Applied Intelligence (2023) 53:12324–12339
https://doi.org/10.1007/s10489-022-04137-4

ERCP: speedup path planning through clustering and presearching

Kun He[1] · Xin-Zheng Niu[2] · Xue-Yang Min[3] · Fan Min[1]

Accepted: 31 August 2022 / Published online: 24 September 2022

Abstract

Path planning is an important task for robot motion, unmanned aerial vehicle obstacle avoidance, and smart cars. Sampling-based algorithms have achieved significant success in this task. The rapidly random-exploring tree (RRT) is one of the representative algorithms. However, it is exceedingly difficult to strike a balance between path quality and search efficiency. In this paper, we propose an enhanced RRT* with clustering and presearching (ERCP) algorithm to handle this issue. In the *clustering* phase, we construct an 8-degree undirected graph according to the neighborhood and obstacles. Then, we adopt the Markov clustering technique, which is appropriate for spatial data. The clustering process is efficient since the coarse-grained map considers only points at integer locations. In the *presearching* phase, we utilize the clustering results to abstract the intact *state space* as an undirected graph. We employ Dijkstra's algorithm to perform a presearch on the graph to determine the effective sampling area. In the *path planning* phase, we use RRT* to extend the space-filling tree within the effective sampling regions. Experiments were conducted on ten maps with different levels of obstacle complexity. The results reveal that ERCP can achieve a more convincing balance between path quality and search efficiency than the three state-of-the-art algorithms with little sacrifice.

Keywords Sampling-based path planning · Optimal path planning · Clustering

1 Introduction

Robotics-related technology contributes to the development of social productivity. It frees humans from repetitive and uncomplicated labor. Typical tasks include path planning [1] and manipulator control [2].

✉ Fan Min
minfan@swpu.edu.cn

Kun He
hk20201008@163.com

Xin-Zheng Niu
xinzhengniu@uestc.edu.cn

Xue-Yang Min
mitchellemin@163.com

[1] School of Computer Science, Southwest Petroleum University, Chengdu 610500, People's Republic of China

[2] School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 610500, People's Republic of China

[3] School of Sciences, Southwest Petroleum University, Chengdu 610500, People's Republic of China

Finding a collision-free path from *source* to *target* is the purpose of robot path planning. Many researchers have made excellent contributions to this task. Among them are grid-based methods, for example, Dijkstra's algorithm [3], A* [4], and D* [5]. Higher resolution results in a superior path have a longer or even unendurable runtime. State-space modeling algorithms include the cell decomposition method [6] and topology method [7]. After a well-modeled approach, promising paths can be obtained, but complex modeling environments may take longer. Bionic methods include the artificial potential field method (APF) [8], ant colony optimization algorithm (ACO) [9], and particle swarm optimization algorithm (PSO) [10]. Among them, individual algorithms have some drawbacks. Multiple bionic algorithms are combined to avoid this situation. For example, the genetic algorithm optimizes the suboptimal solution obtained by the ACO [11], and the sparrow search algorithm adds tent chaos mapping to optimize its initial population [12]. These methods are relatively simple to implement but suffer from the local minimum problem.

Currently, sampling-based algorithms [13] have become increasingly popular and accommodate theories such as probabilistic completeness. They explore the *state space* by probabilistically state sampling and constructing trees/graphs.

Examples include generalized space trees (ESTs) [14], probabilistic road maps (PRMs) [15], and random-exploring tree (RRT) [16]. Specifically, RRT effectively handles obstacles, differential motion constraints, and even high-dimensional spaces. However, the path generated by RRT is significantly different from the optimal solution. RRT* [13] introduces nearest neighbor search and rewiring to guarantee asymptotic optimality. This means that RRT* will obtain the optimal solution after several iterations.

Nevertheless, RRT* cannot strike an excellent balance between path quality and search efficiency. On the one hand, most of the initial solutions generated by RRT* are not excellent enough. On the other hand, RRT* iterates to the optimal solution, which is resource consumptive. The primary factor is that RRT* utilizes a random sampling strategy to blindly search the entire *state space*, resulting in a significant waste of resources.

To overcome the aforementioned problems, we propose the ERCP algorithm. Figure 1 illustrates a running example of the algorithm. In the *clustering* phase, we do not consider the *source* and the *target*. Figure 1a shows the original map. The red and blue discs indicate the *source* and the *target*. The black rectangle represents the obstacle region, and the rest of the space is an obstacle-free area. To facilitate clustering, we discretize the map. Then, it is abstracted into an 8-degree undirected graph. We adopt the Markov clustering algorithm [17] to determine the number of clusters automatically. The clustering results serve as the basis for dividing the *state space* into different parts, as shown in Fig. 1b.

In the *presearching* phase, we need to further reduce the processing difficulty. We abstract the clustering results as a weighted undirected graph. The geometric center of each cluster denotes itself and acts as a node of the graph. It is worth mentioning that the graph created in the *clustering* phase is not the same as that created in the *presearching* phase. The former is the result of the discretization of the continuous *state space*, while the latter represents the clustering results. We connect the adjacent geometric centers to form an edge of the graph since neighboring clustering regions are reachable to each other. The distance between nearby geometric centers is used as the weight of the graph. Then, we calculate which regions are most deserving to be sampled. In this step, the geometric center of each cluster represents all the elements in that region. The same applies to the *source* and the *target*. We adopt Dijkstra's algorithm to compute the shortest path from *target* to *source* on this graph. In Fig. 1c, the bright green color block indicates the effective sampling area.

In the *clustering* and *presearching* phases of ERCP, we transform the pathfinding problem into a graph shortest path problem. Thus, the redundant *state space* is quickly eliminated. In the *path planning* phase, the sampling area of ERCP is different from that of RRT*. We reconsider the obtained discrete effective sampling region as a continuous *state space*. We allow only ERCP to expand the space-filling tree in the effective sampling region. This mitigates the drawback that the optimal solution of the grid-based algorithm depends on the resolution. Figure 1d shows the initial solution. The bright green line indicates the branches of the space-filling tree. The red line indicates the initial solution of ERCP. We successfully excluded the redundant sampled regions during the sampling process.

We designed ten maps with different complexities and tested ERCP against three state-of-the-art algorithms. Two are sampling algorithms (RRT*, IRRT*), and one is an intelligent algorithm (P-ACO) that combines the advantages of traditional algorithms. The experimental results show that our algorithm can effectively balance the initial solution quality and efficiency.

Our main contributions include the following:

(1) A new path planning algorithm based on *clustering* and *presearching* is proposed.
(2) A generic planner is proposed, consisting of the first two phases of ERCP. It can predict an unknown environment's effective sampling area without prior knowledge, guiding different path planning algorithms.
(3) ERCP is applied to several case studies to demonstrate the effectiveness of the proposed method.

The rest of this paper is organized as follows. Section 2 demonstrates some of the concepts covered in this paper. Section 3 shows the detailed implementation of the ERCP and its analysis. Section 4 indicates the experimental results of the ERCP with a variety of popular algorithms. Section 5 presents some improved algorithms for RRT. Finally, Section 6 describes the advantages and disadvantages of ERCP and some future work.

2 Preliminaries

In this section, we briefly introduce the related concepts of path planning, the Markov clustering algorithm, and RRT*.

2.1 Path planning problem

The data model of the path planning problem can be represented by a 5-tuple:

$$(\chi, \chi_{\mathrm{obs}}, x_{\mathrm{sou}}, x_{\mathrm{tar}}, r), \tag{1}$$

where

1. $\chi \subset \mathbb{R}^n$ is the *state space*, usually $n \in \{2, 3\}$;
2. $\chi_{\mathrm{obs}} \subset \chi$ is the *obstacle state space*;
3. $x_{\mathrm{sou}} \in \chi \setminus \chi_{\mathrm{obs}}$ is the *source*;

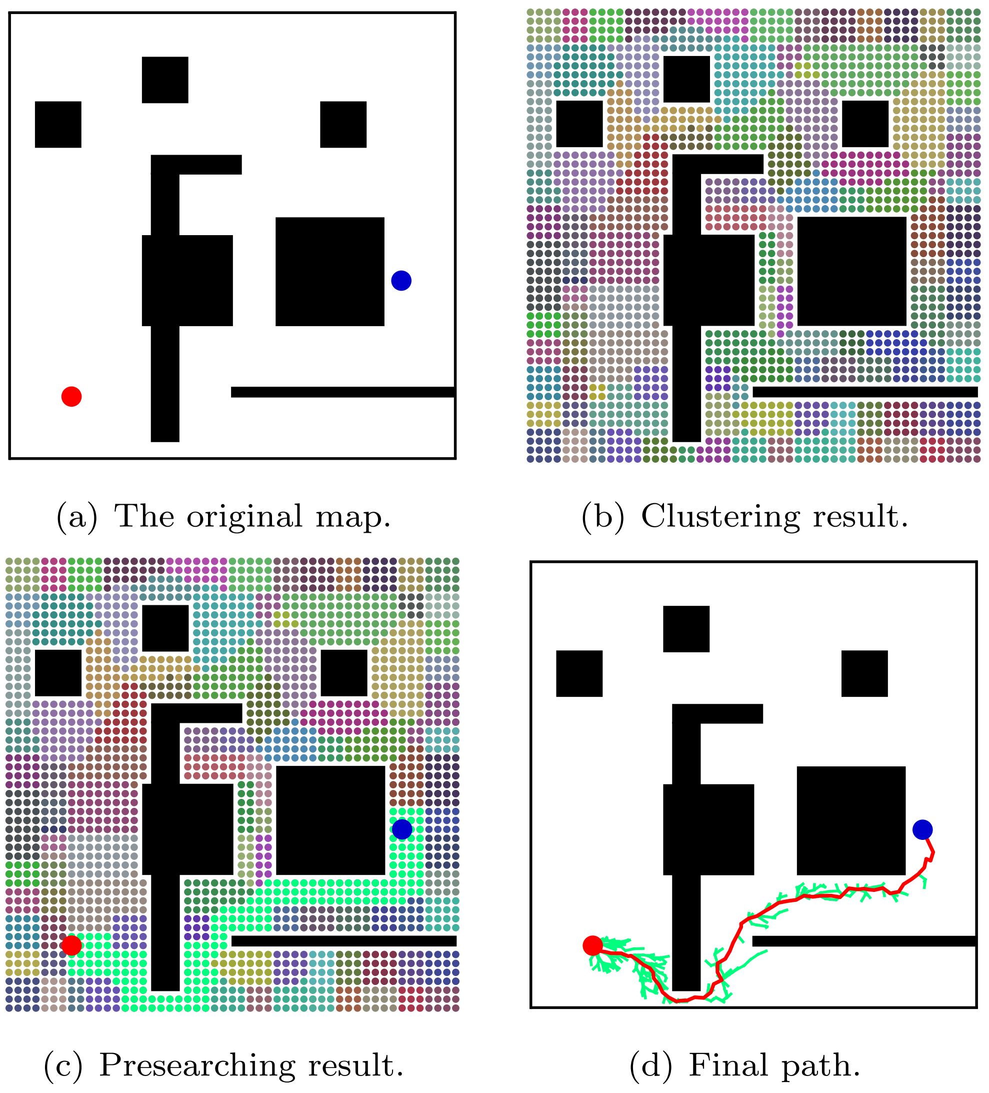


Fig. 1 Illustration of ERCP. (a) is the original map. The red and blue discs indicate the *source* and *target*, respectively. The black rectangles indicate the obstacles. (b) is the clustering result of the *clustering* phase. Different color blocks represent different clusters. (c) is the result of the *presearching* phase. The bright green blocks indicate the effective sampling regions. (d) is the result of the *path planning* phase. The red line signifies the initial solution. The green lines represent the branches of the space-filling tree of ERCP

4. $x_{\text{tar}} \in \boldsymbol{\chi} \setminus \boldsymbol{\chi}_{\text{obs}}$ is the *target*;
5. r is the radius of the *target*.

Moreover, $\boldsymbol{\chi}_{\text{fre}} = \boldsymbol{\chi} \setminus \boldsymbol{\chi}_{\text{obs}}$ is the *obstacle-free state space*. Let $\boldsymbol{\chi}_{\text{tar}} = \{x \in \boldsymbol{\chi}_{\text{fre}} | \|x - x_{\text{tar}}\| < r\}$ be the *target* space. The path planning problem is to find an *obstacle-free feasible continuous path* $\boldsymbol{\sigma} \subset \boldsymbol{\chi}_{\text{fre}}$ from x_{sou} to any $x \in \boldsymbol{\chi}_{\text{tar}}$. That is, we only need to reach the neighborhood of x_{tar}. The path is defined as a sequence

$$\boldsymbol{\sigma} = \sigma_0 \sigma_1 \ldots \sigma_m, \tag{2}$$

where $\sigma_0 = x_{\text{sou}}, \sigma_m \in \boldsymbol{\chi}_{\text{tar}}, \forall \lambda \in [0, 1], \lambda\sigma_i + (1-\lambda)\sigma_{i-1} \in \boldsymbol{\chi}_{\text{fre}} (i = 1, 2, \ldots, m)$, and the direct connection between σ_i and σ_{i-1} is obstacle-free. In other words, the path is essentially a polyline in the map.

The cost of the path is defined as the length of the polyline, i.e.,

$$c(\boldsymbol{\sigma}) = \sum_{i=1}^{m} \|\sigma_i - \sigma_{i-1}\|, \tag{3}$$

where $\|\cdot\|$ is the l_2 norm of a vector.

The path planning problem is to find an optimal path

$$\boldsymbol{\sigma}^* = \arg\min_{\boldsymbol{\sigma}} c(\boldsymbol{\sigma}). \tag{4}$$

2.2 Markov clustering algorithm

The Markov clustering algorithm is a grouping technique for a graph

$$G = (\mathbf{V}, \mathbf{E}), \tag{5}$$

where $\mathbf{V} = \{v_1, \cdots, v_n\}$ is the set of nodes and $\mathbf{E} \subseteq \mathbf{V} \times \mathbf{V}$ is the set of arcs.

The natural clustering in the graph generally has one property. There are more paths between two nodes in a cluster than between different clusters. This means that a random walk from any node on the graph is rarely transferred from one natural cluster to another.

The Markov clustering algorithm represents the graph as a nonnegative probability matrix

$$\mathbf{M} = (m_{ij})_{n\times n}, \tag{6}$$

where m_{ij} is the probability of v_i being shifted to v_j. $\forall j \in [1, n]$,

$$\sum_{i=1}^{n} m_{ij} = 1. \tag{7}$$

The Markov clustering algorithm alternately performs *Expansion* and *Inflation* on the probability matrix $\mathbf{M}$. The *Expansion* operation simulates a random walk in the graph by performing a matrix product on the probability matrix $\mathbf{M}$. We provide two numbers $p \in \mathbb{N}^+$ and $l \in \mathbb{N}^+$. The *Expansion* operation is described as

$$\mathbf{M}^p = \left(m_{ij}^{(p)}\right)_{n\times n} = \prod_{k=1}^{p} \mathbf{M}. \tag{8}$$

The *Inflation* operation operates on the elements of each column individually. It can be divided into two steps. First, we take the l power for each element of column j of the matrix $\mathbf{M}$, denoted as

$$\mathbf{M}^{\langle l\rangle} = \left(m_{ij}^{l}\right)_{n\times n}. \tag{9}$$

Second, we scale each column to obtain $\Gamma^{\langle l\rangle} = \left(\gamma_{ij}^{\langle l\rangle}\right)_{n\times n}$, where

$$\gamma_{ij}^{\langle l\rangle} = \frac{m_{ij}^{l}}{\sum_{k=1}^{n} m_{kj}^{l}}. \tag{10}$$

The *Expansion* operation corresponds to multiple random walks for each node in the graph. It associates the new probabilities with all node pairs. Longer paths are more common within a cluster than between clusters. Nodes within the same cluster have a relatively greater probability of reaching each other. The *Inflation* operation increases the probability of random walk within a cluster. This is achieved without any prior knowledge about the cluster structure. It is just a result of the presence of different clusters in the natural graph.

Eventually, iterating *Expansion* and *Inflation* leads to the graph being divided into different clusters. There are no longer any paths between these clusters. Increasing p and l increases the compactness of the clusters, but a slight change in p and l does not lead to a drastic change in the results.

2.3 Dijkstra's algorithm

Dijkstra's algorithm is a classical method for computing the shortest path in a graph; this graph is defined as

$$G = (\mathbf{V}, w), \tag{11}$$

where $\mathbf{V} = \{v_1, \cdots, v_n\}$ is the set of nodes and $w(v_i, v_j)$ is the weight of arc (v_i, v_j).

Without losing generality, we assume that v_1 is the source. The flow of Dijkstra's algorithm is as follows:

```
1. $\mathbf{W}' = (w'_{ij})_{n\times n} = -\mathbf{1}$; //All elements are -1.
2. $\mathbf{V}' = \mathbf{V} \setminus \{v_1\}$; //Unvisited nodes.
3. $\mathbf{V}'' = \{v_1\}$; //Visited nodes.
4. while $\mathbf{V}' \neq \emptyset$:
5. $(v^*, v_*) = \arg\min_{(v_i,v_j)\in\mathbf{V}''\times\mathbf{V}'} w(v_i, v_j)$;
6. $\mathbf{V}' = \mathbf{V}' \setminus \{v_*\}$;
7. $\mathbf{V}'' = \mathbf{V}'' \cup \{v_*\}$;
8. $w'(v^*, v_*) = w(v^*, v_*)$.
```

2.4 RRT*

Among the sampling-based algorithms, RRT* is one of the representative algorithms that are widely used. RRT* explores the *state space* by expanding space-filling trees

$$T = (\mathbf{N}, r, p) \tag{12}$$

where

1) $\mathbf{N}$ is the set of all nodes in the tree;
2) $r \in \mathbf{N}$ is the root node;
3) $p : \mathbf{N} \to \mathbf{N} \cup \{\phi\}$ is the parent mapping satisfying
    a) $p(r) = \phi$;
    b) $\forall n \in \mathbf{N}, \exists!\, i \geq 0$, s.t. $p^{(i)}(n) = r$;
    c) ϕ represents the empty node.

RRT* uses *source* as the root node r of T. It randomly draws x_{ran} from $\boldsymbol{\chi}_{\text{fre}}$, and x_{ran} tries to find the nearest node x_{nst} in T. If x_{ran} and x_{nst} can be successfully connected, then by steering at a certain angle, we can obtain x_{new} and add it to $\mathbf{N}$.

RRT* adds two operations, *nearest neighbor search* and *rewiring*, to ensure progressive optimization. In the *nearest neighbor search* process, it searches for nodes around x_{new} as X_{nea}. RRT* treats each node of X_{nea} as a temporary parent node of x_{new}. It will select the node with the minimal path cost as the parent node of x_{new}.

In the *rewiring* process, for each single node of X_{nea}, denoted as x_{nea}, x_{new} is taken as the temporary parent node of x_{nea}. It checks whether the path cost from r to x_{nea} is lower than the previous path. If the condition is satisfied, set $p(x_{\text{nea}}) = x_{\text{new}}$.

If x_{new} is in $\boldsymbol{\chi}_{\text{tar}}$ or the maximum number of iterations is reached, RRT* ends the search. The found path is then returned from T if it exists. When the number of iterations tends to infinity, RRT* must be able to give an optimal path. This guarantees probabilistic completeness and asymptotic optimality. However, it also requires a dramatic increase in time and memory. However, RRT* searches the entire *state space*. This leads to a relatively blind search process. Searching the redundant *state space* will consume considerable time and resources.

3 The proposed algorithm

In this section, we first describe the implementation of ERCP. Then, we prove the probabilistic completeness of ERCP.

3.1 Algorithm flow

Figure 2 shows the algorithmic flow of ERCP. The image in the top row shows the output of each stage, and the bottom row describes each stage.

ERCP is divided into three phases in total: the *clustering* phase, *presearching* phase, and *path planning* phase. For each algorithm phase, the previous phase's output is the next phase's input. The *clustering* phase discretizes the input *state space*, whose primary purpose is to divide the *state space* into different parts and provide credentials for the *presearching* phase. The *presearching* phase finds the effective sampling region in the clustered *state space*. The *path planning* phase performs path planning within the best sampling region obtained from the *presearching* phase. Finally, the *path planning* phase outputs the viable paths from the *source* to the *target*.

The *clustering* phase applies the Markov clustering algorithm to partition the *state space* into different sections. This algorithm is a fast and scalable unsupervised clustering algorithm. It divides the graph into multiple clusters based on the simulation of random flows in the graph. The advantage is that it uses few parameters, does not require a priori knowledge, and does not require specifying the number of clusters. Dijkstra's algorithm is applied in the *presearching* phase and is one of the classical and excellent algorithms for finding the shortest path. Its drawback is that it cannot find the shortest path if negatively weighted edges are in the graph. However, this problem does not exist in this paper.

3.2 Algorithm description

Algorithm 1 describes the *clustering* phase and *presearching* phase of ERCP. Algorithm 2 depicts the *path planning* phase of ERCP. The inputs include the 5-tuple specified in (1), and two numbers p and l are used for Markov clustering. The output is the optimal path found by the ERCP from x_{sou} to χ_{tar}.

We automatically divide the *state space* into different parts by an unsupervised clustering algorithm. In algorithm 1, Line 1 calculates the accessibility space χ_{fre}. Line 2 discretizes χ_{fre} by $Discretize$ to obtain $\mathbf{V}_1$. For example, in a two-dimensional map, the discretization operation is to keep only those parts of the continuous χ_{fre} for which the coordinate points are integers and store them to $\mathbf{V}_1$. In the third line, the set of edges $\mathbf{E}_1$ of G_1 is initialized. Each node needs to add edges that reach itself, which is determined by the nature of the Markov clustering algorithm itself. In Lines 4 to 5, the number of nodes n of $\mathbf{V}_1$ is computed, and subsequently, the adjacency matrix $(m_{ij})_{n\times n}$ of G_1 is initialized. In Lines 6 to 10, each node v in $\mathbf{V}_1$ can reach its surrounding neighbor nodes in eight directions if neighbors exist. Mutually reachable pairs of nodes are added to $\mathbf{E}_1$, and the corresponding values in the adjacency matrix $(m_{ij})_{n\times n}$ are updated. In the eleventh line of the algorithm, G_1 consists of $\mathbf{V}_1, \mathbf{E}_1$.

In Lines 12 to 13, each column of the adjacency matrix $(m_{ij})_{n\times n}$ is normalized. Then, each column of the adjacency matrix sums to 1. The normalized matrix is the

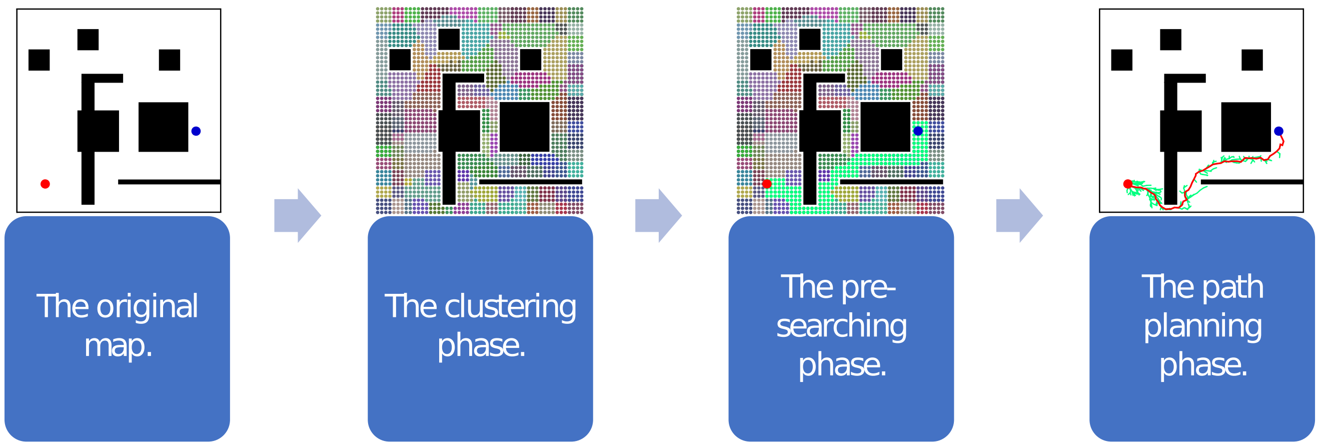


Fig. 2 The algorithm flow of ERCP. The top image shows the output results of each stage, and the bottom shows the description of each phase

probability matrix, which stores the probability of each node reaching the other nodes. In Lines 14 to 19, the procedure of the Markov clustering algorithm is shown. The detailed implementation of *Expansion* and *Inflation* is in Section 2.2. In Line 20, the clustering results are stored in $\mathbb{C}$ via *GetClusters*. The matrix calculated by the Markov clustering algorithm stores the probability of each node reaching the other nodes. Nodes that can reach each other are grouped into a cluster.

We eliminate the redundant *state space* by evaluating the shortest path. The clustering results are first streamlined in the *presearching* phase. The geometric centroid of each cluster represents itself and acts as a node of $\mathbf{V}_2$. In Line 21, the geometric centroids of each cluster are computed. For example, in the two-dimensional *state space*, only cluster $\mathbf{C} \in \mathbb{C}$ contains a total of s elements, and the ith element is denoted as (x_i, y_i). The geometric center of cluster $\mathbf{C}$ is denoted as $(x_{\text{cen}}, y_{\text{cen}})$ and is calculated as

$$(x_{\text{cen}}, y_{\text{cen}}) = \left(\frac{\sum_{i=1}^{s} x_i}{s}, \frac{\sum_{i=1}^{s} y_i}{s} \right). \tag{13}$$

In Lines 22 to 23, the weight matrix $\mathbf{W}$ of G_2 is initialized. In Lines 24 to 27, for v, the geometric centers of its neighboring clusters are obtained by the function *GetAdjacentClusters*. For $v, v' \in \mathbf{V}_2$, if they are adjacent to each other, the distance between v and v' is used as the weight of (v, v'). In Line 28, it is declared that G_2 consists of $(\mathbf{V}_2, \mathbf{W})$. In Lines 29 to 30, the geometric center of a cluster can represent all its elements. x_{sou} is represented by the geometric center of the cluster it is in, and the same is true for x_{tar}. In G_2, the shortest path from *source* to *target* and the corresponding cluster centroid $\mathbf{V}''$ is determined by *FindShortestPath*. It employs Dijkstra's algorithm; refer to Section 2.3 for detailed implementation. Each clustering centroid uniquely corresponds to a clustering region. $\mathbf{V}''$ is converted to the corresponding clustering area by *ConvertToArea*. It is an effective sampling area.

Algorithm 2 documents the detailed implementation of the *path planning* phase. It adopts RRT* for sampling. However, the sampling strategy of ERCP is different from RRT*. ERCP samples in χ_{opt}, while RRT* samples in the whole *state space*. In Line 1, initialize $T = (\mathbf{N}, r, p)$, where $\mathbf{N}$ records the information of all nodes in T, r denotes the root node of tree T, and p records the information of each node's father. χ_{tar} denotes the region of radius r centered at x_{tar}. That is, it is enough to reach the vicinity of x_{tar}. In Line 4, x_{ran} is determined by random sampling from χ_{opt} using *SampleFrom*. The subsequent process is identical to RRT*, and its detailed implementation is shown in Section 2.4. The state x_{nst} in the tree T that is closest to x_{ran} is found using *Nearest*. Then, *Steer* is used to find the new state x_{new}. Go further in the direction of connecting x_{ran} to x_{nst}. If the connection between x_{nst} and x_{new} is collision-free, x_{new} will be added to the tree. *NearestNeighborSearch* and *Rewiring* will be executed to optimize the path. The space-filling tree T is expanded in χ_{opt} until a feasible path is found or the number of iterations reaches a maximum.

```
Input: $\chi, \chi_{\text{obs}}, x_{\text{sou}}, x_{\text{tar}}, p, l$
Output: $\chi_{\text{opt}}$
1: $\chi_{\text{fre}} \leftarrow \chi \setminus \chi_{\text{obs}}$;            ▷ The clustering phase.
2: $\mathbf{V}_1 \leftarrow Discretize(\chi_{\text{fre}})$;
3: $\mathbf{E}_1 \leftarrow \{(v, v) \mid v \in \mathbf{V}_1\}$;          ▷ $G_1 = (\mathbf{V}_1, \mathbf{E}_1)$;
4: $n \leftarrow |\mathbf{V}_1|$;
5: $\mathbf{M} = (m_{ij})_{n\times n} \leftarrow \mathbf{0}_{n\times n}$;     ▷ The adjacency matrix of $G_1$.
6: for $v \in \mathbf{V}_1$ do
7:     $\mathbf{V}' \leftarrow GetNeighbors(v)$;       ▷ $G_1$ is an 8-degree undirected graph.
8:     for $v' \in \mathbf{V}'$ do
9:         $\mathbf{E}_1 \leftarrow \mathbf{E}_1 \cup (v, v')$;
10:        $m_{vv'} \leftarrow 1$;
11: $G_1 = (\mathbf{V}_1, \mathbf{E}_1)$;
12: for $i, j \in \{1, 2, \dots, n\}$ do
13:     $m_{ij} = m_{ij}/m_{*j}$;        ▷ Normalization of columns.
14: while true do
15:     $M' \leftarrow Expansion(M, p)$;  ▷ Refer to Section 2.2.
16:     $M' \leftarrow Inflation(M', l)$;   ▷ Refer to Section 2.2.
17:     if $M' = M$ then
18:         $break$;
19:     $M \leftarrow M'$;
20: $\mathbb{C} \leftarrow GetClusters(M)$;
21: $\mathbf{V}_2 \leftarrow \{\sum \mathbf{C}/|\mathbf{C}| \mid \mathbf{C} \in \mathbb{C}\}$;  ▷ The presearching phase.
22: $n' \leftarrow |\mathbf{V}_2|$;
23: $\mathbf{W} = (w_{ij})_{n'\times n'} \leftarrow \mathbf{0}_{n\times n}$;    ▷ $\mathbf{W}$ is the weight matrix.
24: for $v \in \mathbf{V}_2$ do
25:     $\mathbf{V}' \leftarrow GetAdjacentClusters(v)$;
26:     for $v' \in \mathbf{V}'$ do
27:         $w_{vv'} \leftarrow \|v - v'\|$;
28: $G_2 = (V_2, W)$;
29: $v_{\text{sou}} \leftarrow \{\arg\min_v \|x_{\text{sou}} - v\| \mid v \in V_2\}$;
30: $v_{\text{tar}} \leftarrow \{\arg\min_v \|x_{\text{tar}} - v\| \mid v \in V_2\}$;
31: $\mathbf{V}'' \leftarrow Dijkstra(G_2, v_{\text{sou}}, v_{\text{tar}})$;           ▷ Refer to Section 2.3.
32: $\chi_{\text{opt}} \leftarrow ConvertToArea(\mathbf{V}'', \mathbb{C})$;          ▷ $v \in \mathbf{V}''$ corresponds to $\mathbf{C} \in \mathbb{C}$.
33: Return $\chi_{\text{opt}}$;
```

Algorithm 1 Enhanced RRT* with Clustering and Presearching (ERCP).

```
Input: $\boldsymbol{\chi}$, $\boldsymbol{\chi}_{\text{obs}}$, $x_{\text{sou}}$, $x_{\text{tar}}$, $\boldsymbol{\chi}_{\text{opt}}$, $r$
Output: $T$
1: $T = (\mathbf{N}, r, p) \leftarrow (\emptyset, x_{\text{sou}}, \emptyset)$;    ▷ The path planning phase.
2: $\boldsymbol{\chi}_{\text{tar}} \leftarrow \{x \in \boldsymbol{\chi}_{\text{fre}} | \|x - x_{\text{tar}}\| < r\}$;
3: for $i \in \{1, 2, \ldots, k\}$ do
4:     $x_{\text{ran}} \leftarrow SampleFrom(\boldsymbol{\chi}_{\text{opt}})$;    ▷ Different from RRT*.
5:     $x_{\text{nst}} \leftarrow Nearest(x_{\text{ran}})$;
6:     $x_{\text{new}} \leftarrow Steer(x_{\text{nst}}, x_{\text{ran}})$;
7:     if $obstacleFree(x_{\text{nst}}, x_{\text{new}})$ then
8:         $\mathbf{N} \leftarrow \mathbf{N} \cup x_{\text{new}}$;
9:         $p(x_{\text{new}}) = x_{\text{nst}}$;
10:        $X_{\text{nea}} \leftarrow NearestNeighborSearch(T, r)$;
11:        $T \leftarrow Rewiring(X_{\text{nea}}, x_{\text{new}})$;
12:        if $x_{\text{sou}} \in \boldsymbol{\chi}_{\text{tar}}$ then
13:            Return $T$;
14: Return $failure$;
```

Algorithm 2 Enhanced RRT* with Clustering and Presearching (ERCP).

3.3 Algorithm analysis

ERCP has the property of probabilistic completeness, i.e.,

$$\lim_{n\to\infty} \mathbb{P}\{(T \cap \boldsymbol{\chi}_{\text{tar}}) \neq \emptyset\} = 1. \tag{14}$$

We divide the obstacle-free region in the *state space* into different parts by clustering. Then, the clustering results are abstracted into an undirected graph. The *source* is located in one of the clusters. We use the cluster geometric center to represent itself. The same is true for the *target*. In the extreme case, one cluster can be obtained from the entire *state space*. The effective sampling area is the whole *state space*. The optimal path obtained by ERCP is the same as RRT*. In the more general case, the *state space* will be clustered into several clusters. Then, the clusters are abstracted into an undirected weighted graph. Both the *source* and the *target* are located at a node in the graph. The shortest path is calculated by Dijkstra's algorithm. Then, it is transformed into the corresponding region, which is the effective sampling region $\boldsymbol{\chi}_{\text{opt}}$. The adjacent nodes in the graph correspond to the adjacent clustering regions. That is, $\boldsymbol{\chi}_{\text{opt}}$ is a continuous area. Both the *source* and the *target* are within $\boldsymbol{\chi}_{\text{opt}}$. When the number of iterations tends to infinity, the space-filling tree will be filled with $\boldsymbol{\chi}_{\text{opt}}$. There must be a feasible path σ from x_{sou} to $\boldsymbol{\chi}_{\text{tar}}$. As $\boldsymbol{\chi}_{\text{opt}} \in \boldsymbol{\chi}_{\text{fre}}$, ERCP does not need to explore the entire *state space*. It can also prove the effectiveness of ERCP.

4 Experiments

4.1 Experimental configuration

To evaluate the performance of ERCP, we compare it with three state-of-the-art algorithms (RRT*, IRRT*, P-ACO) in ten different complex environments. We employed Windows 10 as the running platform, Python 3.8 as the programming language, and an AMD Ryzen-7 4800H as the central processor, and the memory size was 16 GB. The parameters in the simulation experiments are set as follows (unit: m): map size 50×50. In particular, the sizes of Map-9 and Map-10 are 20×20 and 70×70, respectively. The step size of the space-filling tree is 1, the radius of x_{tar} is 3, and the steering angle is 20 degrees. For Markov clustering, both p and l are 2. The parameter settings for ERCP, RRT*, and IRRT* are the same.

To evaluate the actual performance of ERCP, numerical experiments with three metrics were undertaken in ten environments of different complexities. The major metrics include the quality of the initial solution, the corresponding time, and the number of nodes. It was also compared with two other state-of-the-art path planning algorithms (RRT* and IRRT*) to verify the effectiveness of ERCP.

ACO is a heuristic global optimization algorithm with distributed computation, positive information feedback, and heuristic search. Therefore, many scholars have used the ACO algorithm for robot path planning and have achieved good results. The ACO algorithm for robot path planning consists of three parts: ant colony initialization, solution construction, and pheromone update. However, it suffers from slow convergence, quickly falling into a local optimum, and premature convergence. P-ACO adopts the potential field to guide ACO to alleviate the above problems, i.e., the current path pheromone spreads in the direction of the potential field during the ant search. As a result, the ants tend to search for more adaptive subspaces. This is similar to ERCP and is one of the reasons for choosing it as the comparison algorithm.

To thoroughly verify the robustness and superiority of the model, we chose P-ACO, which combines the advantages of traditional and intelligent algorithms, for comparison with ERCP. P-ACO uses a potential field to guide the ant colony for pheromone diffusion to alleviate the problems of the ACO algorithm.

Regarding the parameter settings, considering the size of the map, the number of ants is 100, the volatility of pheromones is 0.5, and the number of iterations is 40 to ensure that the optimal solution is found. Finally, different combinations of the radius of x_{tar} and step size were tested to illustrate the robustness of the model.

4.2 Maps and processing results

Figure 3 illustrates the environment used for the experiment. The environments are named Map-1 to Map-10 according to their obstacle complexity. Map-1 (Fig. 3a), Map-2 (Fig. 3c), and Map-3 (Fig. 3e) represent cluttered environments with many rectangular obstacles. This leads to some sampling points on the obstacles, resulting in multiple useless samples. Map-4 (Fig. 3g) indicates a cluttered environment and fewer L-obstacles. Map-5 (Fig. 3i), Map-6 (Fig. 3k), and Map-7 (Fig. 3m) represent the more common L-shaped obstacles in practical applications. Therefore, the sampling-based algorithm needs to search the entire *state space* to reach the *target*. Map-8 (Fig. 3o) shows a typical narrow passage environment, where the *source* and the *target* are distributed on both sides of the narrow passage. Map-9 (Fig. 3q) is a smaller-sized map with a size of 20 × 20, where several different obstacles are distributed. Map-10 (Fig. 3s) is a larger sized map with a size of 70 × 70. The purpose of Map-9 and Map-10 is to test the effect of different sizes of maps on the algorithm. It is well known that the sampling strategies of sampling-based algorithms are mostly random probability sampling. The probability of sampling in a narrow channel *state space* is lower than that in a wide *state space*. This leads to the difficulty of reaching the *target* from the *source*.

In Fig. 3, for each environment, the results of the *presearching* phase are available on its right side. Different color blocks indicate different clustering regions, and the bright green color blocks represent effective sampling regions. It is noteworthy that almost all effective sampling areas successfully connect the *source* and the *target* in the closest way.

4.3 Comparison with RRT*

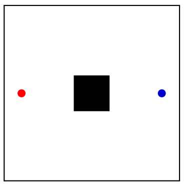
We compare the performance of ERCP with RRT* at a single iteration. Figure 4 compares the metrics associated with ERCP and RRT* for obtaining initial solutions in Map-1 to Map-10 and with labels at the bottom of each figure. The metrics evaluated here include the initial solution cost, the time consumed to obtain the initial solution, and the number of nodes needed.

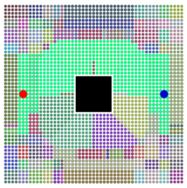
Map-1, Map-2, and Map-3 are environments with rectangular obstacles of different complexity, and the results of their presearching are Fig. 3b, d, and f, respectively. Figure 4a, c, and e are the test results of ERCP on Map-1, Map-2, and Map-3, respectively. Correspondingly, the results of RRT* are Fig. 4b, d, and f. In terms of path cost, the path costs of ERCP are 47.434, 50.137, and 68.039, respectively. The path costs of RRT* are 55.425, 62.466, and 82.768, respectively. This can be seen in an environment with progressively increasing complexity. RRT* grows space-filling trees that spread over the entire space and generates paths with more bends than ERCP. ERCP searches only a tiny fraction of the space, and the path quality obtains a complete victory. In terms of time, the time consumption of ERCP is 0.049 s, 0.122 s, and 4.311 s, respectively. RRT* is 0.337 s, 0.296 s, and 5.157 s, respectively. Regarding the number of nodes, the number of nodes for ERCP is 139, 156, and 1072, respectively. The number of nodes for RRT* is 600, 419, and 1634, respectively. ERCP is still ahead of RRT* in terms of time and number of nodes. This means that ERCP uses fewer resources and has better path quality.

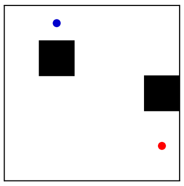
Map-4 is an L-shaped rectangular obstacle environment with Fig. 3h as the presearch result. Figure 4g and h are the test results for ERCP and RRT*, respectively. The path cost of ERCP is 54.123, the time taken is 2.529 s, and the number of nodes is 854. The path cost of RRT* is 93.121, the time taken is 3.295 s, and the number of nodes is 1720. Map-4 deserves special attention to highlight the particular advantages of ERCP. After clustering and presearching, ERCP yields more worthwhile regions to sample. This reduces the invalid sampling, which in turn makes the generation path of ERCP better.

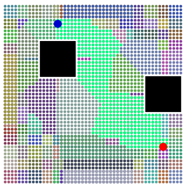
Map-5, Map-6, Map-7, and Map-8 are environments with many narrow channels, and their presearch results are shown in Fig. 3j, l, n, and p. Figure 4i, k, m, and o show the performance of ERCP on those maps. Figure 4j, l, n, and p illustrate the corresponding performance of RRT*. The path costs of ERCP are 56.408, 75.649, 82.419, and 109.373, respectively. The RRT* values are 77.591, 107.723, 110.336, and 128.149, respectively. In terms of time consumption, ERCP was 2.744 s, 8.207 s, 7.751 s, and 19.772 s, respectively. The RRT* values are 9.547 s, 12.306 s, 13.833 s, and 2.407 s, respectively. ERCP has 558, 1909, 1656, and 5488 nodes, respectively. RRT* has 3305, 4520, 3761, and 3003 nodes, respectively. One of the shortcomings of RRT* is that the probability of sampling into narrow channels is negligible. As seen from the figure, this phenomenon did not interfere with the effect of ERCP. The main reason is that the clustering and presearch do not distinguish between narrow channels and other environments.

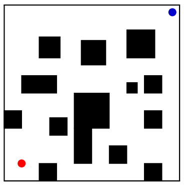
Map-9 and Map-10 represent smaller and larger maps, respectively, with presearch Fig. 3r and t. The relevant results of ERCP are presented in Fig. 4q and s. The RRT* data are shown in Fig. 4r and t. In the path cost, ERCP is 26.764 and 66.924, respectively. RRT* is 30.229 and 71.687, respectively. In the case of resource consumption, ERCP takes 2.018 s and 0.394 s, and the number of nodes is 560 and 425, respectively. The time consumption of RRT* is 0.424 s and 0.711 s, and the number of nodes is 486 and

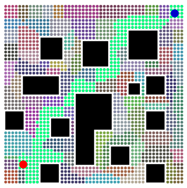

(a) Map-1.

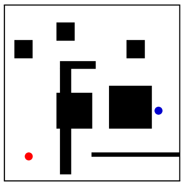

(b) Map-1, result of the *presearching* phase.

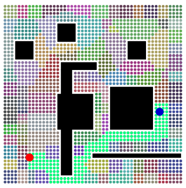

(c) Map-2.

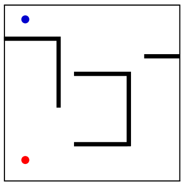

(d) Map-2, result of the *presearching* phase.

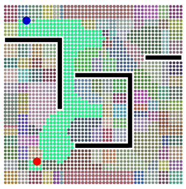

(e) Map-3.

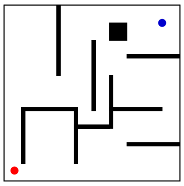

(f) Map-3, result of the *presearching* phase.

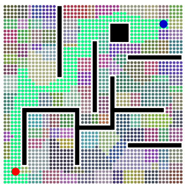

(g) Map-4.

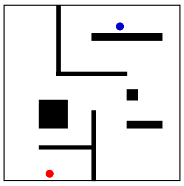

(h) Map-4, result of the *presearching* phase.

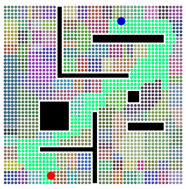

(i) Map-5.

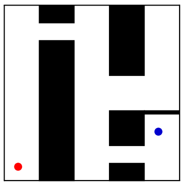

(j) Map-5, result of the *presearching* phase.

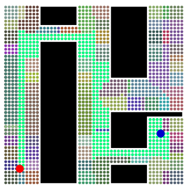

(k) Map-6.

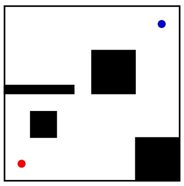

(l) Map-6, result of the *presearching* phase.

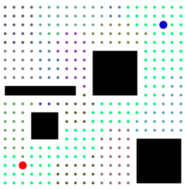

(m) Map-7.

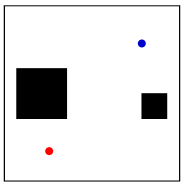

(n) Map-7, result of the *presearching* phase.


(o) Map-8.


(p) Map-8, result of the *presearching* phase.


(q) Map-9.


(r) Map-9, result of the *presearching* phase.


(s) Map-10.

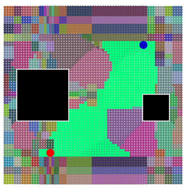
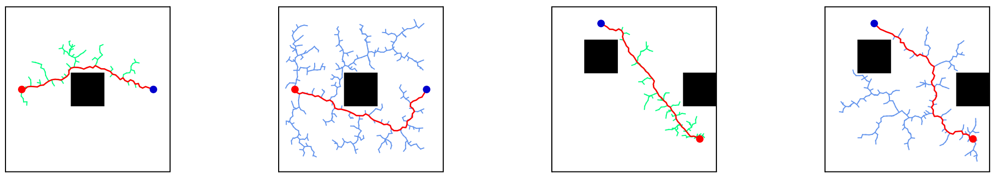

(t) Map-10, result of the *presearching* phase.

Fig. 3 Illustration of the different maps in the experiment and their clustering results. In each row, the first and third images represent the original maps. The second and fourth panels indicate the clustering results of the map. The red and blue discs denote the *source* and *target*. The black rectangles represent obstacles. In the display of clustering results, different color blocks indicate different clusters, and bright green color blocks indicate effective sampling areas

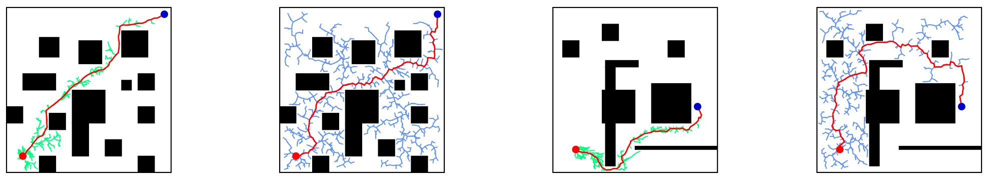


(a) Map-1, ERCP, Cost = 47.434, Time = 0.049 s, Node = 139.

(b) Map-1, RRT*, Cost = 55.425, Time = 0.337 s, Node = 600.

(c) Map-2, ERCP, Cost = 50.137, Time = 0.122 s, Node = 156.

(d) Map-2, RRT*, Cost = 62.466, Time = 0.296 s, Node = 419.

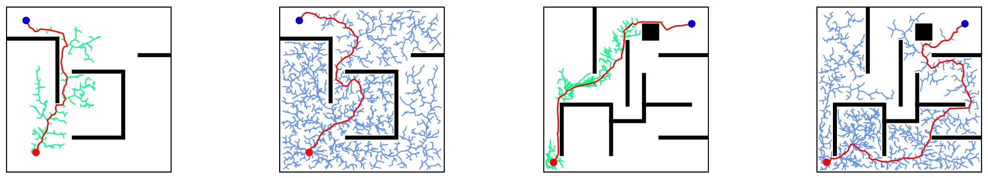


(e) Map-3, ERCP, Cost = 68.039, Time = 4.311 s, Node = 1072.

(f) Map-3, RRT*, Cost = 82.768, Time = 5.157 s, Node = 1634.

(g) Map-4, ERCP, Cost = 54.123, Time = 2.529 s, Node = 854.

(h) Map-4, RRT*, Cost = 93.121, Time = 3.295 s, Node = 1720.

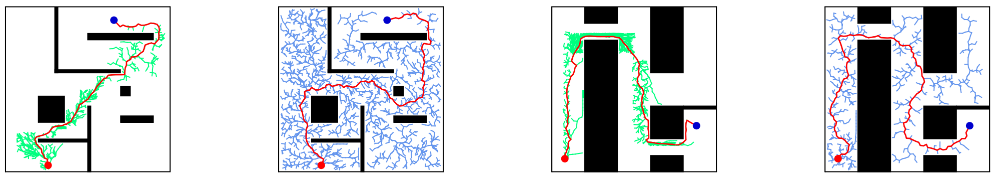


(i) Map-5, ERCP, Cost = 56.408, time = 2.744 s, Node = 558.

(j) Map-5, RRT*, Cost = 77.591, Time = 9.547 s, Node = 3305.

(k) Map-6, ERCP, Cost = 75.649, Time = 8.207 s, Node=1909.

(l) Map-6, RRT*, Cost = 107.723, Time = 12.306 s, Node = 4520.

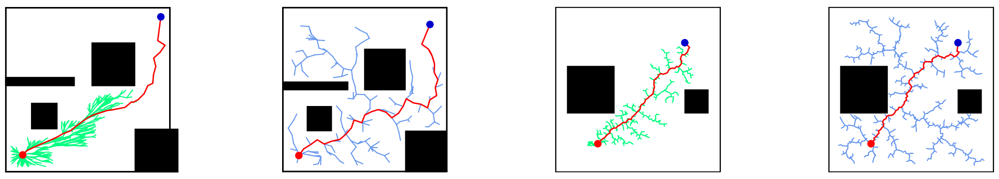


(m) Map-7, ERCP, Cost = 82.419, Time = 7.751 s, Node = 1656.

(n) Map-7, RRT*, Cost = 110.336, Time = 13.833 s, Node = 3761.

(o) Map-8, ERCP, Cost = 109.373, Time = 19.772 s, Node = 5488.

(p) Map-8, RRT*, Cost = 128.149, Time = 2.407 s, Node = 3003.



(q) Map-9, ERCP, Cost = 26.764, Time = 2.018 s, Node = 560.

(r) Map-9, RRT*, Cost = 30.229, Time = 0.424 s, Node = 486.

(s) Map-10, ERCP, Cost = 66.924, Time = 0.394 s, Node = 425.

(t) Map-10, RRT*, Cost = 71.687, Time = 0.711 s, Node = 798.

Fig. 4 Comparison of initial solutions of ERCP and RRT* in different maps. The blue line indicates the branches of RRT*. The green line represents the branches of ERCP. The red line denotes the initial solution. Cost means the length of the initial solution. Time signifies the time taken to obtain the initial solution. Node stands for the number of space-filling tree nodes

798. After data analysis, smaller and larger environments do not make ERCP less effective. This proves that ERCP has excellent generalization ability.

However, in Map-8 and Map-9, ERCP required more time and resources. The same sampling strategy as RRT* is used in the path planning phase of ERCP, and randomness leads to poor performance in a single test. Regarding path cost, ERCP is far ahead of RRT* thanks to clustering and presearch. ERCP can achieve excellent initial solutions in environments with sparse obstacles with less resource consumption. In complex environments, ERCP still shows extraordinary performance. RRT* consumes more resources in the test environment and only obtains suboptimal solutions. ERCP can accurately find a good sampling area from source to target and then explore it. More importantly, the quality of the initial solutions obtained by ERCP is also surprisingly improved after a small amount of preprocessing.

4.4 Extensive comparison experiments

We compare ERCP with IRRT*, RRT*, and P-ACO for 100 iterations in all ten environments and count the performance indicators of the four algorithms. Tables 1, 2, and 3 record the three primary metrics for ERCP, IRRT*, and RRT* in all experimental settings. In addition, P-ACO is added to compare the cost and time to acquire the generated initial solutions. It includes the initial solution quality, the time required to obtain the initial solution quality, and the number of iterations to obtain the initial solution.

Table 1 records the initial solution cost obtained by the four algorithms. IRRT* takes an elliptical heuristic search, which will make it closer to the optimal path for searching. It is noteworthy that ERCP achieves seven wins in ten environments. Furthermore, the remaining three environments include Map-1, Map-3, and Map-8. The initial solution quality of ERCP is far outperformed by RRT* and is not much different from the results of IRRT*. The initial solution variance found by ERCP fluctuates less. This means that the initial solution obtained by ERCP is more stable than others. RRT*, IRRT*, and ERCP use the same sampling and rewiring strategy to find the initial solution. However, the difference is that IRRT* implements the allowed ellipsoid heuristic, RRT* searches the entire state space, and ERCP searches within the effective sampling region obtained by preprocessing. As a result, the initial solution quality obtained by RRT* is lower, while the path quality of ERCP and IRRT* is superior. It is important to note that ERCP outperforms IRRT* in seven out of ten cases in terms of initial solution quality.

Regarding the performance of the different algorithms, ERCP outperforms P-ACO in terms of path quality in all ten environments. The main reason is that even with potential field guidance, P-ACO still requires a random search to reach the best global solution. ERCP has less invalid exploration with the help of effectively sampled regions. It allows ERCP to achieve better path quality compared to P-ACO.

Table 2 records the time taken by the four algorithms to obtain the initial solution. In terms of time, ERCP achieves seven wins. RRT* performs better the remaining three times. Table 3 keeps track of the number of iterations used by the three algorithms to obtain the initial solution. In terms of the number of iterations, ERCP achieves eight wins. The remaining two were obtained by RRT* on Map-6 and IRRT* on Map-8. The time to obtain the initial solution indicates the efficiency of the desired algorithm, and the number of nodes reflects the algorithm's performance in terms of memory. The efficiency and memory performance reflect the availability of the algorithm to some extent. Note that ERCP is successful in most scenarios and provides additional help in a few less than excellent scenarios. In Map-3, for example, ERCP does not perform particularly well in terms of path cost and time to obtain the initial solution. In terms of path cost, RRT*, IRRT*, and ERCP are 85.210 ± 7.377, 68.321 ± 3.615, and 70.537 ± 1.691, respectively. In terms of the time to obtain the initial

Table 1 Cost of the initial solution

	RRT*	IRRT*	ERCP	P-ACO
Map-1	57.901 ± 4.903	**45.968** ± 3.103	51.566 ± 3.305	58.142 ± 5.679
Map-2	65.358 ± 8.129	50.248 ± 4.492	**49.123** ± 1.478	62.524 ± 5.028
Map-3	85.210 ± 7.377	**68.321** ± 3.615	70.537 ± 1.691	76.336 ± 4.319
Map-4	70.515 ± 17.973	72.865 ± 7.345	**52.732** ± 1.278	60.857 ± 5.792
Map-5	88.070 ± 24.244	62.833 ± 15.574	**61.496** ± 3.358	61.263 ± 5.168
Map-6	107.785 ± 11.148	85.722 ± 9.423	**77.210** ± 1.913	88.716 ± 8.465
Map-7	106.613 ± 9.329	81.411 ± 4.854	**81.296** ± 1.989	99.1992 ± 9.094
Map-8	127.309 ± 6.853	**104.060** ± 2.940	109.510 ± 2.980	121.322 ± 4.671
Map-9	29.979 ± 1.979	26.370 ± 0.64	**25.419** ± 0.620	26.347 ± 1.198
Map-10	75.685 ± 7.953	67.653 ± 3.865	**64.162** ± 1.471	71.300 ± 6.276

The boldface entries indicates the best result

Table 2 Time of the initial solution

	RRT*	IRRT*	ERCP	P-ACO
Map-1	0.251 ± 0.164	1.786 ± 0.846	**0.106** ± 0.047	52.997 ± 5.946
Map-2	0.507 ± 0.346	4.257 ± 2.012	**0.127** ± 0.073	55.069 ± 4.447
Map-3	**6.197** ± 3.076	52.212 ± 19.786	8.197 ± 4.097	74.371 ± 7.525
Map-4	6.906 ± 3.066	21.898 ± 11.256	**3.463** ± 2.808	58.283 ± 6.164
Map-5	8.767 ± 5.428	54.300 ± 24.416	**6.053** ± 3.891	32.935 ± 3.964
Map-6	10.063 ± 6.305	84.597 ± 36.913	**8.787** ± 5.451	48.789 ± 6.422
Map-7	**8.756** ± 3.989	67.729 ± 27.909	17.062 ± 13.923	54.789 ± 5.132
Map-8	**10.235** ± 6.850	59.952 ± 53.843	19.167 ± 10.766	78.371 ± 8.525
Map-9	1.298 ± 0.587	5.528 ± 2.604	**0.19** ± 0.023	27.278 ± 2.157
Map-10	1.112 ± 0.620	7.488 ± 1.659	**0.290** ± 0.058	68.612 ± 8.739

The boldface entries indicates the best result

solution, RRT*, IRRT*, and ERCP are 6.197 ± 3.076, 52.212 ± 19.786, and 8.197 ± 4.097, respectively. The better path quality performance, such as IRRT*, takes a longer time. The paths with better performance in terms of time, such as RRT*, do not converge enough.

In terms of time, ERCP achieves a spectacular victory, while P-ACO underperforms. The main reason for this phenomenon is that guided by the potential field, the pheromones produced by the ants are more concentrated in the part that tends to the target. However, P-ACO requires many iterations to approach the optimal solution. Too few iterations prevent the pheromone from converging sufficiently, and the quality of the obtained paths is poor. Too many iterations can result in excellent paths but consume considerable resources.

4.5 Influence of parameters

Another factor that affects the performance of the algorithm is the radius of x_{tar} and the step size of the space-filling tree. Many comparative experiments were conducted on Map-4 using RRT*, IRRT*, and ERCP to explain this relationship more clearly. The radius of x_{tar} was taken as 3 and 5 and the step size as 0.5, 1, and 2 to measure the relationship between the two in terms of path cost and the number of iterations, respectively. The relevant data are shown in Fig. 5. The red line indicates RRT*, the green line indicates IRRT*, and the light blue line indicates ERCP.

Figure 5a and b show the effect of different step sizes on the path cost, where the Y-axis scale is the path cost and the X-axis indicates the different step size values. Figure 5c and d show the effect of different radius and step size values on the number of iterations. The X-axis represents the different step size values, and the Y-axis represents the number of iterations. Figure 5 shows that the path cost and the number of iterations gradually decrease as the step size increases. When the step size is small, there are a large number of small, curved parts in the initial solution, which leads to a high path cost. Increasing the step size reduces the number of bends in the initial solution, which results in a smoother path. In addition, the increase in step size means that each step of RRT*, IRRT*, and ERCP can go farther; thus, χ_{tar} can be reached faster. This means that the number of iterations required to explore the initial solution decreases as the step size increases. The radius of x_{tar} mainly affects the number of iterations and has a relatively small effect on the path cost. The larger the radius of x_{tar} is, the faster the algorithm can reach χ_{tar}, and significantly fewer nodes are required to reach it.

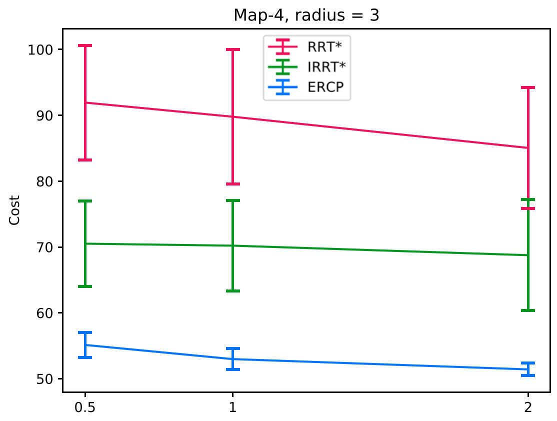
Table 3 Node of the initial solution

	RRT*	IRRT*	ERCP
Map-1	479 ± 173	463 ± 198	**240** ± 79
Map-2	596 ± 278	594 ± 272	**166** ± 67
Map-3	1960 ± 713	2012 ± 739	**1513** ± 630
Map-4	1413 ± 770	1368 ± 642	**861** ± 390
Map-5	3039 ± 1359	2946 ± 1176	**1914** ± 761
Map-6	**3726** ± 1307	3888 ± 1294	4246 ± 2055
Map-7	3363 ± 1077	3490 ± 1166	**3228** ± 1180
Map-8	5078 ± 1860	**5042** ± 1934	5504 ± 1816
Map-9	457 ± 171	426 ± 151	**401** ± 136
Map-10	998 ± 239	1129 ± 342	**331** ± 46

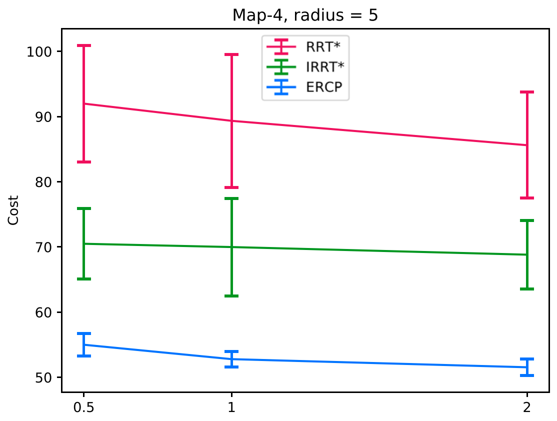
The boldface entries indicates the best result

4.6 Discussion

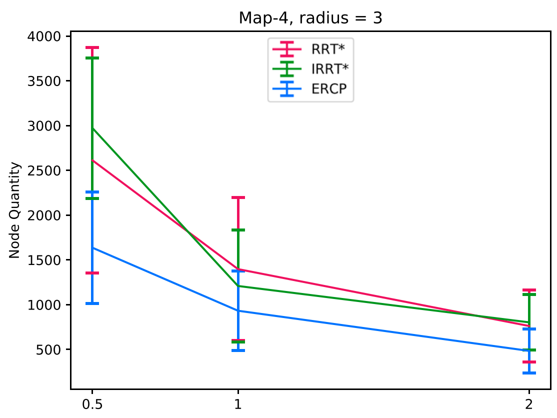
Combining these data, ERCP achieves a 70% win regarding path quality. In Table 1, the three failures are Map-1, Map-3, and Map-8. This is because IRRT* applies the ellipse heuristic to produce smoother paths. In contrast, ERCP, like RRT*, does not optimize the paths specifically, leaving them with some bends. Happily, the difference between the results obtained by ERCP and the path quality of the IRRT* algorithm is not significant. In terms of time, ERCP also achieves a 70% win, as shown in Table 2. IRRT* takes a longer time to obtain the initial solution. This



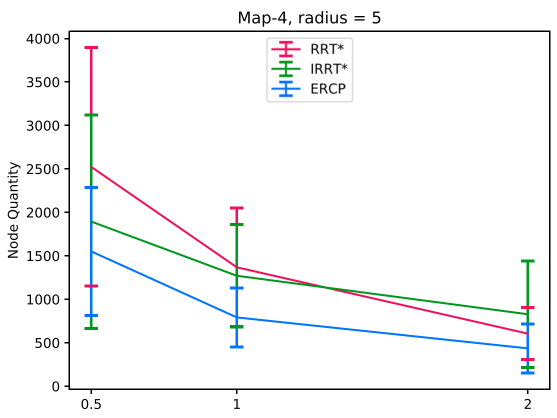
(a) Comparison of path cost on Map-4, the radius of x_{tar} is 3.



(b) Comparison of path cost on Map-4, the radius of x_{tar} is 5.



(c) Comparison of the number of nodes on Map-4 for obtaining the initial solution, the radius of x_{tar} is 3.



(d) Comparison of the number of nodes on Map-4 for obtaining the initial solution, the radius of x_{tar} is 5.

Fig. 5 Performance comparison of different combinations of step size and radius of x_{tar}. The X-axis indicates the different step sizes.The Y-axes in (a) and (b) indicate the path cost. The Y-axes in (c) and (d) illustrate the number of nodes required to obtain the initial solution. The red line indicates RRT*, the green line indicates IRRT*, and the light blue line indicates ERCP

is because IRRT* takes into account the optimization of the paths. ERCP does not optimize the initial solution. One reason is that the quality of the paths implemented by ERCP is good enough. Another reason is that ERCP considers the efficiency problems caused by the optimized paths. Regarding the number of iterations, ERCP achieves a victory of 80%. This is mainly due to the critical role of the effective sampling region, which makes it unnecessary for ERCP to explore the redundant state space. Therefore, it can be concluded that ERCP can strike a good balance between path quality and efficiency.

The above comparison experiments show that the exclusion of the redundant part of the *state space* enables ERCP to achieve excellent path quality. We performed hundreds of comparative experiments on ERCP, RRT*, IRRT*, and P-ACO under different conditions. The four algorithms are evaluated in terms of their initial solution cost, time consumption, and the number of iterations. The redundant *state space* is eliminated by performing clustering and presearching in a shorter period during the ERCP search. In this way, ERCP no longer searches the entire *state space* blindly. Then, ERCP spends less time exploring the effective sampling area to obtain excellent initial solutions. Many experiments indicate that ERCP can achieve a convincing balance between initial solution quality and efficiency under different conditions.

The results of clustering also have an impact on the initial solution cost. The main reasons for the parameters p and l of the Markov clustering algorithm include that p makes each pair of nodes in the graph have a stable probability of reaching other nodes. The main factor affecting the clustering effect is the parameter l, where a smaller l leads to larger individual clusters and a smaller total number of clusters, and vice versa. Reflecting in ERCP, a smaller l

leads to a larger effective sampling area, which makes the final path underconverge and thus degrades to a normal RRT* algorithm. A larger l leads to a smaller effective sampling area. In the extreme case, this may lead to a less helpful path planning phase, degrading ERCP to a grid-based path planning algorithm.

In ERCP, p and l obtain good results by taking the recommended values ($p = 2, l = 2$) of the Markov algorithm. This is due to the superiority of the Markov clustering algorithm itself.

However, the Markov clustering algorithm still has some areas for improvement. In some exceptional cases, there may be overlapping parts of the clustering results. The overlapping part means that the points in one cluster are shared by more than one cluster. Thankfully, overlapping clusters only occur in unique symmetric graphs. That is, only when some points are assigned with equal probability to more than one cluster do these clusters have the same structure as each other.

Taking Map-1 as an example, the whole map is symmetric with symmetric $source$ and $target$. On the one hand, we divide the clusters from left to right and top to bottom according to the clustering results, and the overlapping parts belong to the clusters divided first. On the other hand, the clustering result is only used as a basis for dividing the overall state space. In the path planning stage, it is converted into a continuous state space again. Even if there are overlapping clusters, the impact on the final generated paths is minimal.

5 Related work

Since the 1990s, the RRT algorithm has been widely used in path planning in several areas. However, the quality of the paths obtained by RRT is not guaranteed in most cases. Over many years, researchers have proposed different types of ways to improve the performance of RRT.

Improving RRT performance by improving the algorithm itself is one possible way. RRT* can converge to the optimal solution. However, it requires many resources. IRRT* [18] implements an elliptic heuristic search to obtain more rapid convergence. RRT*-Smart [19] has the same exploration and iteration strategy as RRT*. The difference is that after RRT*-Smart acquires the initial solution, the child nodes try to find the ancestor nodes. The tree structure will be modified if a qualified ancestor node is found. During the bending process, RRT*-Smart can find multiple anchor points. However, these anchor points are often close to obstacles and cannot directly help the child nodes to optimize. KB-RRT* [20] is a random tree method for fast bidirectional exploration. It avoids inefficient tree growth by proposing an effective branch pruning strategy. Compared with traditional path planning algorithms, KB-RRT* achieves better performance. However, there is still room for optimization of its generated paths. DT-RRT* [21] uses a bidirectional tree structure and separates the expansion and optimization processes. It has one original RRT for exploring the unknown environment and another modified RRT* for obtaining the optimal solution for the optimized path. MOD-RRT* [22] is an algorithm known as multiobjective dynamic fast exploration stochastic. The algorithm consists of a path generation process and a path optimization process. First, a modified RRT* is used to obtain an initial solution, and a state tree structure is generated as a priori knowledge. Then, a shortcut method is given to optimize the initial solution. In addition, it considers the case of infeasible paths. The algorithm is improved in terms of path cost, smoothness, and stability.

Improving the RRT with known partial information is a feasible approach. P-RRT* [23] uses an artificial potential field to guide RRT* for exploration. The addition of the artificial potential field provides direction for RRT* exploration and allows P-RRT* to converge faster than RRT*. Optimizing the algorithm in the structure of RRT* itself can also speed up the algorithm's convergence. Q-RRT* [24] uses triangular inequalities to improve the *nearest neighbor search* and *rewiring* processes. Compared to RRT*, Q-RRT* obtains a faster convergence rate. PQ-RRT* [25] is an improved algorithm based on P-RRT* and Q-RRT*. With the same resource consumption, PQ-RRT* produces better initial solutions and faster convergence of the optimal solution than P-RRT* and Q-RRT*. However, a point worth improving is that the effectiveness of PQ-RRT* depends on the initial parameter settings. Skilled-RRT [26] proposes a path planning method for regular 2D building environments. First, Skilled-RRT computes the skeleton in a 2D environment and then uses it to extend the RRT* and the path seeds. GMR-RRT* [27] learns from the demo and improves the convergence speed and path quality of RRT*. However, its generalization ability is poor.

Solving path planning problems by machine learning is also a promising candidate. Li et al. [28] uses a neural network to predict the cost function so that the generated paths are near optimal and designs a new reconstruction method for random search trees. The final paths generated by this algorithm achieve good results. Mohammadi et al. [29] predicts reliable paths via generative adversarial networks [30]. However, it has poor generalization capabilities since it is trained and tested on a straightforward and small neural network. NRRT* [31] uses a convolutional neural network [32] to predict promising sampling regions for RRT*. It speeds up obtaining the optimal solution of RRT* to some extent. However, there is still potential for improvement in the predicted regions it generates. LM-RRT [33] guides multiple RRTs to explore the space through a reinforcement learning algorithm. This strategy improves the RRT's ability to explore the local space. However, it leads to an unusually time-consuming RRT extension process.

6 Conclusion and further work

In this paper, we combine the advantages of unsupervised clustering algorithms, grid-based algorithms, and sampling-based algorithms. We propose an enhanced RRT* algorithm with clustering and presearching (ERCP). In addition, we study a large number of cases to compare ERCP with RRT* and IRRT*. It is worth mentioning that ERCP is a breathtaking winner in comparison with P-ACO, an intelligent algorithm that combines the advantages of traditional algorithms. The effect of different parameters on ERCP performance is discussed at the end of the experiment. The results show that ERCP strikes a convincing balance between the quality and efficiency of the initial solution and has strong robustness and superiority.

The advantage is that ERCP can generate sufficiently good and stable paths for an arbitrary unknown environment while ensuring efficiency. Moreover, ERCP can reuse the valid sampled area multiple times after preprocessing for the same environment. This is one of the unique advantages of ERCP.

The main limitation of our approach is that the resulting effective sampling area is too small in a few cases, resulting in poor intercluster connectivity and thus curvature of the generated paths. Therefore, the Bessel curve method is needed to optimize the paths.

In the future, we can collect real-world datasets and apply ERCP to test its practical application value. Another exciting direction is obtaining the state space's environment semantics by adapting the first two phases of ERCP to extend to dynamic environments.

Acknowledgements This work is in part supported by the Central Government Funds of Guiding Local Scientific and Technological Development (No. 2021ZYD0003), the Sichuan Province Youth Science and Technology Innovation Team (No. 2019JDTD0017), the Science and Technology Planning Project of Sichuan Province (No. 2021YFS0391), and the Scientific Research Project of State Grid Sichuan Electric Power Company Information and Communication Company (No. SGSCXT00XGJS2100116).

Data Availability Data will be available upon reasonable request.

References

1. Siciliano B, Khatib O, Kröger T (2008) Springer handbook of robotics, vol 200. Springer, New York
2. Liu Y, Xiao F, Tong X, Tao B, Xu M, Jiang G, Chen B, Cao Y, Sun N (2022) Manipulator trajectory planning based on work subspace division. Concurr Comput Pract Experience 34(5):e6710
3. Ab Wahab MN, Nefti-Meziani S, Atyabi A (2020) A comparative review on mobile robot path planning: classical or meta-heuristic methods? Ann Rev Control 50:233–252
4. Hart PE, Nilsson NJ, Raphael B (1968) A formal basis for the heuristic determination of minimum cost paths. IEEE Trans Syst Sci Cybernet 4(2):100–107
5. Stentz A (1997) Optimal and efficient path planning for partially known environments. In: Intelligent unmanned ground vehicles, Springer, pp 203–220
6. Lingelbach F (2004) Path planning using probabilistic cell decomposition. In: IEEE international conference on robotics and automation, vol 1. IEEE, pp 467–472
7. Li B, Liu H, Su W (2019) Topology optimization techniques for mobile robot path planning. Appl Soft Comput 78:528–544
8. Khatib O (1985) Real-time obstacle avoidance for manipulators and mobile robots. In: IEEE international conference on robotics and automation, vol 2. pp 500–505
9. Luo Q, Wang H, Zheng Y, He J (2020) Research on path planning of mobile robot based on improved ant colony algorithm. Neural Comput Applic 32(6):1555–1566
10. Kennedy J, Eberhart RC (1997) A discrete binary version of the particle swarm algorithm. In: IEEE international conference on systems, man, and cybernetics. computational cybernetics and simulation. vol 5. pp 4104–4108
11. Shi K, Huang L, Jiang D, Sun Y, Tong X, Xie Y, Fang Z (2022) Path planning optimization of intelligent vehicle based on improved genetic and ant colony hybrid algorithm. Front Bioeng Biotechnol 10:905983
12. Zhang X, Xiao F, Tong X, Yun J, Liu Y, Sun Y, Tao B, Kong J, Xu M, Chen B (2022) Time optimal trajectory planing based on improved sparrow search algorithm. Front Bioeng Biotechnol 10:852408
13. Karaman S, Frazzoli E (2011) Sampling-based algorithms for optimal motion planning. Int J Robot Res 30(7):846–894
14. Hsu D, Latombe JC, Motwani R (1997) Path planning in expansive configuration spaces. In: Proceedings of international conference on robotics and automation. vol 3. pp 2719–2726
15. Kavraki LE, Svestka P, Latombe JC, Overmars MH (1996) Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Trans Robot Autom 12(4):566–580
16. LaValle SM (1998) Rapidly-Exploring Random Trees: A New Tool for Path Planning. Technical Report. pp 98–11
17. Dongen V, Marinus S (2000) Graph Clustering by Flow Simulation
18. Gammell JD, Srinivasa SS, Barfoot TD (2014) Informed RRT*: optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In: IEEE/RSJ international conference on intelligent robots and systems. pp 2997–3004
19. Islam F, Nasir J, Malik U, Ayaz Y, Hasan O (2012) RRT*-Smart: rapid convergence implementation of RRT* towards optimal solution. In: IEEE international conference on mechatronics and automation. pp 1651–1656
20. Wang J, Li B, Meng MQH (2021) Kinematic constrained bi-directional RRT with efficient branch pruning for robot path planning. Expert Syst Appl 170:114541
21. Chen L, Shan Y, Tian W, Li B, Cao D (2018) A fast and efficient double-tree RRT*-like sampling-based planner applying on mobile robotic systems. IEEE/ASME Trans Mechatron 23(6):2568–2578
22. Qi J, Yang H, Sun H (2021) MOD-RRT*: a sampling-based algorithm for robot path planning in dynamic environment. IEEE Trans Ind Electron 68(8):7244–7251

23. Qureshi AH, Ayaz Y (2016) Potential functions based sampling heuristic for optimal path planning. Auton Robot 40(6):1079–1093
24. Jeong IB, Lee SJ, Kim JH (2019) Quick-RRT*: triangular inequality-based implementation of RRT* with improved initial solution and convergence rate. Expert Syst Appl 123:82–90
25. Li Y, Wei W, Gao Y, Wang D, Fan Z (2020) PQ-RRT*: an improved path planning algorithm for mobile robots. Expert Syst Appl 152:113425
26. Dong Y, Camci E, Kayacan E (2018) Faster RRT-based nonholonomic path planning in 2D building environments using skeleton-constrained path biasing. J Intell Robot Syst 89(3):387–401
27. Wang J, Li T, Li B, Meng MQH (2022) GMR-RRT*: sampling-based path planning using Gaussian mixture regression. IEEE Trans Intell Veh
28. Li Y, Cui R, Li Z, Xu D (2018) Neural network approximation based Near-Optimal motion planning with kinodynamic constraints using RRT. IEEE Trans Ind Electron 65(11):8718–8729
29. Mohammadi M, Al-Fuqaha A, Oh JS (2018) Path planning in support of smart mobility applications using generative adversarial networks. In: IEEE international conference on internet of things and IEEE green computing and communications and IEEEcyber, physical and social computing and IEEE smart data. pp 878–885
30. Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, Bengio Y (2014) Generative adversarial nets. In: Conference and workshop on neural information processing systems
31. Wang J, Chi W, Li C, Wang C, Meng MQH (2020) Neural RRT*: learning-based optimal path planning. IEEE Trans Autom Sci Eng 17(4):1748–1758
32. Fukushima K (1980) Neocognitron: a self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biol Cybern 36(4):193–202
33. Wang W, Zuo L, Xu X (2018) A learning-based multi-RRT approach for robot path planning in narrow passages. J Intell Robot Syst 90(1):81–100