b indicates the path of the robot according to the scenario. This simulation result shows that adding weight increases the calculation time compared to the city map due to the reason that every node is given a different level of weight. In this situation, field nodes with small costs are treated as obstacles, the number of the expanded nodes and updated nodes are reduced. Therefore, the calculation time is increased compared to the city map cases as described in Sect. 2, which the weighted method can sometimes lead to a slower search. In this case, although the calculation time is increased compared to the city map, it still performs better than traditional methods.

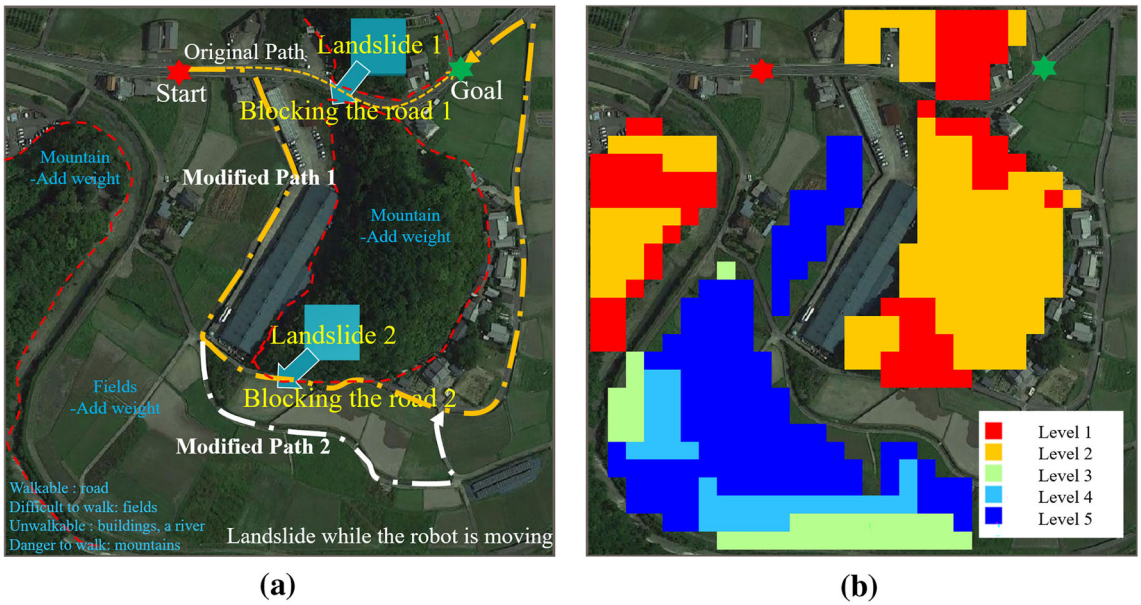


Fig. 20 Rescue robot path planning in rural map. **a** Scenario 2 of rescue robot path planning in rural map, **b** example of landslide alarm mapping

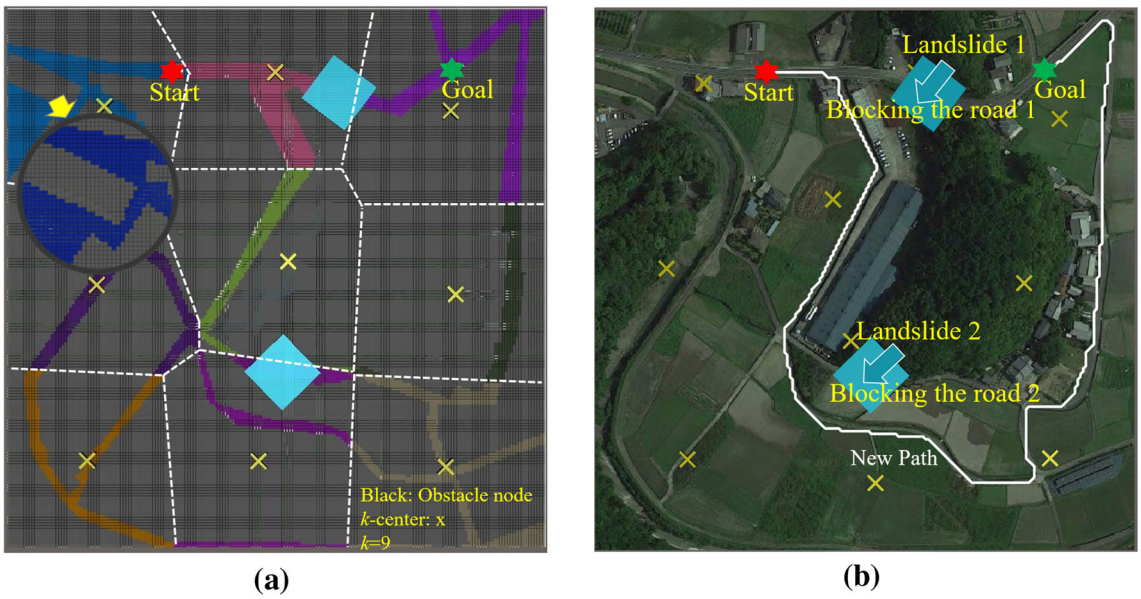


Fig. 21 Path planning in rural map. **a** Auto-splitting map (k=9 or 10 is automatically generated), **b** planned path

5 Conclusion

In this paper, the AS-D* lite algorithm was proposed to perform a dynamic and collision-free path planning task. Two important issues were considered and solved by path planning algorithms and auto-clustering methods. The first issue was the unnecessary area of the D* based path planning algorithms. The proposed AS-D* lite algorithm has solved the issue of the unnecessary area by ceasing the unnecessary calculation of the traditional D* lite, and alleviated the over-calculation issue of dynamic path planners in the partially known and large area. Furthermore, the path planning method was faster with the AS-D* lite algorithms when the environment was large and dynamic. The second issue was the slow processing time for determining the k-value and the way to determine the k-value automatically. To shorten the processing time for determining the k-value, the gap statistics method was applied to find the optimal k-value. Moreover, the Voronoi diagram was combined with the k-means algorithm to draw a segmented map automatically. In particular, the E(RPN) was introduced for representing the efficiency of the AS-D* lite algorithm according to the number of split

maps and the size of different maps. The E(RPN) is possible to be applied in various grid-based algorithms which operate in dynamic and partially known areas.

Author Contributions The first draft of the manuscript was written by Heo and provided the research ideas and theoretical analysis. Heo and Lee wrote the paper. Heo, Chen and Liao performed the simulation and edited the paper. All authors read and approved the manuscript.

Funding This work has no funding to report.

Availability of data and materials The datasets used and analyzed during the study are available from the first author on reasonable request.

Declarations

Conflict of interest Heo declares that he has no conflict of interest. Chen declares that he has no conflict of interest. Liao declares that she has no conflict of interest. Lee declares that he has no conflict of interest.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

1. Stentz A (1993) Optimal and efficient path planning for unknown and dynamic environments. Tech. rep., CARNEGIE-MELLON UNIV PITTSBURGH PA ROBOTICS INST
2. Koenig S, Likhachev M (2002) D^* lite. Aaai/iaai 15:476–483
3. Ferguson D, Stentz A (2006) Using interpolation to improve path planning: the field d* algorithm. J Field Robot 23(2):79–101
4. Zhong X, Tian J, Hu H, Peng X (2020) Hybrid path planning based on safe a* algorithm and adaptive window approach for mobile robot in large-scale dynamic environment. J Intell Robot Syst 99(1):65–77
5. Ebendt R, Drechsler R (2009) Weighted a* search-unifying view and application. Artif Intell 173(14):1310–1342
6. Liu X, Deng R, Wang J, Wang X (2014) Costar: a d-star lite-based dynamic search algorithm for codon optimization. J Theor Biol 344:19–30
7. Wilt CM, Ruml W (2012) When does weighted a* fail?. In: SOCS, pp 137–144
8. Kuffner JJ, LaValle SM (2000) Rrt-connect: an efficient approach to single-query path planning. In: Proceedings 2000 ICRA. Millennium conference. IEEE international conference on robotics and automation. Symposia proceedings (Cat. No. 00CH37065), vol 2, pp 995–1001, IEEE
9. Karaman S, Walter MR, Perez A, Frazzoli E, Teller S (2011) Anytime motion planning using the rrt. In: 2011 IEEE international conference on robotics and automation, pp 1478–1483, IEEE
10. Noreen I, Khan A, Habib Z (2016) A comparison of rrt, rrt* and rrt*-smart path planning algorithms. Int J Comput Sci Netw Secur (IJCSNS) 16(10):20
11. Wang J, Chi W, Li C, Wang C, Meng MQ-H (2020) Neural rrt*: learning-based optimal path planning. IEEE Trans Autom Sci Eng 17(4):1748–1758
12. Hwang YK, Ahuja N et al (1992) A potential field approach to path planning. IEEE Trans Robot Autom 8(1):23–32
13. Barraquand J, Langlois B, Latombe J-C (1992) Numerical potential field techniques for robot path planning. IEEE Trans Syst Man Cybern 22(2):224–241
14. Chen Y-B, Luo G-C, Mei Y-S, Yu J-Q, Su X-L (2016) Uav path planning using artificial potential field method updated by optimal control theory. Int J Syst Sci 47(6):1407–1420
15. Zhang B, Mao Z, Liu W, Liu J (2015) Geometric reinforcement learning for path planning of uavs. J Intell Robot Syst 77(2):391–409
16. Ravankar AA, Ravankar A, Emaru T, Kobayashi Y (2020) Hpprm: Hybrid potential based probabilistic roadmap algorithm for improved dynamic path planning of mobile robots. IEEE Access 8:221743–221766
17. Phung MD, Quach CH, Dinh TH, Ha Q (2017) Enhanced discrete particle swarm optimization path planning for uav vision-based surface inspection. Autom Constr 81:25–33
18. Das P, Jena PK (2020) Multi-robot path planning using improved particle swarm optimization algorithm through novel evolutionary operators. Appl Soft Comput 92:106312
19. Wang Z, Li G, Ren J (2021) Dynamic path planning for unmanned surface vehicle in complex offshore areas based on hybrid algorithm. Comput Commun 166:49–56
20. Peng Y, Li S-W, Hu Z-Z (2019) A self-learning dynamic path planning method for evacuation in large public buildings based on neural networks. Neurocomputing 365:71–85
21. Yonetani R, Taniai T, Barekatain M, Nishimura M, Kanezaki A (2021) Path planning using neural a* search. In: International conference on machine learning, pp 12029–12039, PMLR
22. Panov AI, Yakovlev KS, Suvorov R (2018) Grid path planning with deep reinforcement learning: preliminary results. Procedia Comput Sci 123:347–353
23. Drake D, Koziol S, Chabot E (2018) Mobile robot path planning with a moving goal. IEEE Access 6:12800–12814
24. Cheng P-Y, Chen P-J (2012) Navigation of mobile robot by using d++ algorithm. Intel Serv Robot 5(4):229–243
25. Van Den Berg J, Ferguson D, Kuffner J (2006) Anytime path planning and replanning in dynamic environments. In: Proceedings 2006 IEEE international conference on robotics and automation, 2006. ICRA 2006., pp 2366–2371, IEEE
26. Przybylski M, Putz B (2017) D* extra lite: a dynamic a* with search-tree cutting and frontier-gap repairing. Int J Appl Math Comput Sci, 27
27. Bholowalia P, Kumar A (2014) Ebk-means: a clustering technique based on elbow method and k-means in wsn. Int J Comput Appl, 105(9)
28. Frahling G, Sohler C (2008) A fast k-means implementation using coresets. Int J Comput Geom Appl 18(06):605–625
29. Arslan O, Guralnik DP, Koditschek DE (2016) Coordinated robot navigation via hierarchical clustering. IEEE Trans Rob 32(2):352–371
30. Almeida J, Barbosa L, Pais A, Formosinho S (2007) Improving hierarchical cluster analysis: a new method with outlier detection and automatic clustering. Chemom Intell Lab Syst 87(2):208–217

31. Schubert E, Sander J, Ester M, Kriegel HP, Xu X (2017) Dbscan revisited, revisited: why and how you should (still) use dbscan. ACM Trans Database Syst (TODS) 42(3):1–21
32. Zhang T, Ramakrishnan R, Livny M (1997) Birch: a new data clustering algorithm and its applications. Data Min Knowl Disc 1(2):141–182
33. Telgarsky M, Vattani A (2010) Hartigan's method: k-means clustering without Voronoi. In: Proceedings of the thirteenth international conference on artificial intelligence and statistics, pp 820–827, JMLR Workshop and Conference Proceedings, 2010
34. Senthilnath J, Kumar D, Benediktsson J, Zhang X (2016) A novel hierarchical clustering technique based on splitting and merging. Int J Image Data Fusion 7(1):19–41
35. Anand S, Mittal S, Tuzel O, Meer P (2013) Semi-supervised kernel mean shift clustering. IEEE Trans Pattern Anal Mach Intell 36(6):1201–1215
36. Reddy D, Jana PK, Member IS (2012) Initialization for k-means clustering using voronoi diagram. Procedia Technol 4:395–400
37. Zhou HB, Gao JT (2014) Automatic method for determining cluster number based on silhouette coefficient. Adv Mater Res 951:227–230
38. Kodinariya TM, Makwana PR (2013) Review on determining number of cluster in k-means clustering. Int J 1(6):90–95
39. Tibshirani R, Walther G, Hastie T (2001) Estimating the number of clusters in a data set via the gap statistic. J R Stat Soc Ser B (Stat Methodol) 63(2):411–423
40. Erwig M (2000) The graph Voronoi diagram with applications. Netw Int J 36(3):156–163
41. Niewola A, Podsedkowski L (2018) L* algorithm-a linear computational complexity graph searching algorithm for path planning. J Intell Robot Syst 91(3):425–444
42. Beamer S, Asanovic K, Patterson D (2012) Direction-optimizing breadth-first search. In: SC'12: Proceedings of the international conference on high performance computing, networking, storage and analysis, pp 1–10, IEEE
43. Dijkstra EW et al (1959) A note on two problems in connexion with graphs. Numer Math 1(1):269–271
44. Chaturvedi A, Green PE, Caroll JD (2001) K-modes clustering. J Classif 18(1):35–55
45. Arora P, Varshney S et al (2016) Analysis of k-means and k-medoids algorithm for big data. Procedia Comput Sci 78:507–512
46. Comaniciu D, Meer P (1999) Mean shift analysis and applications. In: Proceedings of the seventh IEEE international conference on computer vision, vol 2, pp 1197–1203, IEEE
47. Kaufman L, Rousseeuw PJ (2009) Finding groups in data: an introduction to cluster analysis, vol 44. Wiley