CDT-Dijkstra: Fast Planning of Globally Optimal Paths for All Points in 2D Continuous Space

Jinyuan Liu, Minglei Fu, Wenan Zhang, *Member, IEEE,* Bo Chen, *Member, IEEE,*
Ryhor Prakapovich, and Uladzislau Sychou

Abstract— The Dijkstra algorithm is a classic path planning method, which in a discrete graph space, can start from a specified source node and find the shortest path between the source node and all other nodes in the graph. However, to the best of our knowledge, there is no effective method that achieves a function similar to that of the Dijkstra's algorithm in a continuous space. In this study, an optimal path planning algorithm called convex dissection topology (CDT)-Dijkstra is developed, which can quickly compute the global optimal path from one point to all other points in a 2D continuous space. CDT-Dijkstra is mainly divided into two stages: SetInit and GetGoal. In SetInit, the algorithm can quickly obtain the optimal CDT encoding set of all the cut lines based on the initial point x_{init}. In GetGoal, the algorithm can return the global optimal path of any goal point at an extremely high speed. In this study, we propose and prove the planning principle of considering only the points on the cutlines, thus reducing the state space of the distance optimal path planning task from 2D to 1D. In addition, we propose a fast method to find the optimal path in a homogeneous class and theoretically prove the correctness of the method. Finally, by testing in a series of environments, the experimental results demonstrate that CDT-Dijkstra not only plans the optimal path from all points at once, but also has a significant advantage over advanced algorithms considering certain complex tasks.

Index Terms— Path Planning, Topology, Path Homotopy, Optimal Distance, Mobile Robots

I. INTRODUCTION

Considering the continuous progress in science and technology in recent years, an unprecedented rapid development of technologies has been achieved in the field of robotics [1]. As an important research direction in artificial intelligence, robotics, and automatic control, the path-planning method aims to find a continuous trajectory in the state space for robots that connect the initial and target states [2], [3]. Path-planning algorithms are now widely used in various fields, such as autonomous driving [4], mobile robotics [5], game development [6], smart homes [7], and biomedical applications [8].

Graph search-based algorithms are widely used in path-planning. These algorithms share the common characteristic of using graph models to represent the environment or search space and to find the shortest or optimal path in a graph. Dijkstra's algorithm [9] is a breadth-first search algorithm that can find the shortest path from each node in the graph to the source node. The A* algorithm [10] is an improvement of the Dijkstra algorithm, which incorporates a heuristic function to improve the search efficiency while ensuring the shortest path. Graph search-based path-planning algorithms are often stable and perform well; however, their search process is conducted on a discrete graph, making it difficult to solve optimal path-planning problems in continuous spaces. While [11] achieves the utilization of graph search in continuous space by visibility graph, such methods are not efficient in complex environments, cause the search for visible points becomes hard, and the visibility graph can become large.

Randomized sampling-based algorithms have become a popular research topic in the field of motion planning owing to their ability to avoid discretization problems of graph search techniques. The rapidly-exploring random Tree (RRT) algorithm [12] is the most commonly used sampling-based algorithm. Subsequently, researchers have proposed various improvements to the RRT algorithm. The RRT* algorithm incorporates a rewiring process into the RRT sampling process, ensuring both probabilistic completeness and asymptotic optimality [13]. The informed-RRT* algorithm was proposed for path planning in complex environments, and it uses a non-uniform sampling strategy to converge to the optimal path more quickly [14]. The batched informed trees (BIT*) [15] and adaptively informed trees (AIT*) [16] algorithms combine the advantages of graph search algorithms and sampling-based algorithms, use heuristics for all aspects of path cost to prioritize the search for highquality paths and focus on improvements.

Based on cell decomposition, the path planning method is highly effective in addressing road connectivity and rapid planning problems in static environments. These methods mainly involve dividing the map into a series of connected regions or polygons, connecting these regions or polygons to form a connected graph, and usinge the connection relationship of cells to determine a collision-free path that the robot can run [17], [18]. However, path-planning algorithms based on cell decomposition only consider connectivity and ignore other properties, such as optimality (e.g., path length) and computational complexity (e.g., time for partitioning the environment into a finite set of regions) [19]. This often requires the assistance of other algorithms to obtain a globally optimal path. For example, methods referenced in previous studies created heuristic topological maps using

generalised Voronoi diagrams (GVD), which then guide the sampling process of RRT to ensure that the algorithm can find the globally optimal path [20], [21]; these heuristic paths are not inevitably optimal and the GVD initialization time for complex maps is extensive.

In recent years, the concept of path homotopy has received significant attention in the field of path planning. Homotopy is a concept in topology [22] that refers to the ability of two paths in a continuous space to transform into one another without colliding with obstacles. In path planning, homotopy can be used to verify whether two paths are topologically equivalent, and obstacles in the environment divide the possible path space of the robot into a countable number of homotopy classes. These homotopy classes can serve as a high-level abstraction of robot paths and are a powerful tool for solving robot planning tasks [23]–[25]. Homotopy invariants are labels used to identify the homotopy class to which a path belongs. In a 2D space, the h-signature is commonly used as a homotopy invariant. However, the h-signature does not directly record the connectivity of the space; therefore, it cannot guide the path-planning process. In addition, it is relatively difficult to deduce the reference path in a homotopy class based on the h-signature. In our previous study [26], we developed a more effective homotopy invariant, this is, a homotopy path class encoder based on convex dissection topology (CDT). The CDT Encoder can easily encode paths in a homotopy class to the same CDT encoding, and the CDT encoding can also be quickly decoded into a reference trajectory in the homotopy class. The CDT Encoder constructs a topological graph that contains the connectivity information of the environmental space, which allows it to effectively guide the path planning process.

In the previous afore [26] mentioned studies, graph-search-based planning methods have difficulty in solving the problem of optimal path planning in a continuous space. The cell decomposition-based method is commonly used to inspire sampling-based planning methods; however, it is difficult to ensure that these heuristic paths are optimal. In response to these issues, this study presents the highly challenging path-planning task of efficiently planning a globally optimal path from one point to all other points in a 2D continuous space.[1]

In this study, we mainly investigated the aforementioned task by combining the relevant theories of cell decomposition, graphs, and homotopy invariants. The contributions of our study can be summarized as follows:

- Based on the theoretical foundation of map convex division [26], we further proposed a planning principle that only considers the points on the cutting lines, which reduces the state space of the optimal path planning task from 2D to 1D.
- A method for efficiently finding the optimal path in the homotopy class in a map partitioned by convex division is proposed.

---

[1]The description here is not rigorous because there are infinitely many points in the continuous space. Thus, if an algorithm can very rapidly return the optimal path from any goal point to the initial point after processing the initial point, it is considered compliant with this description.

- an algorithm called CDT-Dijkstra was proposed, which can quickly plan globally-optimal paths from one point to all other points in a 2D continuous space.

The problem formulation, relevant mathematical definitions, and proofs are presented in Section II. Details of our algorithm are discussed in Section III. The experimental results are presented and analyzed in Section IV, which are followed by the conclusions in Section V.

II. PRELIMINARIES

This study was mainly based on the theoretical foundation of our previous work [26]. In this study, we directly adopt the relevant mathematical expressions and theoretical foundations presented in our previous study [26] without redundancy. Further details regarding the referenced study can be found in Sections II and III of [26].[2]

A. Path Planning Problem Formulation

This study investigated the problem of distance-optimized path planning for mobile robots in a 2D bounded space. The problem is defined as follows:

Let $X \subset \mathbb{R}^2$ be the state space; the obstacle space and free space are denoted as X_{obs} and $X_{free} = X/X_{obs}$, respectively. The path planning algorithm aims to determine a feasible path $f(t)$ such that

$$f \in P(X_{free}; x_{init}, x_{goal}), \tag{1}$$

where x_{init} denotes the initial state and x_{goal} denotes the goal state. Let $P_f = P(X_{free}; x_{init}, x_{goal})$. The optimal path[3] planning problem can be defined as follows:

$$f^{\circledast} = \arg \min_{f \in P_f} S(f), \tag{2}$$

where $S : P(X) \to \mathbb{R}_{\geqslant 0}$ is a function of the path length, which is defined as follows:

$$S(f) = \lim_{n \to N^+} \sum_{t=0}^{n} \|f(t + n^{-1}) - f(t)\|, \tag{3}$$

where $\| \cdot \|$ is the 2-Norm in Euclidean space and N^+ is a sufficiently large positive integer.

Compared to typical path planning tasks that only search for the optimal path between two given points, this study focuses on how to quickly find the optimal paths from x_{init} to all the points in a space, given the initial point x_{init}.

B. Dimensionality Reduction of State Space

Lemma 1: A necessary and sufficient condition for $f^* \in P(X_{free})$ to be the shortest path in $[f^*]$ is that, for any $t_1, t_2 \in [0, 1]$, there exists the shortest path $g(t) = f^*(t_1 + (t_2 - t_1)t)$ in $[g]$.

Proof:

---

[2]Since [26] is still under review, we temporarily publicise its implementation to verify the correctness of the method presented in this paper, at `https://arxiv.org/abs/2302.13026`. In fact, the mathematical representation of paths in this paper is consistent with the standard representation in topology.

[3]In this study, $f^{\circledast}$ represents the global optimal path with the same start and end points as f, and f^* represents the local optimal path in $[f]$.

Necessary: The law of proof by contradiction is used herein. Suppose f^* is the shortest path in $[f^*]$, and $\exists t_1, t_2 \in [0, 1]$ such that $g(t) = f^*(t_1 + (t_2 - t_1)t)$ is not the shortest path in $[g]$; that is, there exists $g' \simeq_p g$ such that $S(g') < S(g)$. The path f^* can be expressed as $f^* \cong_p g_s * g * g_e$, where

$$g_s(t) = f^*(a \cdot t), \qquad a = \min(t_1, t_2), \quad (4)$$
$$g_e(t) = f^*(b + (1 - b)t), \qquad b = \max(t_1, t_2). \quad (5)$$

Hence,

$$\begin{aligned} S(f^*) &= S(g_s) + S(g) + S(g_e) \\ &< S(g_s) + S(g') + S(g_e) \quad (6) \\ &< S(g_s * g' * g_e). \end{aligned}$$

Therefore, there exists a path $g_s * g' * g_e$ in $[f^*]$ that is shorter than f^*. This contradicts our original hypothesis. Therefore, the sufficiency condition is true.

Sufficient: When the latter of the proposition holds, let $t_1 = 0$ and $t_2 = 1$. At this moment $g(t) = f^*(t)$. Therefore, f^* is the shortest path in $[f^*]$. Therefore, the necessary condition is true. ∎

Theorem 2: For any optimal path $f^* \in P(X_{free})$ where $x_s = f^*(0)$, $x_e = f^*(1)$. The f^* must be expressed in the form of multiple line splices as follows. When x_s and x_e are within the same convex polygon,

$$f^* \cong_p l_{x_s}^{x_e}. \quad (7)$$

When x_s and x_e are not within the same convex polygon,

$$f^* \cong_p l_{x_0}^{x_1} * l_{x_1}^{x_2} * \cdots * l_{x_{n-1}}^{x_n}, \quad (8)$$

where $x_0 = x_s$, $x_n = x_e$, $x_1, \ldots, x_{n-1}$ is the intersection of f^* in turn with the cutlines.

Proof: According to the properties of convex polygons, (7) is established, and according to the characteristics of the convex dissection method used in [26] (refer to Subsection III.A of [26]), the points $x_0, x_1, \cdots, x_n$ in (8) exist. Therefore, f^* can be expressed as follows:

$$f^* \cong_p f_1 * f_2 * \cdots * f_n, \quad (9)$$

where $f_k \in P(X_{free}, x_{k-1}, x_k)$. Moreover, based on **Lemma 1**, the optimal path f^* must guarantee that each of its segments is optimal; hence, for any f_k, $f_k = l_{x_{k-1}}^{x_k}$. ∎

According to **Theorem 2**, we only need to consider the points on the cutlines. The state space of the optimal path planning task is reduced from the 2D space of X_{free} to a 1D space on the cutlines.

C. Problem Simplification

In this subsection, we simplify the problem addressed in this study. Under the guidance of **Lemma 1** and **Theorem 2**, we can easily obtain the following corollary:

Corollary 3: When x_s and x_e are not within the same convex polygon, there must exist a point x_c on its adjacent[4]

---

[4]We define the cutlines adjacent to a point as the cutlines of the convex polygon it belongs to, and consider the points on the cutline as belonging to two convex polygons simultaneously.

cutlines such that the optimal path $f_e^\circledast$ from x_s to x_e can be expressed as follows:

$$f_e^\circledast \cong_p f_c^\circledast * l_{x_c}^{x_e}, \quad (10)$$

where $f_c^\circledast$ is the optimal path from x_s to x_c.

Based on the afore mentioned, the optimal path $f_e^\circledast$ and $f_c^\circledast$ share the same CDT encoding in **Corollary 3**; that is, $\Gamma \circ f_e^\circledast \cong_p \Gamma \circ f_c^\circledast$. Thus we can derive a new constraint for the global optimal path, that is, when x_{goal} and x_{init} are not within the same convex polygon, the following is obtained:

$$\Gamma \circ f^\circledast \in \Gamma \circ \bigcup_k C_k^\circledast = \bigcup_k (\Gamma \circ C_k^\circledast). \quad (11)$$

Here, $C_k^\circledast$ represents the set of optimal paths for the all points on each adjacent cutline of x_{goal}. Note, the set $C_k^\circledast$ has an infinite number of elements, but its CDT encoding set $\Gamma \circ C_k^\circledast$ has a finite number of elements. Therefore, for a given x_{init}, if we can obtain the optimal CDT encoding set of all the cutlines, it is easy to obtain the global optimal path from any x_{goal} to x_{init}. These optimal CDT encoding sets of cutlines can be obtained by iterating outward from the adjacent cutlines of x_{init} using (10) and (11). For convenience, we use ξ_{c_k} to represent the optimal CDT encoding set $\Gamma \circ C_k^\circledast$ corresponding to cutline c_k in the following.

III. APPROACH

In this subsection, we propose and explain the details of a unique optimal path planning algorithm, CDT-Dijkstra, based on a convex division topological graph. The characteristic of this algorithm is as follows: (i) given an initial point x_{init}, it can quickly obtain the optimal CDT encoding set for all the cutlines, and (ii) for any goal point x_{goal} in the free space X_{free}, it can return the optimal path from x_{init} to x_{goal} at a very high speed. The CDT corresponding to its characteristics can be divided into two independent stages (threads), which are namely SetInit and GetGoal. A visualization of CDT-Dijkstra at work is shown in Fig. 3.

A. Main Algorithm

The main part of the CDT-Dijkstra is shown in Algorithm 1. For lines 1-2 of the algorithm we use the method in [26] to construct the topological graph X_{tp} corresponding to the free space X_{free} based on the input Map. In line 3, the optimal CDT encoding set for each cutline will be created. Thereafter, the algorithm enters the SetInit stage (lines 4-19).

In the SetInit stage, the algorithm obtains the initial point of the planning task (line 5) and updates the sets $\{\xi_{c_1}, \xi_{c_2}, \cdots, \xi_{c_n}\}$ when x_{init} changes (lines 6-18). To introduce the pseudocode of the update process, we define a special class of symbol $\hat{f}_c$ used in the pseudocode.

Definition 4 (Symbol $\hat{f}_c$): Let c be the cutline connecting two convex polygons, denoted by X_{x_1} and X_{x_2}, respectively. The corresponding nodes of X_{x_1} and X_{x_2} in X_{tp} are denoted by x_1 and x_2, respectively. The symbol $\hat{f}_c$ represents both (x_1, x_2) and (x_2, x_1) in $P(X_{tp})$. When $\hat{f}_c$ is involved in the $*$ operation, it uses the form that makes the operation meaningful.
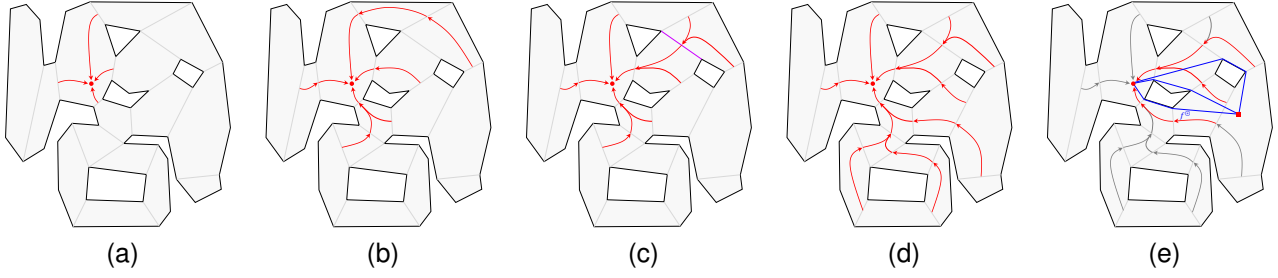
Fig. 1. Visual illustration of CDT-Dijkstra at work. Firstly, (a) create the CDT encoding sets of the cutlines adjacent to the initial point, followed by (b) gradually updating the CDT encoding sets for other cutlines outward. (c) The diagram of Algorithm 4, when the red line is added, it affects the CDT encoding set of its adjacent cutline. (d) The computation of CDT encoding set for all cutlines is completed. (e) All valid CDT encodings adjacent the goal point are found and the globally optimal path is obtained.

During the update process, all encoding sets are first emptied. Thereafter, the encoding sets for the cutlines adjacent to x_{init} are set, and a set ϑ_{ex} for recording the existing cutlines is created (lines 8-10). In line 11 of the algorithm, a list ϑ_{add} is created to record the cutlines that can be added to ϑ_{ex}.

The algorithm iterates in lines 12-18. During each iteration, the algorithm pops a cutline c_k from ϑ_{add} and uses Algorithm 3 to update the encoding sets ξ_{c_k} corresponding to c_k (lines 13,14). The algorithm then pushes c_k into ϑ_{ex}, and pushes the unused cutlines adjacent to ξ_{c_k} into ϑ_{add}. Similar to the rewiring of RRT*, the change in the optimal homotopy class code in ξ_{x_k} has an impact on other encoding sets. Therefore, at the end of each iteration, we optimized the existing encoding sets using Algorithm 4. When ϑ_{add} is empty, the update to sets $\xi_{c_1}, \xi_{c_2}, \cdots, \xi_{c_n}$ is complete.

Algorithm 1 CDT-Dijkstra.

1: **Input:** Map
2: $X_{tp} \leftarrow$ BuildTopologyGraph(Map)
3: Initialise $\{\xi_{c_1}, \xi_{c_2}, \cdots, \xi_{c_n}\}$ with the number of cutlines

4: **loop**
5:      $x_{init} \leftarrow$ WaitingTaskInput()
6:      **if** x_{init} is changed **then**
7:          Fill $\{\xi_{c_1}, \xi_{c_2}, \cdots \xi_{c_n}\}$ with $\varnothing$
8:          $\vartheta_{ex} \leftarrow$ AdjacentCutlines(x_{init})
9:          **for** $c_k \in \vartheta_{ex}$ **do**
10:             push $(\Gamma \circ e_{x_{init}}) * \hat{f}_{c_k}$ to ξ_{c_k}
11:          $\vartheta_{add} \leftarrow$ AdjacentCutlines(ϑ_{ex})
12:          **repeat**
13:             $c_k \leftarrow$ PopFirst(ϑ_{add})
14:             $\xi_{c_k} \leftarrow$ GetHomotopyClassesForCutline(c_k)
15:             Push AdjacentCutlines(c_k)/ϑ_{ex} to ϑ_{add}
16:             Push c_k to ϑ_{ex}
17:             OptimizedExistingHomotopyClasses(c_k)
18:          **until** ϑ_{add} is empty
19: **end loop**

After the SetInit stage, CDT-Dijkstra can enter the Get-Goal stage, and the global optimal path from x_{init} to x_{goal} can be quickly obtained using Algorithm 2. In Algorithm 2, if x_{init} and x_{goal} are in the same convex polygon, the

algorithm directly returns $l^{x_{goal}}_{x_{init}}$ (lines 2,3). Otherwise, the algorithm finds all the CDT encoding on cutlines adjacent to x_{goal} (lines 4-7). Finally, these CDT encodings are mapped into the path space $P(X_{free}; x_{init}, x_{goal})$ using Algorithm 5, and the global optimal path $f^{\circledast}$ is determined (lines 8-14). In line 10 of the algorithm, we need to determine whether x_{goal} is in the convex polygon corresponding to the last element of f because the cutline belongs to two different convex polygons at the same time, and its encoding set may record the CDT encoding to the other convex polygon.

Algorithm 2 GetGoal.

1: **Input:** x_{goal}
2: **if** x_{init}, x_{goal} in the same convex polygon **then**
3:      **return** $l^{x_{goal}}_{x_{init}}$
4: $\xi_{goal} \leftarrow \varnothing$
5: $\vartheta_{goal} \leftarrow$ AdjacentCutlines(x_{goal})
6: **for** $c_k \in \vartheta_{goal}$ **do**
7:      $\xi_{goal} \leftarrow \xi_{goal} \cup \xi_{c_k}$
8: $cost \leftarrow \infty$
9: **for** f $\in \xi_{goal}$ **do**
10:      **if** $x_{goal} \in$ f(-1) **then**
11:          $f^* \leftarrow$ GetShortestPath(f, x_{init}, x_{goal})
12:          **if** $cost > S(f^*)$ **then**
13:             $f^{\circledast} \leftarrow f^*$
14:             $cost \leftarrow S(f^*)$
15: **return** $f^{\circledast}$

B. Get Homotopy Classes For Cutline

The function of Algorithm 3 is to update the encoding set of the input cutline c_k according to the current encoding sets of adjacent cutlines in ϑ_{ex}. According to (10) and (11), the element in the encoding set of c_k must be expressed as the product of the element in its adjacent encoding sets and $\hat{f}_{c_k}$. In Algorithm 3, we first determine all the codes of the encoding sets adjacent to c_k (lines 2,3). In lines 5-14 of Algorithm 3, ξ_u is used to compute the optimal path for each point on c_k.[5] In this process, if encoding f can make f $* \hat{f}_{c_k}$ meaningful and f $* \hat{f}_{c_k}$ can obtain the optimal path

---

[5] We will discretize c_k; when the interval is very small, the discretisation has no effect on the planning results.

to a certain point (lines 8-14), then $f * \hat{f}_{c_k}$ is placed into an encoding set ξ_{out}.

Algorithm 3 GetHomotopyClassesForCutline.

1: **Input:** c_k
2: $\vartheta_{near} \leftarrow$ AdjacentCutlines$(c_k) \cap \vartheta_{ex}$
3: $\xi_u \leftarrow \bigcup_{c_i \in \vartheta_{near}} \xi_{c_i}$
4: $\xi_{out} \leftarrow \varnothing$
5: **for** $t \in [0,1]$ **do**
6:      $x_t \leftarrow c_k(t)$
7:      $cost_{old} \leftarrow \infty$
8:      **for** $f \in \xi_u$ **do**
9:          **if** $f * \hat{f}_{c_k}$ is meaningful **then**
10:              $f^* \leftarrow$ GetShortestPath$(f * \hat{f}_{c_k}, x_{init}, x_t)$
11:              $cost_{new} \leftarrow S(f^*)$
12:              **if** $cost_{old} > cost_{new}$ **then**
13:                  $f_{min} \leftarrow f * \hat{f}_{c_k}$
14:                  $cost_{old} \leftarrow cost_{new}$
15:      $\xi_{out} \leftarrow \xi_{out} \cup \{f_{min}\}$
16: **return** ξ_{out}

C. Optimized Existing Homotopy Classes

The purpose and process of the optimization by Algorithm 4 are similar to the rewiring operation of RRT*. First, Algorithm 4 creates a list ϑ_r of cutlines to be optimized based on the input c_k, which subsequently starts the optimization of the encoding set of cutlines in ϑ_r. The optimization of the encoding set of all the cutlines in ϑ_{ex} is complete when ϑ_r is empty (lines 3-9). In each optimization iteration, the algorithm pops a cutline c_r to be optimized from ϑ_r and recomputes the encoding set of that cutline. If the encoding set of this cutline changes, the algorithm will push the cutlines adjacent to c_r in ϑ_{ex} into ϑ_r.

Algorithm 4 OptimizedExistingHomotopyClasses.

1: **Input:** c_k
2: $\vartheta_r \leftarrow$ AdjacentCutlines$(c_k) \cap \vartheta_{ex}$
3: **repeat**
4:      $c_r \leftarrow$ PopFirst(ϑ_r)
5:      $\xi_{temp} \leftarrow$ GetHomotopyClassesForCutline(c_r)
6:      **if** $\xi_{c_r} \neq \xi_{temp}$ **then**
7:          $\xi_{c_r} \leftarrow \xi_{temp}$
8:          Push AdjacentCutlines$(c_k) \cap \vartheta_{ex}$ to ϑ_r
9: **until** ϑ_r is empty

D. Get Shortest Path

Algorithm 5 is primarily used to fast obtain the shortest path in the homotopy path class. In line 2 of Algorithm 5, $\epsilon_1, \epsilon_2, \cdots, \epsilon_m$ are the cutlines through which f passes subsequently. In line 3 $x_0 = x_s$, and $x_{m+1} = x_e$, $x_1, x_2, \cdots, x_m$ are the midpoints of the cutlines $\epsilon_1, \epsilon_2, \cdots, \epsilon_m$ respectively. Thereafter, Algorithm 5 enters an iterative compression process for f^* (lines 5-9). From line 9, it is evident that f^* can be given by the sequence $\{x_0, x_1, \cdots, x_m, x_{m+1}\}$. Thus lines 6-9 can be abstracted

as map $\Psi : P(X_{free}) \rightarrow P(X_{free})$, and map Ψ consists of m submaps $\psi_1, \psi_2, \cdots, \psi_m$ (line 8), such that:

$$\Psi = \psi_m \circ \psi_{m-1} \circ \cdots \circ \psi_1. \tag{12}$$

For any $\psi_k \in \{\psi_1, \psi_2, \cdots, \psi_m\}$ there is:

$$\begin{aligned}
& l_{x_0}^{x_1} * \cdots * l_{x_{k-1}}^{x_k'} * l_{x_k'}^{x_{k+1}} * \cdots * l_{x_n}^{x_{n+1}} \\
& = \psi_k \circ (l_{x_0}^{x_1} * \cdots * l_{x_{k-1}}^{x_k} * l_{x_k}^{x_{k+1}} * \cdots * l_{x_n}^{x_{n+1}}).
\end{aligned} \tag{13}$$

To simplify, let:

$$g = l_{x_0}^{x_1} * \cdots * l_{x_{k-2}}^{x_{k-1}} \quad \text{and} \quad h = l_{x_{k+1}}^{x_{k+2}} * \cdots * l_{x_n}^{x_{n+1}}. \tag{14}$$

According to line 7 of the algorithm,

$$\begin{aligned}
S(g * l_{x_{k-1}}^{x_k} * l_{x_k}^{x_{k+1}} * h) &= S(g) + S(l_{x_{k-1}}^{x_k} * l_{x_k}^{x_{k+1}}) + S(h) \\
&\geqslant S(g) + S(l_{x_{k-1}}^{x_k'} * l_{x_k'}^{x_{k+1}}) + S(h) \\
&= S(g * l_{x_{k-1}}^{x_k'} * l_{x_k'}^{x_{k+1}} * h) \\
&= S(\psi_k \circ (g * l_{x_{k-1}}^{x_k} * l_{x_k}^{x_{k+1}} * h)).
\end{aligned} \tag{15}$$

Hence, every map in $\{\psi_1, \psi_2, \cdots, \psi_m\}$ is compressed by f^*. According to (12), the map Ψ is also compressed for f^*.

Therefore, Algorithm 5 can find the shortest path between input points x_s and x_e in the homotopy class f. In addition, the solution to line 8 of the algorithm is simply given by the following: (i) If lines $l_{x_{k-1}}^{x_{k+1}}$ and ϵ_k intersect, x_k is their intersection, and (ii) if lines $l_{x_{k-1}}^{x_{k+1}}$ and ϵ_k do not intersect, x_k is the endpoint of ϵ_k that is closest to $l_{x_{k-1}}^{x_{k+1}}$.

According to Line 5 of Algorithm 3, we must execute Algorithm 5 for each point x_t on the segmentation line (line 10 of Algorithm 3), which is clearly time-consuming. However, for the neighboring points $x_{t'}$ of x_t, the locally optimal paths calculated using Algorithm 5 for each CDT encoding $f * \hat{f}_{c_k}$ are significantly similar. Therefore, in practical programming, it is only necessary to run Algorithm 5 once for x_t at $t = 0$. For the subsequent x_t, when running Algorithm 5, $l_{x_0}^{x_1} * l_{x_1}^{x_2} * \cdots * l_{x_n}^{x_{n+1}}$ in line 4 can be directly replaced with the locally optimal path of x_{t-dt}, which can significantly greatly reduce the number of iterations of the 5-10 lines. During testing, this optimization method can reduce the number of iterations to 2-4.

Algorithm 5 GetShortestPath.

1: **Input:** f, x_s, x_e
2: Use f to get the cutline sequence $\{\epsilon_1, \epsilon_2, \cdots, \epsilon_m\}$
3: $\{x_0, x_1, \cdots, x_m, x_{m+1}\} \leftarrow \{x_s, \epsilon_1(\frac{1}{2}), \cdots, \epsilon_m(\frac{1}{2}), x_e\}$
4: $f^* \leftarrow l_{x_0}^{x_1} * l_{x_1}^{x_2} * \cdots * l_{x_n}^{x_{n+1}}$
5: **repeat**
6:      $cost \leftarrow S(f^*)$
7:      **for** $k \in \mathbb{N}_1^m$ **do**
8:          $x_k = \underset{x \in \epsilon_k}{\arg\min} S(l_{x_{k-1}}^x * l_x^{x_{k+1}})$
9:      $f^* \leftarrow l_{x_0}^{x_1} * l_{x_1}^{x_2} * \cdots * l_{x_n}^{x_{n+1}}$
10: **until** $S(f^*) - cost < \varepsilon$
11: **return** f^*

TABLE I

INITIALIZATION OF FOUR MAPS

Map Name	Map Image Resolution	Cutlines	Convex Polygons	Time (ms)
Cluttered	$1000 * 1000$	135	122	19.7 ± 2.5
Trap	$1000 * 1000$	38	37	9.2 ± 3.3
Maze	$1000 * 1000$	95	95	15.4 ± 3.6
Museum	$9706 * 9706$	646	608	397.2 ± 18.9

IV. RESULTS

In this section, we present the results of the several simulation experiments that were conducted to demonstrate the effectiveness and efficiency of CDT-Dijkstra. An Intel NUC (next unit of computing) computer was used, which was an Intel i7-1165G7 (4.7GHz) with 32 GB of RAM. For the simulation, we used C++ on Ubuntu 20.04. As shown in Fig. 3, we chose three regular environments (cluttered, trap, and maze) and a challenging mega map (2D map of the German Museum) to test our algorithm in this experiment. Fig. 3(e)-(h) demonstrate the results of the convex dissection of the four test environments using the method in [26]; details of the initialized construction of the topological space X_{tp} are listed in **Table I**.

CDT-Dijkstra is a unique path-planning algorithm that consists of two parts: SetInit and GetGoal. To demonstrate that this algorithm achieves the effect of needing one run to obtain the optimal path from all points to the initial point, we set up four planning tasks in each environment, where the optimal path was planned from the initial point S to goal points 1-4, respectively. In the experiment, CDT-Dijkstra executed SetInit[6] once for the initial point S in each environment, and subsequently used GetGoal to obtain the optimal path for each goal point separately; the time spent on SetInit and GetGoal was recorded. This operation was performed 100 times for each environment. The average values of SetInit and GetGoal times for each task are shown in Fig. 2.

Based on the experiments, the SetInit stage of CDT-Dijkstra consumed the majority of the algorithm time; however, it maintains a high level of real-time performance (the average usage was 17.5 ms, 3.7 ms, 2.4 ms, and 28.8 ms). Moreover, the time used by the GetGoal stage remains in the sub-microsecond range, even for complex tasks such as Task 4 in Fig. 3(d), where the time used by GetGoal does not exceed 1.5 us; these times are almost negligible. Therefore, we consider that CDT-Dijkstra achieves a fast search for the optimal path from one point to all the other points in a continuous space.

To determine the advantages and disadvantages of CDT-Dijkstra compared to general optimal path planning algorithms, we present a comparison of the results between the proposed CDT-Dijkstra algorithm and the following four advanced algorithms in the OMPL open source framework: RRT* algorithm, PRM* algorithm [13], Informed-RRT*
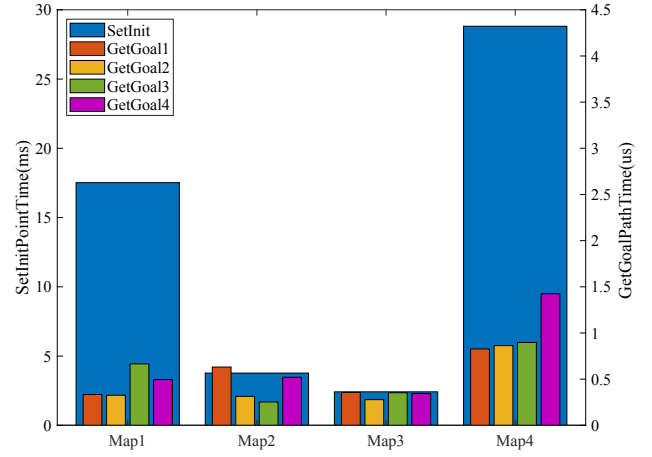

Fig. 2. CDT-Dijkstra test results. The timescales of the SetInit stage and GetGoal stages are on the left and right, respectively.

TABLE II

THE ABBREVIATION FOR EACH CRITERIA TYPE FOR EACH ALGORITHM

Criteria	RRT*	PRM*	Informed-RRT*	ABIT*
t_{init}	R_{f}	P_{f}	I_{f}	A_{f}
$t_{1\%}$	R_{o}	P_{o}	I_{o}	A_{o}

[14], and advanced batch informed trees algorithm (ABIT*) [27]. Two parameters were used to compare the performance of the algorithms t_{init}, including the time of the initial solution and $t_{1\%}$, which is the time required to determine a solution of cost $1.01 \times Cost_{opt}$, where $Cost_{opt}$ is the optimal cost. These four algorithms were executed 100 times for each task, as shown in Fig. 3(a)-(h), and the mean values of t_{init} and $t_{1\%}$ for these algorithms for each task are shown in Fig. 3(i)-(l), respectively. For a better comparison with other algorithms, the completion times of the CDT-Dijkstra algorithms of SetInit and AllInit are indicated by the blue and red lines, respectively, where AllInit is equal to the SetInit time plus the corresponding map initialization[7] time show in **Table I**.

The experimental results demonstrate that CDT-Dijkstra not only plans the optimal path from all the points to the starting point by running SetInit once, but also has a significant advantage over other algorithms when managing complex tasks. The planning speed of CDT-Dijkstra is one to three orders of magnitude faster than the other four algorithms in the maze and museum environments, especially in the museum environment, where the other algorithms require 10-50 times more time than CDT-Dijkstra to find the initial path; CDT-Dijkstra also performs better in the trap environment. However, based on the experimental results, we also found that CDT-Dijkstra performed relatively poorly in certain simple tasks, such as cluttered environments and Task 3 in trap environments. These phenomena are related to

---

[6]In the experiment, we discretized the cutlines with an interval of two pixels in length.

[7]In practice, a map only needs to be initialized once. The initialisation result can be saved as a file that can be directly accessed for future use.
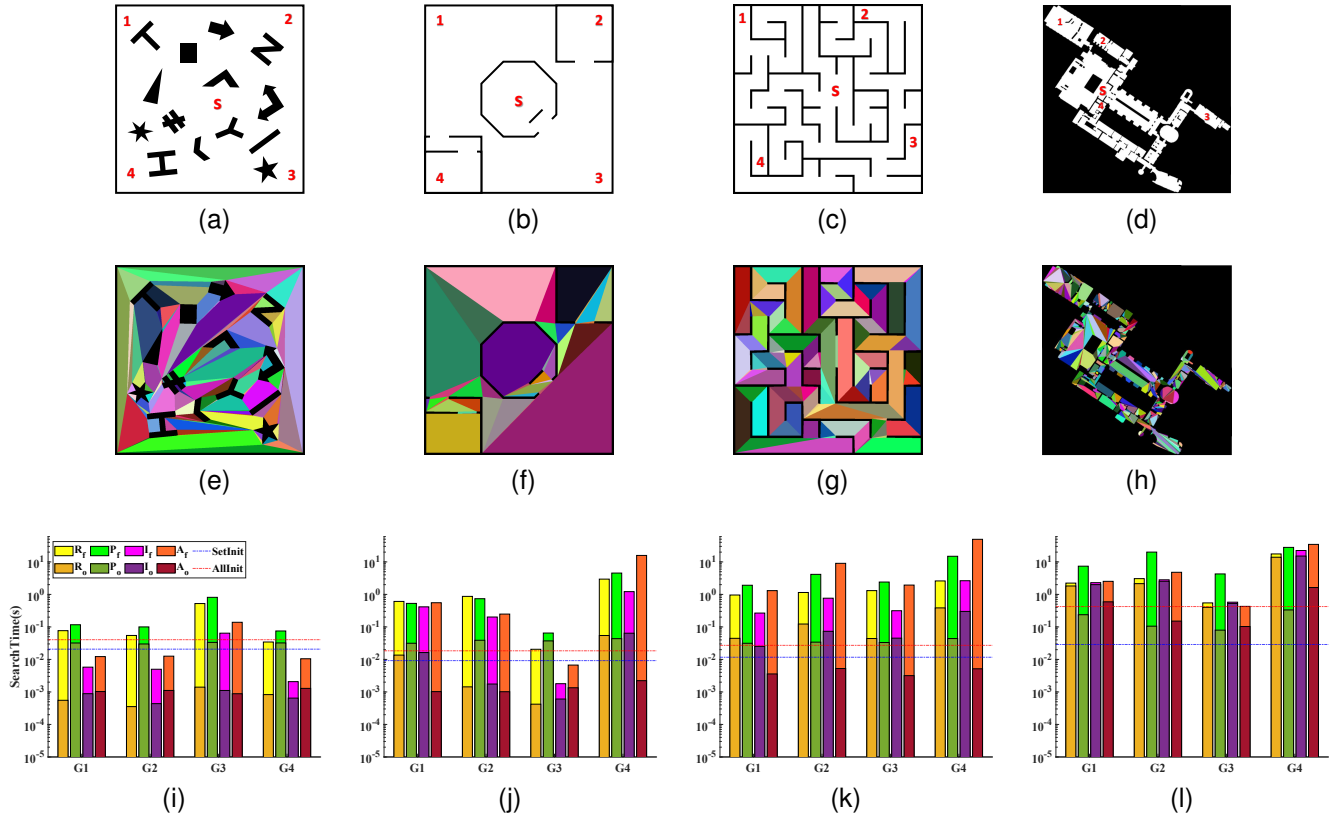
Fig. 3. Illustration of the experiment map and planning tasks. (a) Cluttered environment, (b) trap, (c) maze and (d) 2D map of the German Museum. Each environment contains four planning tasks starting from S and ending G. (e)-(h): convex division results of each environment. (i)-(l): performance statistics of each planner in the comparative experiment. The completion times of the CDT-Dijkstra algorithms of SetInit and AllInit are indicated by the blue and red lines, respectively, where AllInit is equal to the SetInit time plus the corresponding map initializatio.

the working principle of the CDT-Dijkstra. When calculating the CDT encoding set of cutlines, CDT-Dijkstra is guided by the topological graph X_{tp}, and updates in an orderly manner from the cutline near the initial point to the outside. Therefore, CDT-Dijkstra is not affected by the twists and turns of free space and the narrow gaps of the traps during operation. On the other hand, sampling-based optimal path planning methods are more easily trapped. In the trap environment shown in Fig. 3, the sampling-based methods encounter significantly fewer challenges in Task 3 compared to several other tasks, thus they perform better in Task 3 and plan faster than CDT-Dijkstra.

The issue reflected by the experimental data show in Fig. 3(i) must be analyzed in conjunction with the number and distribution of independent obstacles in the experimental environment. Here, an independent obstacle refers to the subset of obstacles in the obstacle space X_{obs} that are surrounded by the free space X_{free}. The number of independent obstacles directly determines the number of homotopy classes of paths between any two connected points in free space; this relationship is exponential with base 2. CDT-Dijkstra can remove the most invalid homotopy classes by orderly updating outward from the cutline near the initial point when calculating the CDT encoding set of the cutline. However, when several independent obstacles are uniformly

distributed near the initial point, CDT-Dijkstra will waste significant time in Algorithm 4 because, when the computation of the CDT encoding set of a new cutline is completed, the CDT encoding set of the existing adjacent cutlines is easily changed; that is, the condition in line 6 of Algorithm 4 is more likely to be triggered. The data presented in **Table III** also support the aforementioned, with the trap and maze environments having a relatively small number of independent obstacles and $\xi_{c_r} \neq \xi_{temp}$ being triggered significantly less often than the other two environments. Comparing the cluttered and museum environments, we find that although the museum has a higher number of independent obstacles, the independent obstacles in cluttered environments are more evenly distributed; therefore, $\xi_{c_r} \neq \xi_{temp}$ is triggered significantly more often in a cluttered environment than in a museum. Therefore, in cluttered environments, CDT-Dijkstra has a slightly weaker performance than other sampling-based methods when performing optimal path planning tasks between two points.

V. CONCLUSIONS

In this study, based on the theoretical framework of the CDT encoder, we propose a planning principle that only considers the points on the cutline, thereby reducing the state space in the path-planning task from 2D to 1D. Based

TABLE III

EFFECT OF THE NUMBER OF INDEPENDENT OBSTACLES IN THE MAP

Maps	Cluttered	Trap	Maze	Museum
Number of Independent Obstacles	14	2	1	38
Number of Homotopy Classes	16384	4	2	2^{38}
Number of triggers for $\xi_{cr} \neq \xi_{temp}$	5437	44	32	962

on this principle, for a given x_{init}, only the optimal CDT encoder set of all the cutlines can be obtained, and the global optimal path from any x_{goal} to x_{init} can be easily obtained. Based on the aforementioned work, we propose a unique optimal path-planning algorithm called CDT-Dijkstra. Guided by the topological graph X_{tp}, CDT-Dijkstra completes the calculation of the CDT encoding set of each cutline in an orderly manner from the cutline near the initial point outward, subsequently returning at an extremely fast global optimal path to any goal point. The CDT-Dijkstra algorithm implements a fast computation of the global optimal path from one point to all the other points in a 2D continuous space. In addition, we also proposed a method for rapidly finding the optimal path in a homotopy class (Algorithm 5), proved the correctness of this method through theories, and provided an optimization scheme for this method in practical applications. Finally, by testing in a series of environments, the experimental results demonstrate that CDT-Dijkstra not only plans the optimal path from all points at once, but also has a significant advantage over other algorithms when encountering certain complex tasks, that is, CDT-Dijkstra is not affected by the twists and turns of free space and the narrow gaps of traps during operation.

In practical robotics applications, CDT-Dijkstra can be run in a dual-threaded manner, that is, one thread runs SetInit according to the target position of the robot, and the other thread uses GetGoal to update the optimal path of the current position of the robot in real-time during the robot movement. CDT-Dijkstra is ideal for application to tasks in which a robot is required to move from a fixed starting point to a location in space and return; for example, inspection and food delivery robots.

REFERENCES

[1] L. Wang, S. Liu, H. Liu, and X. V. Wang, "Overview of human-robot collaboration in manufacturing," in *Proceedings of 5th International Conference on the Industry 4.0 Model for Advanced Manufacturing: AMP 2020*. Springer, 2020, pp. 15–58.

[2] X. Zhang, J. Wang, Y. Fang, and J. Yuan, "Multilevel humanlike motion planning for mobile robots in complex indoor environments," *IEEE Transactions on Automation Science and Engineering*, vol. 16, no. 3, pp. 1244–1258, 2018.

[3] S. D. Bopardikar, B. Englot, and A. Speranzon, "Multiobjective path planning: Localization constraints and collision probability," *IEEE Transactions on Robotics*, vol. 31, no. 3, pp. 562–577, 2015.

[4] T. Duhautbout, R. Talj, V. Cherfaoui, F. Aioun, and F. Guillemard, "Efficient speed planning in the path-time space for urban autonomous driving," in *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2022, pp. 1268–1274.

[5] S. G. Tzafestas, "Mobile robot control and navigation: A global overview," *Journal of Intelligent & Robotic Systems*, vol. 91, no. 1, pp. 35–58, 2018.

[6] K. Naderi, J. Rajamäki, and P. Hämäläinen, "Rt-rrt* a real-time path planning algorithm based on rrt," in *Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games*, 2015, pp. 113–118.

[7] H. Do, A. V. Le, L. Yi, J. C. C. Hoong, M. Tran, P. Van Duc, M. B. Vu, O. Weeger, and R. E. Mohan, "Heat conduction combined grid-based optimization method for reconfigurable pavement sweeping robot path planning," *Robotics and Autonomous Systems*, vol. 152, p. 104063, 2022.

[8] Y. Zhao, Y. Wang, J. Zhang, X. Liu, Y. Li, S. Guo, X. Yang, and S. Hong, "Surgical gan: Towards real-time path planning for passive flexible tools in endovascular surgeries," *Neurocomputing*, 2022.

[9] E. W. Dijkstra, "A note on two problems in connexion with graphs:(numerische mathematik, 1 (1959), p 269-271)," 1959.

[10] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[11] T. Lozano-Pérez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," *Communications of the ACM*, vol. 22, no. 10, pp. 560–570, 1979.

[12] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," *The international journal of robotics research*, vol. 20, no. 5, pp. 378–400, 2001.

[13] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[14] J. D. Gammell, T. D. Barfoot, and S. S. Srinivasa, "Informed sampling for asymptotically optimal path planning," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 966–984, 2018.

[15] ——, "Batch informed trees (bit*): Informed asymptotically optimal anytime search," *The International Journal of Robotics Research*, vol. 39, no. 5, pp. 543–567, 2020.

[16] M. P. Strub and J. D. Gammell, "Adaptively informed trees (ait*) and effort informed trees (eit*): Asymmetric bidirectional sampling-based path planning," *The International Journal of Robotics Research*, vol. 41, no. 4, pp. 390–417, 2022.

[17] M. Kloetzer, C. Mahulea, and R. Gonzalez, "Optimizing cell decomposition path planning for mobile robots using different metrics," in *2015 19th international conference on system theory, control and computing (ICSTCC)*. IEEE, 2015, pp. 565–570.

[18] Z. Li, Z. Zhang, H. Liu, and L. Yang, "A new path planning method based on concave polygon convex decomposition and artificial bee colony algorithm," *International Journal of Advanced Robotic Systems*, vol. 17, no. 1, p. 1729881419894787, 2020.

[19] R. Gonzalez, M. Kloetzer, and C. Mahulea, "Comparative study of trajectories resulted from cell decomposition path planning approaches," in *2017 21st International Conference on System Theory, Control and Computing (ICSTCC)*. IEEE, 2017, pp. 49–54.

[20] W. Chi, Z. Ding, J. Wang, G. Chen, and L. Sun, "A generalized voronoi diagram-based efficient heuristic path planning method for rrts in mobile robots," *IEEE Transactions on Industrial Electronics*, vol. 69, no. 5, pp. 4926–4937, 2021.

[21] S.-K. Huang, W.-J. Wang, and C.-H. Sun, "A path planning strategy for multi-robot moving with path-priority order based on a generalized voronoi diagram," *Applied Sciences*, vol. 11, no. 20, p. 9650, 2021.

[22] J. R. Munkres, *Elements of algebraic topology*. CRC press, 2018.

[23] S. Bhattacharya, M. Likhachev, and V. Kumar, "Search-based path planning with homotopy class constraints in 3d," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 26, no. 1, 2012, pp. 2097–2099.

[24] S. McCammon and G. A. Hollinger, "Topological path planning for autonomous information gathering," *Autonomous Robots*, vol. 45, pp. 821–842, 2021.

[25] J. Wakulicz, K. M. B. Lee, T. Vidal-Calleja, and R. Fitch, "Topological trajectory prediction with homotopy classes," *arXiv preprint arXiv:2301.09821*, 2023.

[26] J. Liu, M. Fu, A. Liu, W. Zhang, B. Chen, R. Prakapovich, and U. Sychou, "Homotopy path class encoder based on convex dissection topology," 2023. [Online]. Available: https://arxiv.org/abs/2302.13026

[27] M. P. Strub and J. D. Gammell, "Advanced bit*(abit*): Sampling-based planning with advanced graph-search techniques," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 130–136.