2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

Tree-based Representation of Locally Shortest Paths for 2D k-Shortest Non-homotopic Path Planning

Tong Yang, *Member, IEEE*, Li Huang, *Student Member, IEEE*,
Yue Wang, *Member, IEEE*, and Rong Xiong, *Member, IEEE*

***Abstract*—A novel algorithm to solve the 2D *k-shortest non-homotopic path planning* (*k-SNPP*) task is proposed in this paper. The task is of practical significance as a sub-module for higher-level planning and scheduling tasks, and is gaining increasing attention and focus in recent years. There have existed algorithms that explicitly characterised non-homotopic paths using topological invariants such as h-signature and winding number. However, these algorithms are inefficient due to their separate treatment of topology and geometry: Topological invariants are singularly utilised for distinguishing non-homotopic property among paths, which significantly increases the volume of the robot configuration space. Meanwhile, distance-optimal path planners search for locally shortest paths in the augmented space, which becomes extremely time-consuming. In this paper, a topological tree is proposed to simultaneously leverage topology and geometry. The tree grows from the starting location and explores all topological routes, until the best k of its leaves reach the goal. It is proven that different branches of the tree explore different homotopy classes of paths, and all the branches are locally shortest. Comparative experiments for k-SNPP are conducted in challenging grid-based simulated environments to validate the performance of the proposed algorithm. The C++ implementation of the proposed algorithm is released for the benefit of the robotics community.**

I. Introduction

The pathfinding of locally shortest non-homotopic paths in a 2D environment, a set of robot paths that bypass environmental obstacles in different topological directions, has emerged as a crucial application for mobile robots in recent years. Its significance becomes evident as a fundamental sub-module of higher-level planning tasks. For instance, when coordinating a group of mobile robots [1] to travel from a common starting location to a common goal location, following non-homotopic locally shortest paths can effectively reduce congestion-related issues, conflicts, and overall travelling costs. Moreover, finding locally shortest non-homotopic paths also serves other purposes, including providing good alternative routes [2] for a robot navigating in a dynamic environment, providing multiple initial paths for non-homotopic trajectory optimisation [3], and holding potential applications in multi-robot explorations [4].

Corresponding author: Yue Wang and Rong Xiong.
Tong Yang, Yue Wang, and Rong Xiong are with the State Key Laboratory of Industrial Control and Technology, Zhejiang University, Hangzhou, 310027, China (e-mail: *t.yang22@imperial.ac.uk*; *ywang24@zju.edu.cn*; *rxiong@zju.edu.cn*).
Li Huang is with the Institute of Advanced Digital Technologies and Instrumentation, Zhejiang University, Hangzhou, 310027, China (e-mail: *li.huang@zju.edu.cn*).
This work was supported by the National Key R&D Program of China under Grant 2021ZD0114500.

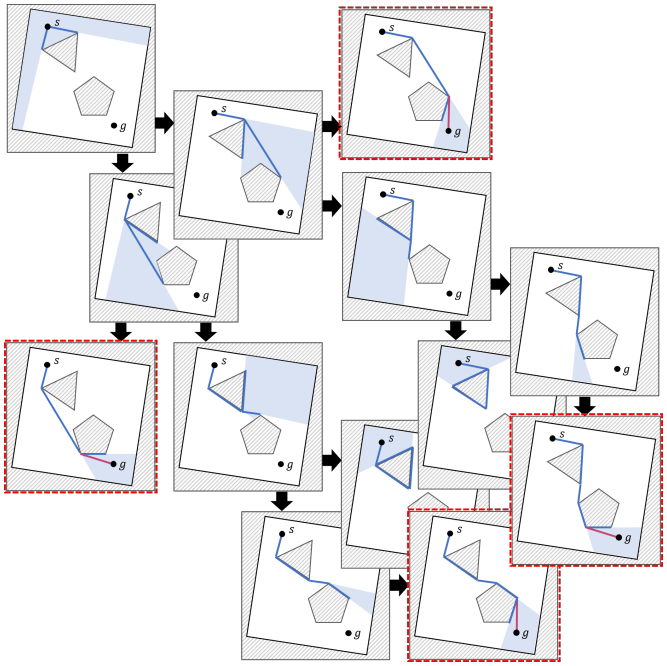


Fig. 1. Illustration of the homotopy-aware path planning process from s to g using the proposed algorithm. The environment is considered as the 2D configuration space of the mobile robot, treating the robot as a particle. In each sub-figure, the blue region is the *cell* of the node, which is an angle-ranged section of the visibility region constructed from the *source point* of the node. Blue paths are tree edges. When a cell covers the goal location, a straight path is created to the goal, finalising the construction of a locally shortest path. Given the current state of the tree which has 13 nodes, four non-homotopic locally shortest paths have been constructed, indicated by dashed red boxes.

Existing algorithms typically solved the *k-shortest non-homotopic path planning* (k-SNPP) problem by formally defining the *homotopy-augmented graph* [5] and conducting path searching within this graph. Popular strategies for parameterising homotopy-augmented space leveraged topological invariants, such as h-signature [5] and winding number [6], which were followed by a distance-optimal pathfinding technique such as A* [7] and RRT* [8] to formulate a valid k-SNPP solver. However, as a discretised representation of the covering space [9] of the 2D collision-free environment, when self-crossing robot paths are allowed, the homotopy-augmented space theoretically becomes an infinite space. Furthermore, the volume of the searching space exponentially increases with the number of internal obstacles in the environment, making the k-SNPP task computationally unaffordable. Although there have been techniques to restrict the range for

pathfinding, such as limiting the number of cycles that the robot path can wind around an obstacle, or predicting a subset of homotopy classes that the k resultant paths might rest in, the k-SNPP task remains challenging even for simple scenarios. This suggests that explicit construction of a homotopy-augmented graph followed by a common pathfinding technique might not be an appropriate approach.

In this paper, an efficient k-SNPP solver is proposed which adopts 2D visibility region as the building block to construct a topological tree. See Fig. 1 for illustration. The tree is topology-aware, because it is proven to find paths in all homotopy classes of paths. Also, it is geometry-aware, because the tree edges connecting parent node and child node are proven to be locally shortest. What is more important is that the concatenation of tree edges keeps this local distance optimality. The combination of geometry awareness and topology awareness ensures that every locally shortest resultant path is a branch of the tree. The solution departs from other visibility-based algorithms that took the costly but unnecessary explicit calculation of the visibility graph, to instead leveraging only visibility region which has linear complexity in relation to the number of obstacle vertices. This makes the proposed algorithm highly efficient and computationally affordable, even in grid-based environments. The contributions of this paper can be listed as follows:

1) The introduction of a novel topological tree, which is an alternative representation of the homotopy-augmented graph of a 2D environment.
2) An efficient k-SNPP solver based on the tree structure.
3) The open-sourcing of the proposed algorithm [1].

The remainder of this paper is organised as follows. Section II reviews existing literature. Section III formally states the k-SNPP problem. Section IV presents the proposed algorithm in detail. Discussions of the completeness and optimality of the proposed algorithm are provided in Section V. Experimental evaluations are collected in Section VI, with final concluding remarks gathered in Section VII.

II. Related Works

A. Homotopy-aware Planning Algorithms

The k-SNPP task is treated as a challenging sub-topic of the homotopy-aware path planning problem in the past decades. This subsection provides a brief overview of the development of homotopy-aware path planners.

Early efforts in finding non-homotopic paths essentially focused on generating multiple paths using either searching-based [5] or sampling-based algorithms [6], with the hope that some of these resultant paths turn out to be non-homotopic. Later, the introduction of "visibility condition" offered a sufficient criterion for identifying the homotopic property between two paths [10]. However, it still cannot guarantee the generation of non-homotopic resultant paths during the planning process. More recently, explicit characterisation of the homotopy-augmented space has been proposed by utilising topological invariants, specifically, the h-signature [5] and the winding number [6] of the paths connecting the starting location and the goal location. The 2D goal location was augmented into multiple goal "configurations", each possessing distinct topological indices. As such, finding the shortest path to a goal configuration in the homotopy-augmented space was mathematically equivalent to finding the 2D locally shortest path in the specific homotopy class indicated by the topological invariants of the goal configuration. This is a significant milestone, as it ensures the construction of non-homotopic resultant paths, which however comes at the cost of significantly increased computational complexity. This is the most apparent difference between the algorithm to be proposed in this paper and previous algorithms, because in our algorithm topological invariants are not explicitly incorporated. Instead, they will implicitly guide the expansion directions of tree.

B. Map-abstraction-based Algorithms

A range of algorithms effectively abstracted the environmental topology [11] [12] or geometry [13]. For geometry awareness, the Visibility graph [13] and its generalisation, the Tangent graph [14], have played prominent roles in collecting all the path segments that are required for constructing locally shortest path in a 2D environment. The probabilistic roadmap (PRM) [15] randomly selected and connected valid robot locations via collision-free straight paths, creating a graph of potential robot routes. In terms of topology awareness, the Voronoi graph [16] captured unique path segments between two obstacles, ensuring that different paths derived from a Voronoi graph must be non-homotopic in the original environment. Recent years have also seen the appearance of many variants [17] [18] [19] [20], which are omitted here.

It is crucial to note that the proposed algorithm is not a representation of the environment. In fact, it is unnecessary to exhaustively represent the entire environment, given that we already have the starting location and the goal location. As evidence of the limitations of this type of algorithms, the Visibility graph [13] struggles to handle all the vertices of obstacles with intricate shapes in a reasonable computational time. Usually, it is used along with a polygonal approximation algorithm, leading to the lack of distance optimality. Similarly, given a limited computational capability, PRM [15] cannot always disperse samples into narrow passages, leading to its incompleteness in describing the topology of potential robot routes. Whilst the Voronoi graph [12] guarantees pairwise non-homotopic resultant paths, the k-best paths are not necessarily homotopic to the k-shortest non-homotopic paths in the original environment. This indicates that the Voronoi graph does not encapsulate the distance optimality. In contrast, the proposed algorithm only acquires necessary topological and geometric information for the specific k-SNPP task, leading to its validity and high efficiency.

III. Problem Statement

The discussed map scenarios in this paper assume a 2D, polygonal, and Euclidean configuration space $\tilde{M}$, which is

[1] https://github.com/ZJUTongYang/raystar

the same as the scenarios suitable for the Visibility graph algorithm. To be precise, the proposed algorithm is well-suited for the following cases:

1) The robot is modelled as a point particle, whilst environmental obstacles are represented as polygons.
2) The robot's footprint is polygonal and the robot's kinematic model is translational. Obstacles in this case are also polygonal.
3) The robot is circular in shape, and both the robot's circular shape and the shapes of obstacles are approximated using grids. This aligns with the most widely adopted settings in contemporary Costmap applications.

Two planning problems are addressed in this paper. The *k-shortest non-homotopic path planning (k-SNPP)* problem is defined as finding k resultant paths satisfying the following criteria:

1) (**Validity**) Each path is free of collisions.
2) (**Non-homotopic Property**) Each pair of these resultant paths cannot be continuously deformed within the collision-free environment.
3) (**Distance Optimality**) Each of the resultant paths is a locally shortest path between the starting location and the goal location.
4) (**k-Best Property**) Among all locally shortest paths, the resultant paths are the k shortest ones.

And another essential variation of the k-SNPP is enforcing the resultant paths to be non-selfcrossing. This is referred to as the *non-selfcrossing k-SNPP* problem.

IV. Algorithm

In this section, the proposed algorithm is presented in detail. The pseudo-code of the proposed algorithm is provided in **Algorithm 1**.

A. Definitions

Some definitions are provided to assist the presentation of the proposed algorithm. First of all, the terminology *cell* is defined to refer to an angle-ranged part of visibility region [21].

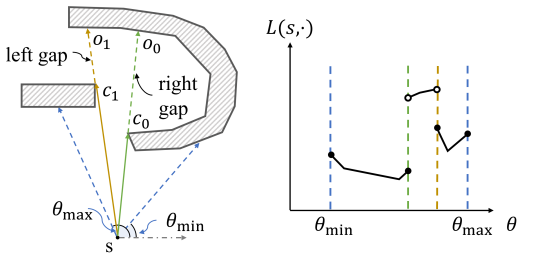
Definition 1. *(Cell) Given the configuration space $\tilde{M}$, a source point s, and an angle interval $[\theta_{\min}, \theta_{\max}]$, a cell is an angle-ranged part of the visibility region [21], i.e., the subset of points p in the visibility region such that*

$$0 \le \text{wrapTo2Pi}(\text{atan2}(\overrightarrow{sp}) - \theta_{\min}) \le \theta_{\max} - \theta_{\min} \tag{1}$$

Then, gap structures are defined within a cell as follows:

Definition 2. *(Gap) Given the cell constructed from $\tilde{M}$, s, and $[\theta_{\min}, \theta_{\max}]$, along each orientation $\theta \in [\theta_{\min}, \theta_{\max}]$, the distance from s to the nearest obstacle, denoted as $L(s,\theta)$, is well-defined as:*

$$L : [\theta_{\min}, \theta_{\max}] \to \mathbb{R},\ \theta \mapsto L(s,\theta) \tag{2}$$



(a) cell and gap (b) distance to nearest obstacle

Fig. 2. Illustration of the cell structure. (a) Visualisations of symbols. (b) The corresponding distance function $L(s,\cdot)$.

Algorithm 1 k-SNPP Solver

Input: 2D configuration space $\tilde{M}$, starting location p_{start}, goal location p_{goal}, the number of resultant paths k
Output: resultant paths R

R=∅ % the list of resultant paths
Q=∅ % the priority queue
Node 0=Node(0, p_{start}, [p_{start}], [0, 2π], 0, $\| p_{\text{start}} - p_{\text{goal}} \|_2$)
if isGoalInCell(V^0, p_{goal}) **then**
    R.push([R^0, p_{goal}])
end if
for the j-th child of node 0 **do**
    Q.push([0, j, g_j^0+h_j^0])
end for
n=1 % the number of expanded nodes
while !isEmpty(Q) **do**
    [i, j, ~]=selectBestChild(Q)
    Q.pop()
    node n = Node(n, c_j^i, R_j^i, [$\theta_{\min,j}^i$, $\theta_{\max,j}^i$], g_j^i, h_j^i)
    if isGoalInCell(V^n, p_{goal}) **then**
        R.push([R^n, p_{goal}])
    end if
    for the j-th child of node i **do**
        if isNonSelfcrossing(R_j^n) **then**
            Q.push([n, j, $g_j^n + h_j^n$])
        end if
    end for
    n = n+1
    if R.size()≥ k **then**
        break
    end if
end while

A gap is formed when L is discontinuous. Two kinds of gaps are distinguished based on the different orientations that the robot needs to turn in order to "enter" the gap:

$$\begin{aligned} \lim_{t\to\theta^-} L(s,t) > \lim_{t\to\theta^+} L(s,t) &\Rightarrow \textit{a left gap} \\ \lim_{t\to\theta^-} L(s,t) < \lim_{t\to\theta^+} L(s,t) &\Rightarrow \textit{a right gap} \end{aligned} \tag{3}$$

And the limits at boundaries, $\lim_{t\to\theta_{\min}^-} L(s,t)$ and $\lim_{t\to\theta_{\max}^+} L(s,t)$, are assigned when the cell is constructed. For notations, the near obstacle point is denoted as c, and the far obstacle point is denoted as o. See Fig. 2 for illustration.

Some computer geometry-related discussions about corner cases, such as the de-duplication when $[\theta_{\min}, \theta_{\max}] = [0, 2\pi]$,

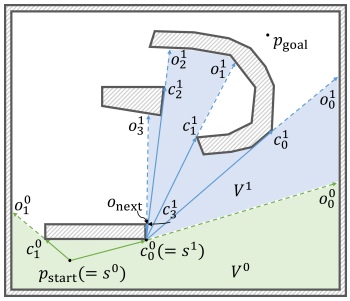


Fig. 3. Illustration of node expansion. In this example, the first child of Node 0 is expanded as Node 1. When Node 1 is constructed, its source point is c_0^0 which is the right-bottom corner of the rectangular obstacle, and the gap is left-turning. Hence the right-top corner is selected as o_{next}. Other variables are provided for reference regarding the indexing convention used in this paper.

where a gap might appear both as a left gap at $t = 0$ and a right gap at $t = 2\pi$, are omitted here for the sake of simplicity.

Finally, the *node* and *leaf node* of the tree are defined as follows:

Definition 3. *(Node) A node consists of the following elements:*

$$\text{node } i = \left\{i, s^i, R^i, [\theta^i_{\min}, \theta^i_{\max}], V^i, g^i, h^i\right\} \tag{4}$$

where i is the index of the node, s^i is the source point, R^i is the tree branch from s^0 to s^i which is constructed with the expansion of the tree, $[\theta^i_{\min}, \theta^i_{\max}]$ is the angle interval, V^i is the cell constructed from s^i and $[\theta^i_{\min}, \theta^i_{\max}]$, g^i is the length of R^i, and h^i is the cost-to-go for planning.

Definition 4. *(Leaf Node, Child) A leaf node, also referred to as a child node, is a sub-state of a node, which does not have its index i as a standalone node and its associated cell V. A child node is always defined in conjunction with its parent node.*

Without abuse of symbols, the elements in a leaf node are defined the same as those in a node, except that superscript indicates the index of the parent node, and that subscript indicates the index of the leaf node as a child.

B. Algorithm

Tree Initiation. At the beginning of the proposed algorithm, the starting location p_{start} is selected as the source point of the first node, $s^0 = p_{\text{start}}$. The angle interval is set as $[\theta^0_{\min}, \theta^0_{\max}] = [0, 2\pi]$. The tree path consists solely of s^0 itself, represented as $R^0 = [s^0]$. The cost-to-come is set as zero, $g^0 = 0$. The cost-to-go is the Euclidean heuristics, $h^0 = \| s^0 - p_{\text{goal}} \|_2$. Then, V^0 is constructed as the visibility region from s^0 in $\tilde{M}$. After V^0 is constructed, gaps are identified.

Child Construction. Taking the j-th gap of the i-th node as an example whose near obstacle and far obstacle are denoted as c^i_j and o^i_j, a child node is created as follows.

1) The source point is defined as the near obstacle: c^i_j.
2) c^i_j is appended to the tree branch: $R^i_j = [R^i; c^i_j]$.
3) The cost-to-come is set as the length of R^i_j.
4) The cost-to-go is set as the Euclidean distance.
5) The angle interval is set such that the cell of the child is seamlessly connected to the cell of its parent and the obstacle that c^i_j is adjacent to. To be precise, assuming that the boundary vertices of each obstacle are indexed in counter-clockwise order, and the previous and the next vertices are denoted as o_{prev} and o_{next}:
   a) For a left gap,
   $$\varphi_{\min} = \text{atan2}(\overrightarrow{c^i_j o^i_j}),\ \varphi_{\max} = \text{atan2}(\overrightarrow{c^i_j o_{\text{next}}}) \tag{5}$$
   b) For a right gap,
   $$\varphi_{\min} = \text{atan2}(\overrightarrow{c^i_j o_{\text{prev}}}),\ \varphi_{\max} = \text{atan2}(\overrightarrow{c^i_j o^i_j}) \tag{6}$$
6) The limits of the distance function $L(c^i_j, \cdot)$ are defined as length of the above-mentioned vectors. To be precise,
   a) For a left gap,
   $$\begin{cases} \lim\limits_{t \to \varphi^-_{\min}} L(c^i_j, t) \triangleq \| c^i_j - o^i_j \|_2 \\ \lim\limits_{t \to \varphi^+_{\max}} L(c^i_j, t) \triangleq \| c^i_j - o_{\text{next}} \|_2 \end{cases} \tag{7}$$
   b) For a right gap,
   $$\begin{cases} \lim\limits_{t \to \varphi^-_{\min}} L(c^i_j, t) \triangleq \| c^i_j - o_{\text{prev}} \|_2 \\ \lim\limits_{t \to \varphi^+_{\max}} L(c^i_j, t) \triangleq \| c^i_j - o^i_j \|_2 \end{cases} \tag{8}$$

See Fig. 3 for the illustration where Node 1 is created as the 0-th child of Node 0.

Iterative Tree Expansion. Once the first node is constructed, along with the identification of all its children, the children are added to a priority queue. The heuristic for each child follows the convention of A* [7], i.e., the sum of cost-to-come and cost-to-go. In each iteration, the algorithm selects the child with the lowest heuristic cost from the priority queue, removes it, and expands it to a new node. After that, the children of the new node are added to the priority queue again.

For easy reference, if a point is within a cell, its *tree path* is defined as follows:

Definition 5. *(Tree path) Let i be the index of a node. For a point $p \in V^i$, the tree path of p is defined as the concatenation of R^i and the straight path segment from s^i to p.*

Termination Condition. When the cell of a node contains the goal location, the tree path is identified as a resultant path, and the tree expansion continues. Once k resultant paths are obtained, the algorithm finishes and the k-SNPP problem is considered solved.

C. Generalisation to Non-selfcrossing k-SNPP

The proposed algorithm is readily extended to solve the non-selfcrossing k-SNPP problem. The only modification required is to introduce a non-selfcrossing checking for the child before adding it to the priority queue. In contrast, it should be noted that adding this is non-trivial for existing algorithms such as [5], because the node adopted for exploring the homotopy-augmented space, typically consisting of a $X - Y$

location augmented with a sequence of h-signature or winding numbers, does not memorise the path how it is reached from the starting location.

V. Optimality and Completeness

Theorem 6. *(Distance Optimality) The resultant paths obtained by the proposed algorithm are locally shortest.*

Proof. Denoting the locally shortest path in the same homotopy class of a resultant path R as R^*, the proof of $R = R^*$ is given through induction. For easy indexation, a sequence of nodes following parent-child relationship are re-labelled as Node 0, 1, 2, and so forth. Each child node is assumed to be the 0-th child of its parent, i.e.,

$$s^1 \triangleq c_0^0,\ s^2 \triangleq c_0^1,\ \cdots,\ s^{i+1} \triangleq c_0^i,\ \cdots \tag{9}$$

Base Step. When $i = 0$, V^0 is the visibility region. If $p_{\text{goal}} \in V^0$, R^* is the straight path connecting $s^0 (= p_{\text{start}})$ to p_{goal}. Hence we have $R = R^*$.

Inductive Step. Assume that the claim is correct for i, i.e., for any point in V^i, the tree path is locally shortest. Consider Node $i+1$ (See Fig. 4 for illustration):

Since R^* is homotopic to R, R^* has at least one intersection with the gap formed by c_0^i and o_0^i, which is denoted as p^i. Since any truncated part of R^* is also locally shortest, the part from p_{start} to p^i is locally shortest. Since $p^i \in V^i$, by the induction hypothesis, R^* visits s^i. Moreover, since s^i, c_0^i, and p^i are co-linear, R^* is proven to visit c_0^i, which is also s^{i+1}. This means that R^{i+1} is the beginning part of R^*. In summary, R^* consists of two parts, R^{i+1} and the locally shortest path from s^{i+1} to p_{goal} which is a straight path segment. This means $R^* = R$, i.e., the claim is correct for $i+1$.

Conclusion. By the principle of mathematical induction, the claim is correct for all nodes. Whenever the cell of a node covers the goal location, a locally shortest path is obtained. □

Theorem 7. *(Completeness) Any locally shortest path is a tree path.*

Proof. Denoting a locally shortest path as R^*, the proof is given by construction. See Fig. 4 for illustration.

To begin with, let's assume that R^* is totally within V^0. Since V^0 is the visibility region and is simply-connected, now that R^* is the locally shortest, it has to be the straight path. Therefore, R^* is the tree path.

Next, let's assume that R^* is not totally within V^0. In this case, it has an intersection with one of the gaps of Node 0. Say the gap is the one formed by c_0^0 and o_0^0 and the intersection is denoted as p^0. Since R^* is locally shortest, the truncated part from s^0 to p is also locally shortest. Since s^0, c_0^0, and p^0 are co-linear, R^* is proven to visit $c_0^0 (= s^1)$.

If the truncated part of R^* from s^1 to p_{goal} is totally within V^1, the construction is finished. Otherwise, keep doing this process along the topological route of R^* and notice that R^* is of finite length, we will eventually reach a node indexed as i where p_{goal} lies within V^i. In this case, R^* is the concatenation of R^i and the straight path from s^i to p_{goal}, i.e., a tree path, thus completing the proof. □

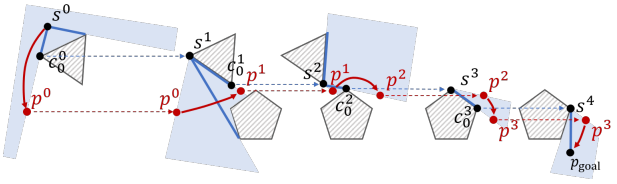


Fig. 4. Auxiliary figure for the proof of **Theorem 6** and **Theorem 7**. This illustration is collected from a branch of the tree shown in Fig. 1. The cells of nodes are drawn separately for easy distinguishment. The tree path is shown by blue edges, which are connected by dashed blue arrows. It is apparent that the locally shortest path R^* in the homotopy class of interest will have intersections with all the gaps, and the intersected points are denoted as p^0, p^1, $\cdots$. And by continuous shortening it is proven to visit c_0^0, c_1^0, $\cdots$, hence is identical to the tree path.

VI. Simulated Results

In this section, the proposed algorithm is compared against state-of-the-art homotopy-aware path searching algorithm [5] to demonstrate its computational advantage. Both the normal k-SNPP and the non-selfcrossing k-SNPP are tested.

To demonstrate the applicability of the proposed algorithm in various environments, comparisons are conducted in grid-maps but not polygonal maps. The maps are of size 150×150 grids, with 8 randomly positioned obstacles, as illustrated in Fig. 5. These selected maps pose significant challenges, as they all contain at least 5 internal obstacles, resulting in the existence of $2^5 = 32$ non-selfcrossing topological routes. The starting and goal locations are set as $(10, 10)$ and $(140, 140)$ for all experiments. The inflation of obstacles is assumed to be completed by established cost-map modules, simplifying the process of checking the validity of a robot location by reading a value from the cost matrix.

For detailed statistics, please refer to Table. I. For normal k-SNPP tasks, [5] can be directly applied, whilst before applying the proposed algorithm, we first collect the corners of the obstacle grids that are at the boundary of each internal obstacle to form the polygonal description of obstacles. The computational time for this step is listed in the "Poly" column in Table. I. Both algorithms have been theoretically proven to generate the correct k-SNPP solution. However, results show that the proposed algorithm consistently outperformed the existing state-of-the-art algorithm [5]. The paths generated by the proposed algorithm are visualised in Fig. 5. It is worth noting that the maps rarely contain large concave obstacles, which provide an advantage for searching-based algorithms. However, even in these scenarios, the existing algorithm [5] is still slower than the proposed algorithm.

For non-selfcrossing k-SNPP tasks, the proposed algorithm introduces non-selfcrossing checking for each node. This addition only incurs a minor increase in computational time due to the limited number of expanded nodes. However, this is not the case when the prior algorithm [5] is adopted. Two different strategies are tested to incorporate the non-selfcrossing constraint into [5]. Using [5](1), the path searching continues even

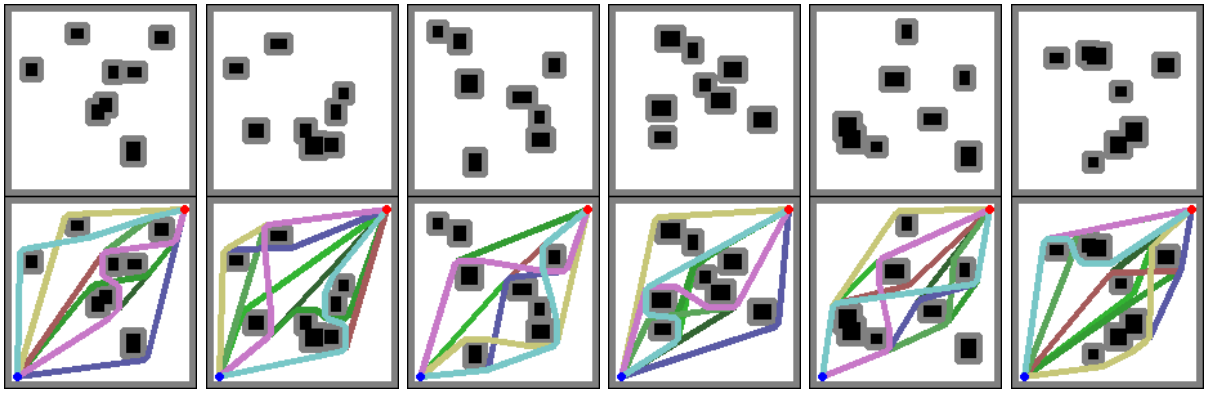


Fig. 5. Illustration of six random maps and the 9-shortest non-homotopic resultant paths generated by the proposed algorithm. The maps are labelled as from M1 to M6 in Table. I. The starting location is marked as a blue point, and the goal location is marked as a red point. Different locally shortest paths are drawn in different colours.

TABLE I
COMPUTATIONAL TIME OF ALGORITHMS FOR k-SNPP TASKS AND NON-SELFCROSSING k-SNPP TASKS

Map	Num*	Type	Alg.	Poly*	1-SNPP	2-SNPP	3-SNPP	4-SNPP	5-SNPP	6-SNPP	7-SNPP	8-SNPP	9-SNPP
M1	6	normal	Ours	0.58ms	**2.87ms**	**3.81ms**	**7.38ms**	**9.87ms**	**12.79ms**	**14.31ms**	**14.97ms**	**17.57ms**	**21.07ms**
			[5]	-	123.15ms	181.59ms	254.51ms	369.60ms	391.04ms	448.86ms	622.28ms	742.41ms	810.28ms
		NS*	Ours	0.58ms	**3.58ms**	**4.31ms**	**6.35ms**	**9.35ms**	**12.21ms**	**14.81ms**	**15.30ms**	**18.58ms**	**22.16ms**
			[5](1)	-	126.58ms	197.79ms	256.51ms	378.53ms	390.01ms	453.82ms	621.33ms	746.12ms	795.18ms
			[5](2)	-	196.98ms	251.35ms	349.36ms	499.36ms	531.60ms	605.89ms	854.58ms	1078.17ms	1139.26ms
M2	5	normal	Ours	0.67ms	**1.06ms**	**2.46ms**	**7.28ms**	**9.58ms**	**13.29ms**	**16.67ms**	**19.77ms**	**21.84ms**	**21.93ms**
			[5]	-	50.27ms	384.37ms	664.78ms	764.07ms	838.10ms	1201.37ms	1255.71ms	1364.83ms	1304.36ms
		NS*	Ours	0.67ms	**1.82ms**	**3.71ms**	**8.14ms**	**8.90ms**	**13.25ms**	**17.12ms**	**17.28ms**	**21.13ms**	**21.48ms**
			[5](1)	-	51.36ms	391.41ms	678.32ms	789.95ms	845.39ms	1219.95ms	1237.18ms	1344.22ms	1362.83ms
			[5](2)	-	77.18ms	473.55ms	869.72ms	1069.00ms	1210.10ms	1707.38ms	1736.46ms	1917.73ms	1897.84ms
M3	5	normal	Ours	0.71ms	**2.26ms**	**4.00ms**	**6.20ms**	**7.23ms**	**12.24ms**	**15.60ms**	**19.51ms**	**21.96ms**	**22.80ms**
			[5]	-	77.70ms	286.54ms	445.08ms	610.64ms	884.62ms	1022.77ms	1043.30ms	1251.21ms	1286.11ms
		NS*	Ours	0.71ms	**2.08ms**	**3.79ms**	**6.50ms**	**9.02ms**	**13.52ms**	**12.23ms**	**16.49ms**	**20.70ms**	**21.24ms**
			[5](1)	-	77.48ms	280.99ms	471.17ms	602.93ms	880.26ms	965.88ms	1095.40ms	1258.70ms	1277.80ms
			[5](2)	-	91.39ms	373.36ms	587.46ms	818.98ms	1193.04ms	1339.16ms	1469.73ms	1753.63ms	1756.30ms
M4	5	normal	Ours	0.71ms	**2.65ms**	**5.95ms**	**6.84ms**	**7.31ms**	**13.54ms**	**15.88ms**	**19.07ms**	**21.57ms**	**30.60ms**
			[5]	-	101.22ms	333.56ms	410.31ms	528.84ms	749.64ms	1094.78ms	1253.73ms	1511.88ms	1878.60ms
		NS*	Ours	0.71ms	**3.03ms**	**5.43ms**	**6.22ms**	**8.50ms**	**15.11ms**	**15.98ms**	**18.63ms**	**19.60ms**	**24.30ms**
			[5](1)	-	101.91ms	347.00ms	408.38ms	514.07ms	749.89ms	1077.52ms	1358.61ms	1496.79ms	1772.31ms
			[5](2)	-	147.16ms	476.56ms	521.92ms	700.79ms	976.21ms	1454.21ms	1762.93ms	2043.37ms	2385.09ms
M5	6	normal	Ours	0.63ms	**1.86ms**	**4.84ms**	**6.32ms**	**7.29ms**	**8.00ms**	**9.01ms**	**14.61ms**	**15.57ms**	**19.10ms**
			[5]	-	60.72ms	333.37ms	330.23ms	691.23ms	669.75ms	890.93ms	1177.98ms	1517.38ms	1589.62ms
		NS*	Ours	0.63ms	**1.33ms**	**5.29ms**	**6.57ms**	**7.32ms**	**8.64ms**	**9.25ms**	**13.56ms**	**16.56ms**	**20.38ms**
			[5](1)	-	58.68ms	333.85ms	341.99ms	687.84ms	672.01ms	847.20ms	1155.80ms	1511.09ms	1578.28ms
			[5](2)	-	88.44ms	452.01ms	454.35ms	877.61ms	856.36ms	1122.85ms	1458.79ms	1943.91ms	2050.78ms
M6	5	normal	Ours	0.63ms	**1.06ms**	**3.49ms**	**5.16ms**	**8.94ms**	**9.38ms**	**9.84ms**	**12.38ms**	**14.67ms**	**26.51ms**
			[5]	-	114.76ms	214.67ms	343.35ms	420.88ms	545.00ms	590.72ms	611.88ms	892.55ms	1248.36ms
		NS*	Ours	0.63ms	**1.14ms**	**2.97ms**	**5.17ms**	**9.68ms**	**9.61ms**	**10.34ms**	**12.86ms**	**14.62ms**	**22.07ms**
			[5](1)	-	117.98ms	217.32ms	344.81ms	417.16ms	551.20ms	582.59ms	603.92ms	882.53ms	1241.73ms
			[5](2)	-	141.42ms	267.70ms	452.92ms	578.14ms	766.61ms	842.26ms	874.48ms	1290.37ms	1761.24ms

* Num: Number of internal obstacles. NS: Non-Selfcrossing. Poly: Computational time for obstacle polygonalisation.
All results have been averaged over 20 runs on a laptop with I7-7700HQ CPU for a fair evaluation.

if self-crossing occurs, and the non-selfcrossing checking is only performed for resultant paths. This saves time on checking non-selfcrossing property, but results in more expanded nodes. Results show that its has little computational difference compared to its original form [5] (allowing selfcrossing). This is because the 9-th shortest locally shortest path is not much longer than the 1-st one, leading to a still manageable number of extra nodes with selfcrossing topological routes being constructed before the algorithm finishes. On the other hand, the second approach [5](2), verifies every node at the moment of its creation. Whilst this guarantees a non-selfcrossing searching process, the computational burden is significantly increased. To sum up, both strategies are notably slower than the proposed algorithm.

Besides the comparative study detailed above, please also refer to the supplementary video for a detailed case study, which is omitted due to the space constraint of this paper.

VII. CONCLUSION

The algorithm proposed in this paper solved the 2D k-SNPP task by constructing a topological tree. Rooting at the starting location, the topological tree expands iteratively, created visibility-based sub-regions as the cell of new nodes. The tree has been proven to keep both topological completeness and geometric optimality. Extensive simulated comparisons are provided to validate the computational advantage of the proposed algorithm against state-of-the-art algorithms. These have been supplemented by a video and an open-sourced C++ implementation for the benefit of the community.

REFERENCES

[1] A. S. Kimmel and K. Bekris, "Minimizing conflicts between moving agents over a set of non-homotopic paths through regret minimization," in *Workshops at the 2013 AAAI Conference on Artificial Intelligence*, 2013.

[2] M. Werner and S. Feld, "Homotopy and alternative routes in indoor navigation scenarios," in *Proceedings of the 2014 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pp. 230–238, IEEE, 2014.

[3] C. Rösmann, F. Hoffmann, and T. Bertram, "Integrated online trajectory planning and optimization in distinctive topologies," *Robotics and Autonomous Systems*, vol. 88, pp. 142–153, 2017.

[4] S. Kim, S. Bhattacharya, R. Ghrist, and V. Kumar, "Topological exploration of unknown and partially known environments," in *Proceedings of the 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3851–3858, IEEE, 2013.

[5] S. Bhattacharya, M. Likhachev, and V. Kumar, "Topological constraints in search-based robot path planning," *Autonomous Robots*, vol. 33, no. 3, pp. 273–290, 2012.

[6] F. T. Pokorny, D. Kragic, L. E. Kavraki, and K. Goldberg, "High-dimensional winding-augmented motion planning with 2d topological task projections and persistent homology," in *Proceedings of the 2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 24–31, IEEE, 2016.

[7] 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.

[8] 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.

[9] G. F. Simmons, "Introduction to topology and modern analysis," *The American Mathematical Monthly*, vol. 71, no. 4, 1964.

[10] L. Jaillet and T. Simeon, "Path deformation roadmaps: Compact graphs with useful cycles for motion planning," *The International Journal of Robotics Research*, vol. 27, no. 11-12, pp. 1175–1188, 2008.

[11] T. Siméon, J.-P. Laumond, and C. Nissoux, "Visibility-based probabilistic roadmaps for motion planning," *Advanced Robotics*, vol. 14, no. 6, pp. 477–493, 2000.

[12] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006. Available at http://planning.cs.uiuc.edu/.

[13] 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.

[14] Y. Liu and S. Arimoto, "Path planning using a tangent graph for mobile robots among polygonal and curved obstacles," *The International Journal of Robotics Research*, vol. 11, no. 4, pp. 376–382, 1992.

[15] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[16] E. Masehian and M. R. Amin-Naseri, "A voronoi diagram-visibility graph-potential field compound algorithm for robot path planning," *Journal of Robotic Systems*, vol. 21, no. 6, pp. 275–300, 2004.

[17] F. Aurenhammer, "Voronoi diagrams: a survey of a fundamental geometric data structure," *ACM Computing Surveys (CSUR)*, vol. 23, no. 3, pp. 345–405, 1991.

[18] J. Pettre, J.-P. Laumond, and D. Thalmann, "A navigation graph for real-time crowd animation on multilayered and uneven terrain," in *First International Workshop on Crowd Simulation*, vol. 43, p. 194, New York: Pergamon Press, 2005.

[19] B. Banerjee and B. Chandrasekaran, "A framework for planning multiple paths in free space," in *Proceedings of 25th Army Science Conference, Orlando, FL*, vol. 5, 2006.

[20] J. Wang and M. Q.-H. Meng, "Optimal path planning using generalized voronoi graph and multiple potential functions," *IEEE Transactions on Industrial Electronics*, vol. 67, no. 12, pp. 10621–10630, 2020.

[21] H. El Gindy and D. Avis, "A linear algorithm for computing the visibility polygon from a point," *Journal of Algorithms*, vol. 2, no. 2, pp. 186–197, 1981.