An efficient tangent graph-based path planning for multiple topologically distinctive paths

Zhuo Yao, Wei Wang∗

***Abstract*—Classic conventional topologically distinctive path planning relies on indicators related to the number of isolated obstacles to determine whether two paths belong to the same topology. This reliance often hinders real-time processing (<1s) in maps with numerous isolated obstacles. Furthermore, most existing topologically distinctive path planning methods require multiple graph searches to obtain multiple paths, resulting in non-real-time performance when searching for hundreds of topologically distinctive paths. In light of these challenges, we propose an efficient path planning approach based on tangent graphs to discover multiple topologically distinctive paths. Our method diverges from existing algorithms by eliminating the need to distinguish whether two paths belong to the same topology. Instead, it generates multiple topologically distinctive paths based on the locally shortest property of tangents. Additionally, we introduce an expansion reduction mechanism for the queue during graph search, thereby preventing the exponential expansion of the queue size. To illustrate the advantages of our method, we conducted a comparative analysis with various typical algorithms using a widely recognized public dataset. The results indicate that, on average, our method generates 400 topologically distinctive paths within 250 milliseconds. This outcome underscores a significant enhancement in efficiency compared to existing methods.**

***Index Terms*—distinctive topology path, tangent graph, trajectory optimization, path planning.**

I. Introduction

The following sections of this article are organized as follows: Section II introduces relevant studies on distinctive topology path planning. Section III provides a detailed description of the theoretical basis and key processes involved in our method. In Section IV, we delve into the details of constructing the tangent graph on specified maps. Additionally, we present a comprehensive comparison between our method and several typical methods in terms of time cost, success rate, and path length. This section also includes information about how our method performs as the number of paths increases and the scale of the map increases. Finally, in Section V, we discuss the results obtained and potential drawbacks of the proposed method.

This article contributes the following:

1) A new theory: "For 2D Euclidean space, each locally shortest path corresponds to a topologically unique path," along with its proof.
2) A topologically distinctive path planning method based on the locally shortest property of tangents.
3) The introduction of an expansion reduction mechanism to mitigate the exponential growth of the queue size in path planning that utilizes breadth-first search.

Corresponding by Wei Wang: wangweilab@buaa.edu.cn. This work was supported by the National Key Research and Development Program of China under grant number 2020YFB1313600.

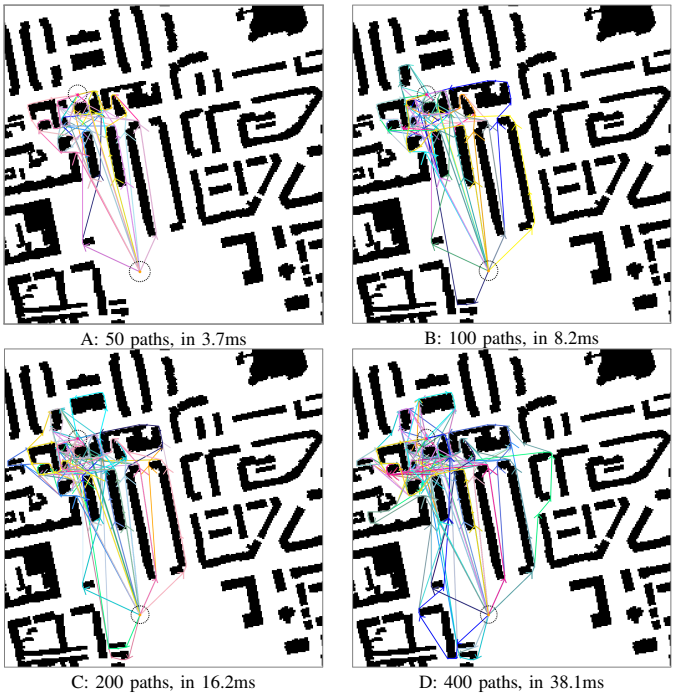


Fig. 1. These figures display multiple paths, with some of them partially overlapping, between the same start and target points, all determined by our method. The map employed is a 256x256 grid map ("Berlin_1_256" from the mentioned public map dataset). The start and target points, highlighted in pink and yellow, respectively, have coordinates (59, 72) and (109, 214), located at the center of circles. These experiments were conducted on a standard laptop running the Ubuntu operating system, equipped with a 3.2GHz CPU and 16GB of memory. Further details can be found in the Results section.

II. Methodology

A. Definitions

In this section, we give a brief introduction about concept we used in

1) Grid space: Let $\mathcal{S}_{\mathcal{N}}$ denote a finite $\mathcal{N}$-dimensional integer Euclidean space, where the size of the space is defined as $\mathcal{D} = d_1, d_2, ..., d_i, ..., d_{\mathcal{N}}$ and $d_i \in \mathbb{N}$. The coordinate of an element g in this space is defined as a vector $(x_1, x_2, ..., x_i, ..., x_{\mathcal{N}})$, where $x_i \in ([0, d_i) \cap \mathbb{N})$.

In the following, we manily focus on $\mathcal{S}_2$.

2) Distance metrics: The distance between two grids g_1 and g_2 is denoted as $\Omega(g_1 \rightarrow g_2)$, which is defined as the Euclidean distance between the two grids.

3) Grid states: There are only two possible states for a grid element in $\mathcal{S}_\mathcal{N}$: passable or unpassable. The set of all passable grids in $\mathcal{S}_\mathcal{N}$ is denoted as $\mathcal{F} \to \mathcal{S}_\mathcal{N}$, while the set of all unpassable grids is denoted as $\mathcal{O} \to \mathcal{S}_\mathcal{N}$. Therefore, $(\mathcal{F} \to \mathcal{C}_\mathcal{N}) \cup (\mathcal{O} \to \mathcal{S}_\mathcal{N}\mathcal{S}_\mathcal{N}, (\mathcal{F} \to \mathcal{S}_\mathcal{N} \cap (\mathcal{O} \to \mathcal{S}_\mathcal{N} = \emptyset$.

4) Path: A path p is defined as a sequence of waypoints. For a path, its first waypoint is the start, and if it is finished, its last waypoint is the target. All intermediate waypoints are nodes of the roadmap graph. A set of paths is defined as P. A finished path or a set of finished paths is denoted as p_f and P_f, respectively.

5) ϵ-neighboring region of path [11]: For a path p and a small positive real ϵ, a region $W = \{w : w \in \mathcal{S}_\mathcal{N}$ is an ϵ-neighboring region of path p if $\Omega(w, c) < \epsilon$ for any $c \in p$. Here, w and c are points on the neighboring region of the path and the path itself, respectively.

6) Locally shortest path: Building upon the ϵ-neighboring region, a path p is considered a locally shortest path if it is impossible to find a shorter collision-free path than p within any small ϵ-neighboring region.

7) Smooth transformation: A transformation that transforms path p_1 to p_2 is considered a smooth transformation if, for any transitional path (denoted as p_t) during the transformation, there exist both an earlier transitional path (p_{t-1}) and a later transitional path (p_{t+1}) that are in any small ϵ-neighboring region of p_t, and all transitional paths are collision-free. During a smooth transformation, the length of the transitional path changes continuously.

8) Topology about path: Two trajectories (or paths), assuming p_1 and p_2, are said to belong to the same homotopy class if one can be smoothly transformed into the other without intersecting obstacles; otherwise, they belong to different homotopy classes (i.e., they are two topologically distinctive paths). This definition is quoted from [1].

B. Theory basis

In this section, we aim to establish the relationship between topologically distinctive paths and locally shortest paths. It is essential to note that all paths discussed herein share the same start and target points. It is noteworthy that all paths discussed in this section do not contain loops (i.e., they do not intersect with themselves), which is a common constraint in topologically distinctive path planning.

Lemma 1: For $\mathcal{S}_2$, there is no closed curve that encircles no obstacles, and any part of it is locally shortest.

Proof 1:

In $\mathcal{S}_2$, for a closed curve C_1 that encircles no obstacle, we can always construct a shorter inner curve C_2 which is in any small ϵ-neighboring region of C_1. However, if any part of C_1 is locally shortest, C_2 should be longer than C_1, resulting in a contradiction. Therefore, for a closed curve that encircles no obstacle, not every part of it is locally shortest. An example is shown in Fig. 2(A).

In contrast, for a closed curve that encircles an obstacle, every part of it may be locally shortest. An example is a closed curve that consists of continuous tangents on the boundary of an obstacle, which cannot find a shorter curve in its neighboring region. An example of such a closed curve is shown in Fig. 2(B).

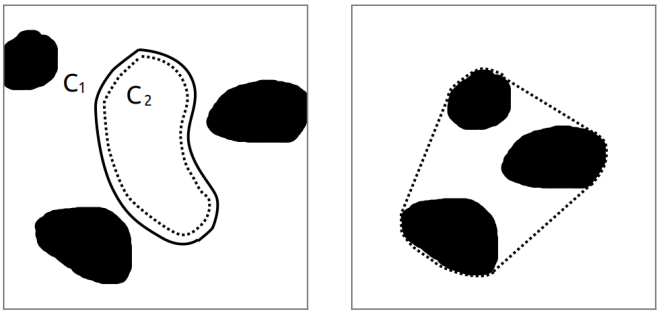


Fig. 2. Figure A shows an example of a closed curve (C_1, solid line) that encircles no obstacles and an inner curve (C_2, dotted line) in its neighboring region. Figure B shows a closed curve (dotted line) that consists of tangents, which encircles three obstacles. In this and all following figures, the white area represents passable grid states, and the black area represents impassable grid states.

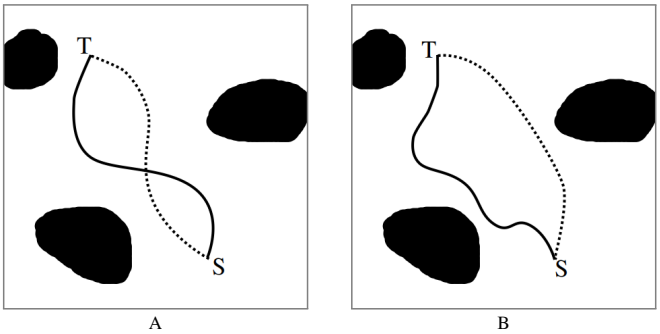


Fig. 3. For the two figures, solid lines and dotted lines represent different paths, "T" and "S" represent the common target and start of those paths, the same applies hereinafter. Figure A shows an example of two homotopic paths that intersect with each other, while Fig B shows an example of two paths that have no intersection with each other. As shown in Figure A and Figure B, two homotopic paths can always form at least one closed curve.

Theorem 1: For $\mathcal{S}_2$, any two locally shortest paths that share the same start and target cannot belong to the same homotopy class.

Proof 2: For $\mathcal{S}_2$, assuming two locally shortest paths (p_1 and p_2) are in the same homotopy class, considering they share the same start and target, there exists at least one closed curve that encircles no obstacles formed by a part of p_1 and p_2, regardless of whether p_1 and p_2 intersect with each other. Related examples are shown in Fig. 3. However, this contradicts the statement that "for $\mathcal{C}_2$, there exists no closed curve that encircles no obstacles, and any part of it is locally shortest." Therefore, any two locally shortest paths that share the same start and target cannot belong to the same homotopy class.

Theorem 2: For $\mathcal{S}_3$, two locally shortest paths that share the same start and target may belong to the same homotopy class.

Proof 3: Here we can illustrate an example for $\mathcal{S}_3$, as shown in Fig. 4. Both of the two paths are locally shortest, and one can be transformed into another without intersecting with obstacles. So, for $\mathcal{S}_3$, two locally shortest paths that share the same start and target may belong to the same homotopy class.

Building upon this theoretical foundation, for $\mathcal{S}_2$, if K locally shortest paths exist, we consequently obtain K topo-

Fig. 4. This figure shows an hourglass-shaped obstacle, and there are two locally shortest paths that share the same start and target ("S" and "T"). The straight line connecting the start and target is blocked by the narrow pipes in the center of the hourglass. One path bypasses the recess side of the "hourglass," while the other path bypasses the recess on the opposing side, and both of them are locally shortest.

logically distinctive paths. However, for $\mathcal{S}_3$, a locally shortest path may not correspond to a unique homotopy class.

Theorem 3: For $\mathcal{S}_2$, locally shortest paths can be formed by tangents connecting the start and target points, as proven in [11, 18].

In summary, for $\mathcal{S}_2$, utilizing the tangent graph allows the generation of multiple topologically distinctive paths, but this does not work for $\mathcal{S}_3$. Meanwhile, the time cost of tangent graph detection remains insensitive to the scale increase of maps, as the tangent graph is solely determined by the geometric elements of the maps. These advantages will be demonstrated and discussed in Section IV.

C. Algorithm

In this section, we focus on the implementation of using the tangent graph to obtain multiple topologically distinctive paths. Building upon the theoretical foundation, we can extract multiple topologically distinctive paths from the tangent graph. Our focus in this section is on the efficient utilization of the tangent graph to obtain multiple topologically distinctive paths. Traditionally, two well-known methods for determining paths between two nodes in a graph are Breadth-First Search (BFS) and Best-First Search. Given that existing methods (refer to Section II for more details) often require multiple searches to obtain multiple paths, our objective is to find multiple paths in a single graph search. Consequently, we choose to implement our method based on Breadth-First Search.

1) Construction of tangent graph: Researchers have proposed several methods [11, 18, 12] to generate tangent graphs from grid maps. Here, we choose ENL-SVG [12] to generate tangent graphs because it uses line-of-sight scans to construct the graph. Line-of-sight scans utilize a row-by-row scan to avoid point-to-point line-of-sight checks, which is very effective in maps with dense obstacles. However, for large grid maps with sparse obstacles, we replace the line-of-sight scan with Jump Over Block (JOB) [17], which accelerates LOS checks significantly under those maps.

Although the graph generated by ENL-SVG [12] was called a "visibility graph" in [12], they use the taut condition to ensure local shortest paths, which is different from an ordinary visibility graph that only concerns visibility between nodes (i.e., waypoints). The taut condition imposes restrictions on any two connecting edges (i.e., three continuous waypoints) in the path. Informally, the taut condition is two connecting edges

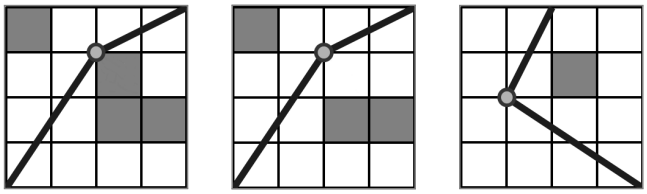


Fig. 5. These figures show three examples of the taut condition, where (A) is taut, while (B) and (C) are not. These figures are quoted from [12].

Fig. 6. These figures show examples of two possible states of loops in a path. Figure A shows loops caused by the revisitation of waypoints (p_2 and p_5), while Figure B shows loops caused by line intersection ($p_2 \rightarrow p_3$ and $p_4 \rightarrow p_5$). The order of waypoints is shown in increasing numbers. For example, in Figure A, p_1 is the first waypoint, and $p_{2,5}$ means the second and the fifth waypoints are overlapped.

which, when treated as a string, cannot be made "tighter" by pulling on its ends. Some examples of the taut condition are shown in Fig. 5. A path is locally shortest only when any two continuous edges in it meet the taut condition.

The construction of the tangent graph is not the main purpose or contribution of this article. More details about the construction of the tangent graph can be found in [12].

2) Loop avoidance: As a fundamental principle in topologically distinctive paths [3], path planning with distinctive topology strictly avoids the presence of loops. Loops in this context can take two forms: 1) overlap waypoint, where a waypoint is revisited within a single path; 2) line intersection, where two lines of the path intersect with each other, as shown in Fig. 6.

So, there are two steps to check whether a path has loops when adding a new waypoint: the first step is to check whether the new node has already occurred in the path; the second step is to check whether the newly added edge (the connection of the new node and the last node of the path) intersects with other edges in the path. If there is no such revisitation or intersection, the addition of the new node passes the loop avoidance check.

Notably, as the number of waypoints in a path increases, the time required for loop detection also grows. In detail, when checking whether adding a waypoint to a path with n waypoints, n revisitation checks and $n-1$ line intersection checks are performed. If a finished path has n waypoints, it takes $\frac{(n-1)n}{2}$ revisitation checks and $\frac{(n-2)(n-1)}{2}$ line intersection checks in total before it reaches the target. We analyzed the time cost component of path search via Perf[1] and found that loop avoidance takes 50% to 80% of the total time cost on average.

[1]https://www.jetbrains.com/help/clion/cpu-profiler.html

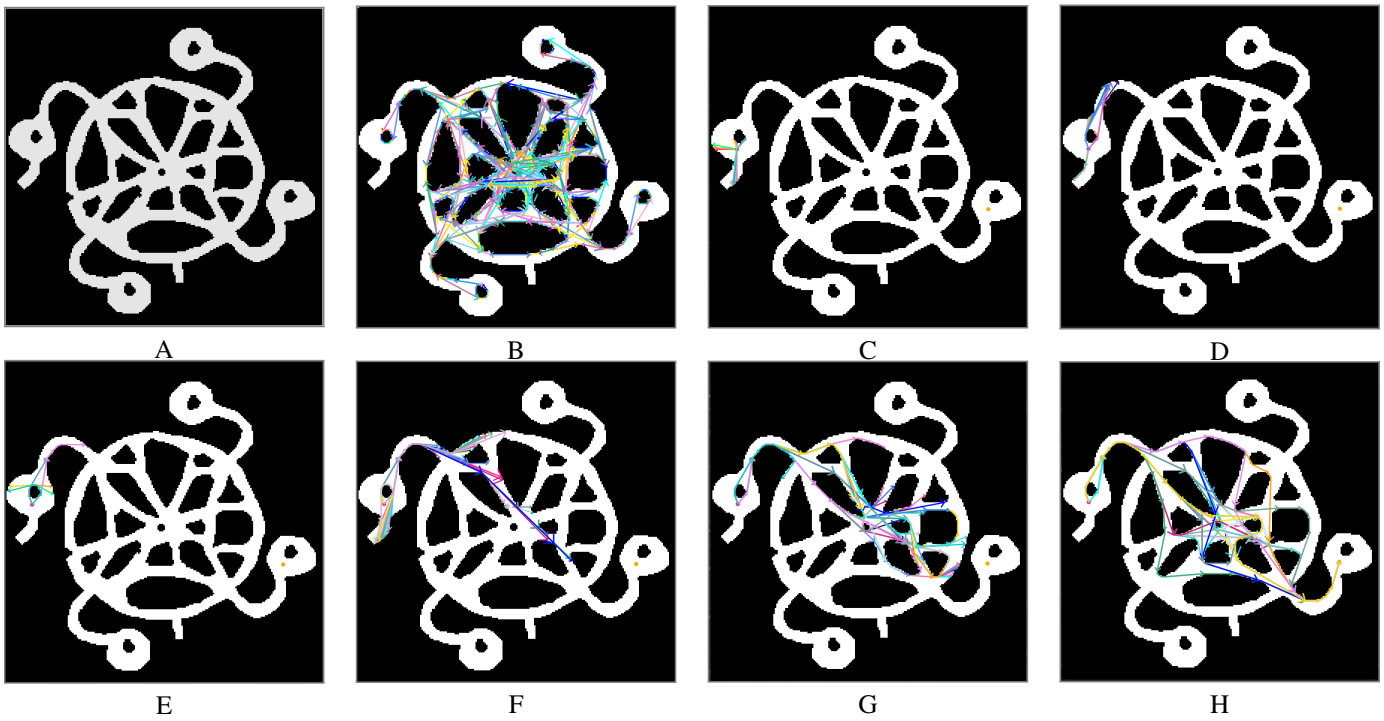


Fig. 7. These figures show the whole process of our algorithm. Figure A shows the raw grid map; Figure B shows the tangent graph of the grid map, with colored arrows representing all edges and their orientation. For a better visual effect, only edges that form loops are shown in different colors; Figure C shows the newly added edges connecting the start (point in the left-center of the map); Figures D to H show the Breadth-First search that starts from the newly added edges and ends when 100 paths are found. The map is AR0205SR with a size of 212*214, introduced from [16]. The start and target are (18, 95) and (186, 135) respectively, with the origin of the map being the upper-left corner.

Algorithm 1: Topology-aware Breadth First Search

Input: $g_s, g_t, G(V, E), K$
Output: P

```
P = {g_s}, P' = ∅, P_f = ∅;
add g_s and g_t to G;
while find no K paths and P is not empty do
    P' = ∅;
    for each unfinished path p in P do
        for each nodes v of G that connect to last nodes of p do
            p' = add v to p;
            if p' have no loops and p' meet taut condition then
                if p' reach g_t then
                    add p' to P';
                else
                    add p' to P_f;
    P = P' ;
return P_f;
```

3) Path search: Our method comprises two key components: tangent graph construction and path search. Tangent graph detection from grid maps is a well-explored area, and there exist established methods for this purpose [11, 18, 12]. Considering the efficiency of line-of-sight scans, we adopt ENL-SVG[12] to construct our tangent graph. Consequently, our primary focus in this manuscript lies in the efficient search for paths.

TABLE I
TABLE OF NOTATIONS

Name	Meanning	Name	Meanning
g_s	start point	g_t	target point
P, P'	set of unfinished path	P_f	set of finished path
$G(V, E)$	tangent graph	v	a node in tangent graph
p, p'	path	P_B	set of unifinished path (backup queue)

We have modified the classical Breadth-First Search (BFS) algorithm used in graph search-based path planning to search for multiple topologically distinctive paths. In detail, there are three major modifications:

1. Using a tangent graph rather than other kinds of graphs;
2. Exiting when finding the required number of paths (i.e., K paths) rather than exiting when finding the first finished path;
3. Adding the taut condition[12] and loop avoidance as constraints during expansion to ensure locally shortest paths and reduce the search space, as outlined in Algorithm 1. It is noteworthy that if there is no K path, the set P will be empty after find all possible solutions, then the main loop will exit when the set P is empty.

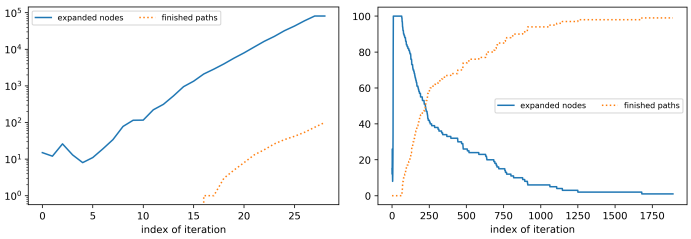
The notation used in the pseudocode is detailed in Table I, assuming the required number of paths is denoted as K. An illustrative example of path search is provided in Fig. 7.

For some scenarios that require considering the volume of the moving agent, three steps are required: 1, inflate all

Algorithm 2: Topology-aware Breadth First Search with expansion reduction

Input: $g_s, g_t, G(V, E), K$
Output: P

$P = \{g_s\}, P' = \emptyset, P_f = \emptyset, P_B = \emptyset$;
add g_s and g_t to G;
while *P_f have no K paths and P is not empty* **do**
  $P' = \emptyset$;
  for *each unfinished path p in P* **do**
    for *each nodes v of G that connect to last nodes of p* **do**
      p' = add v to p;
      if *p' have no loops and p' meet taut condition* **then**
        if *p' reach g_t* **then**
          add p' to P';
        else
          add p' to P_f;
  $P = \emptyset$;
  sort P' in increasing heurstic value;
  move first K path of P' to P and move remaining path in P' to P_B;
  sort P_B in increasing heurstic value;
  if *P have no K paths* **then**
    while *P_B is not empty and P have no K path* **do**
      move path with minimum heurstic value in P_B to P;
return P_f;



A: no expansion reduction B: using expansion reduction

Fig. 8. These two figures present the number of nodes expanded and the number of paths finished (i.e., reaching the target) during each iteration of path search when no expansion reduction is applied (Algorithm 1, Figure A) and when using expansion reduction (Algorithm 2, Figure B). It is noteworthy that Figure A uses a logarithmic coordinate system while Figure B does not.

obstacles of the grid map with the length of the maximum inscribed circle radius of the shape of the moving object; 2, run the topology-aware path search; 3, check whether each path is safe with consideration of the volume of the moving object and only keep all safe paths.

4) Expansion reduction: One drawback of Algorithm 1 is the exponential growth in the size of unfinished paths as the number of expansions increases. To address this issue, we propose an expansion reduction technique to mitigate the exponential growth during iterations of BFS. This technique involves splitting the queue into two: a priority queue and a backup queue.

In each iteration of BFS, we select the first K paths with the minimal heuristic value (calculated in the same manner as A*) and store them in the priority queue as candidates for expansion. The remaining unfinished paths are cached in a backup queue, sorted by their heuristic values.

If there are fewer than K unfinished paths in the priority queue, we extract unfinished paths from the backup queue and insert them into the priority queue until there are K elements in the priority queue or the backup queue is empty, as shown in Algorithm 2. The relevant code related to the expansion reduction has been highlighted.

Essentially, this approach changes the order of expansion of unfinished paths, prioritizing paths that are closer to the target (i.e., have lower heuristic values). As a result, the path search concludes sooner compared to the scenario without expansion reduction. Importantly, the expansion reduction does not compromise completeness or path's locally shortest property. Algorithm 2 outlines BFS with expansion reduction, specifying its differences compared to Algorithm 1.

An illustrative example highlighting the benefits of introducing expansion reduction is presented in Fig. 8. All configurations for path search are the same as the case in Fig. 7. The only difference is whether to introduce expansion reduction.

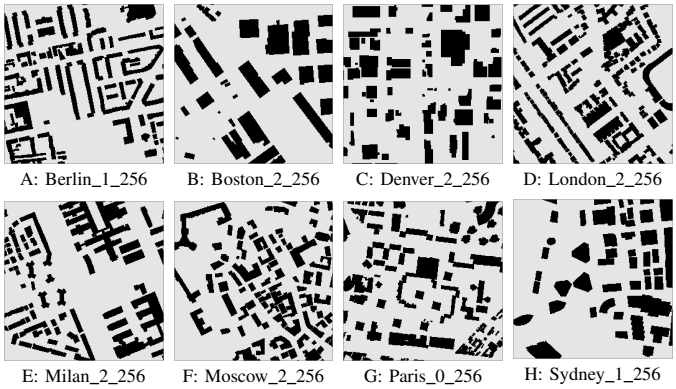
When no expansion reduction is applied, the total time cost of path search is 573.9 ms, and the number of nodes expanded during each iteration grows exponentially, stopping when the required number ($K = 100$) of paths reach the target. A total of 374662 node expansions are performed during path search, and most of them are irrelevant to finished paths.

However, when using expansion reduction, the total time cost of path search is only 133.2 ms, and the number of nodes expanded stops growing when it reaches 100, then decreases as more and more paths reach the target. Thus, only 36385 expansions are performed, and most of them are relevant to finished paths.

Although using expansion reduction takes more iterations than no expansion reduction, there are only a few nodes expanded during each iteration, as shown in Fig. 8. In summary, the introduction of expansion reduction significantly reduces the time cost of path search by reducing the number of node expansions.

III. Results

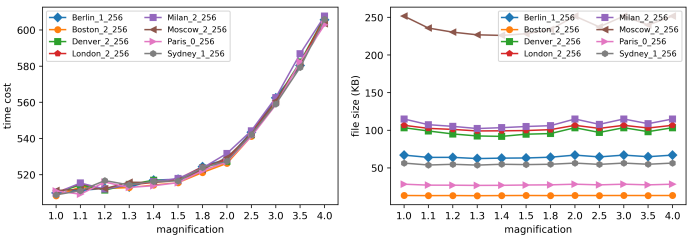
In this section, we delve into a detailed examination of our algorithm's performance, comparing it with other distinctive topology path planning methods. The first subsection provides insights into the construction of the tangent graph. The second section is about comparison with other algorithms. To ensure a fair comparison, we utilize the implementations provided by the respective authors. There are other algorithms that are worth comparing, such as TARRT*[**19**] and Kuderer's algorithm[9], but we found no source code for them. Thus our method is compared with HA*, HTheta*, and RHCF, focusing on the time cost as the required number of paths and the scale



A: Berlin_1_256 B: Boston_2_256 C: Denver_2_256 D: London_2_256

E: Milan_2_256 F: Moscow_2_256 G: Paris_0_256 H: Sydney_1_256

Fig. 9. These figures showcase eight grid maps from [16] utilized in the comparison with other algorithms. The size of each map is 256×256. In these maps of city streets, grey areas represent passable areas, while black areas represent unpassable areas.



A: time cost comparison B: time cost comparison

Fig. 10. These two figures present comprehensive data pertaining to the tangent graph detection process of our method. The data includes information on time cost and file size to save the graph, encompassing various magnifications of the map. Figure 1 details the time cost across different magnifications, while Figure 2 provides insights into the corresponding file sizes.

of the map increases. Path length comparisons for each method are also conducted, revealing [insert findings here]. HA* and HTheta* are H-signature-based algorithms [2], while RHCF is a Voronoi graph-based algorithm [3]; more details about these methods can be found in Section II. To foster further research within the community, we have made the source code of our proposed algorithm publicly accessible[4].

The experiments are conducted using a well-known grid map dataset [16] [5]. City maps from the dataset are selected for their high obstacle density, allowing the generation of hundreds of topologically distinctive paths, providing ample space for each method to demonstrate its capability. For each map, 100 start and target combinations are randomly sampled as inputs for path planning. To study how method performance changes with an increasing map scale, raw maps are magnified to create larger maps. The magnification ranges from 1.0 to 4.0 (including 1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.8, 2.0, 2.5, 3.0, 3.5, 4.0 specifically), resulting in map scales from 256×256 to 1024×1024. Experiments were conducted on a laptop running Ubuntu 20.04, equipped with a Ryzen 7 5800h (3.2GHz) CPU and 16GB of memory.

To showcase the methods' capability to find a substantial number of topologically distinctive paths, we set all methods to find paths ranging from 10 to 400 (including 10, 20, 30, 40, 50, 80, 120, 160, 200, 250, 300, 350, 400 specifically). For HA* and HTheta*, their success rate drops to less than 10% when attempting to find 200 topologically distinctive paths, so we set the maximum number of required paths for them to 200 to reduce the overall time cost of the experiment.

A. Construction of tangent graph

In this section, we elaborate on the construction of the tangent graph for the aforementioned eight maps across various magnifications. The associated data, including time cost and file size, is summarized in Fig 10.

[2]https://github.com/subh83/DOSL
[3]https://github.com/srl-freiburg/srl_rhcf_planner
[4]https://joeyao-bit.github.io/posts/2023/09/07/
[5]https://movingai.com/benchmarks/grids.html

The construction of the tangent graph for nearly all maps is completed in approximately 0.5 seconds, indicating nearly real-time updates. Notably, the time cost exhibits a gradual increase as the scale magnifies. Specifically, when the map's scale increases fourfold, the time cost only experiences a 20% increment.

The file size required to store the tangent graph for these maps ranges from 10KB to 300KB, demonstrating a manageable size for ordinary computing platforms. Importantly, the size of the tangent graph remains stable as the map scale increases. This observation suggests that the tangent graph exhibits insensitivity to variations in map resolution, making it well-suited for real-world maps with high resolutions.

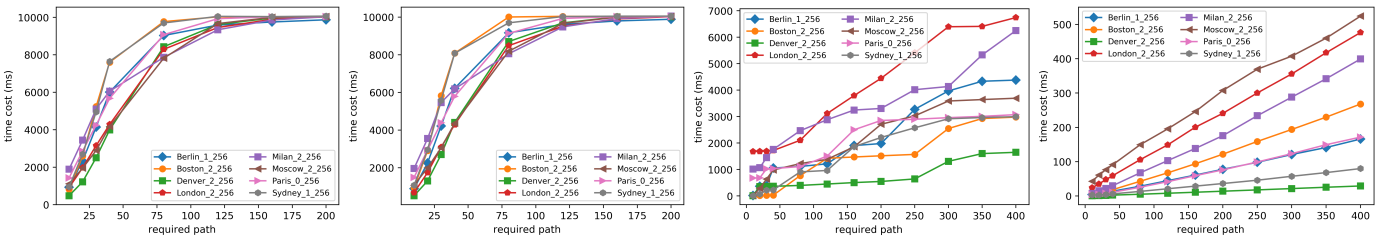
B. Comparison with other methods

In this section, our focus centers on the comparative analysis of our method with other prominent approaches, namely HA*, HTheta*, and RHCF.

In practical applications, topologically distinctive path planning is often constrained by an upper limit on time costs to avoid exceeding expected time constraints. When a method fails to find all solutions within the specified time bound, it returns the solutions it has discovered up to that point. Therefore, in this section, we impose a time upper bound of 10 seconds and employ the success rate – the ability to find all solutions within the time constraint – as an indicative measure for evaluating the performance of the methods. Essentially, the success rate is determined by the time cost incurred.

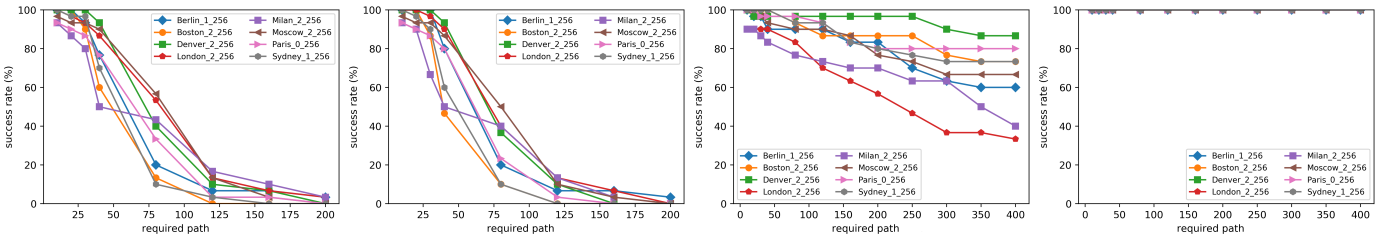
Furthermore, we conduct a comparative assessment of all methods with respect to path length – a crucial metric in path planning. Recognizing that map scale significantly influences the performance of path planning algorithms, we conduct tests on the four aforementioned methods using maps with identical content but scales ranging from 256×256 to 1024×1024.

1) Time cost and success rate: Initially, we subjected all four methods to testing under the specified eight maps, utilizing them to search varying numbers of topologically distinctive paths (ranging from 10 to 400). The mean time cost and success rate were recorded for each map, as depicted in Fig. 11 and Fig. 12 respectively. The amalgamated results, showcasing the interplay between time cost and success rate, are illustrated in Fig. 13.



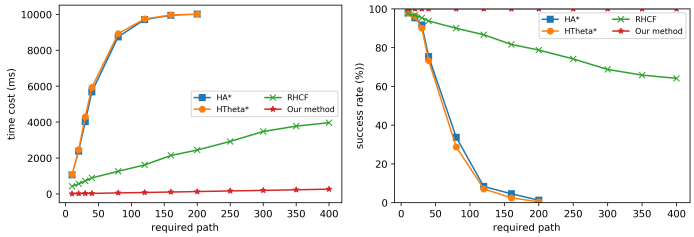
A: HA* B: HTheta* C: RHCF D: Our method

Fig. 11. These figures illustrate the mean time cost for finding the required number of paths for each map using all four methods. The magnification for all maps is set to 1.0.



A: HA* B: HTheta* C: RHCF D: Our method

Fig. 12. These figures illustrate the mean success rate for finding the required number of paths for each map using all four methods. The magnification for all maps is set to 1.0.



A: time cost comparison B: success rate comparison

Fig. 13. These figures show the comparison of the success rates and time costs of the four methods across all eight maps, with a consistent magnification set to 1.0.

As illustrated in Fig. 11 and Fig. 12, our method achieves a mean time cost of approximately 500ms to find 400 paths, boasting a 100% success rate. In contrast, RHCF takes around 7s to find 400 paths on average, with its success rate varying from 80% in the best case to 40% in the worst case. For HA* and HTheta*, their mean time cost approaches the time upper bound (10s) when attempting to find more than 80 paths. Moreover, their success rate declines as the mean time cost increases, dropping to 50% after attempting to find more than 40 paths. Notably, our method exhibits a slower increase in time cost compared to other methods, as it circumvents multiple graph searches for multiple paths, obtaining all required paths in a single search. With an average time cost of only 500ms in the worst case, our method consistently achieves a 100% success rate in finding the required number of paths.

This section underscores our method's substantial advantage over other methods in terms of time cost and success rate.

2) Path length: Path length, a critical attribute in path planning, is the focus of comparison among all four methods in this section. However, it's worth noting that methods like HA* and HTheta*, which incur longer computation times for extended paths, may prioritize finding shorter paths and consequently may not reach the required number of paths. To ensure fairness in our comparison, we exclusively examine path lengths when finding 10 topologically distinctive paths, a scenario where all methods achieve a success rate exceeding 90%, as depicted in Fig. 16.

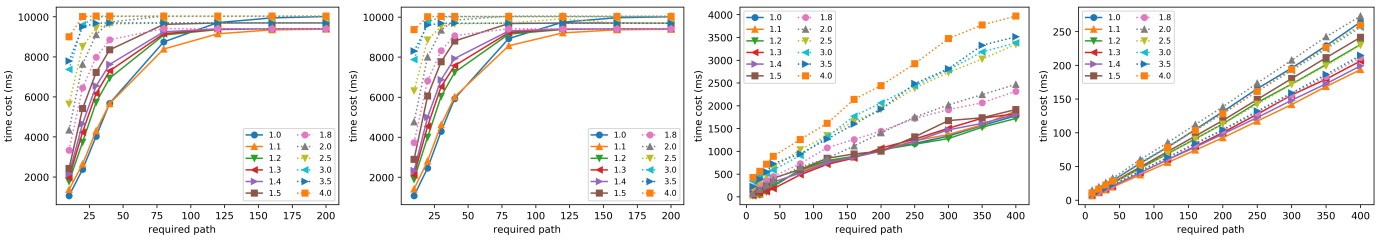
Notably, RHCF exhibits the longest paths, given its reliance on the Voronoi Graph. Both HTheta* and HA* yield shorter paths than RHCF, as they are rooted in A*-based shortest path finding. Interestingly, HTheta* and our method achieve shorter paths than HA*, leveraging tangent graph-based path planning and the any-angle shortest path planning characteristic of HTheta*, whereas A* relies on four-neighbor or eight-neighbor expansion.

3) Map scale and time cost: In this section, our focus centers on exploring the relationship between mean time cost and map scale. We conducted tests for all four methods across eight maps with varying magnifications, recording the mean time cost and success rate for different required numbers of paths, as depicted in Fig. 14 and Fig. 15.

As evident in these figures, HA* and HTheta*'s time cost increases significantly, and their success rate decreases as the map magnification scales from 1.0 to 4.0. This decrease is more pronounced as the larger map size requires more node expansions and H-signature calculations.

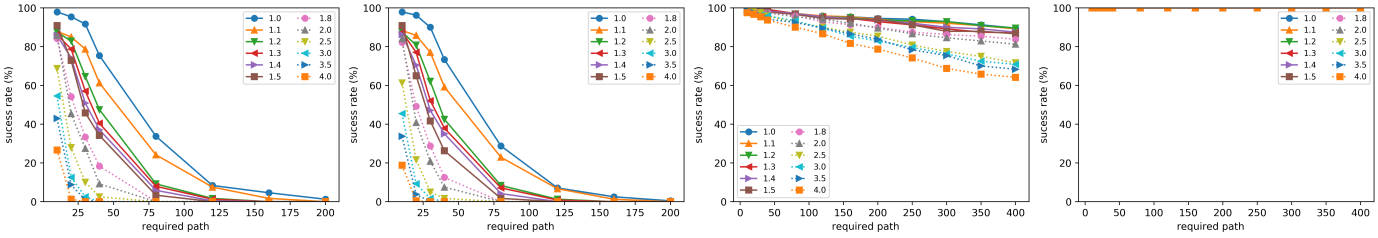
RHCF exhibits a similar trend, although in a milder manner. The size of the Voronoi graph remains relatively unchanged, but determining connections from start/target to the graph takes more time with larger maps.

In contrast, our method demonstrates a gradual increase in time cost. The main factors influencing its cost are the number of required paths and the size of the tangent graph, which remains almost constant with increasing map scale. The slight fluctuations in our method's cost are attributed to the minor volatility in the CPU's computational performance.



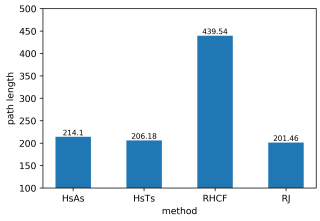
A: HA* B: HTheta* C: RHCF D: Our method

Fig. 14. These figures illustrate that the mean time cost to find the required number of paths increases with the magnification of the maps, covering all eight maps.



A: HA* B: HTheta* C: RHCF D: Our method

Fig. 15. These figures illustrate that the mean success rate to find the required number of paths increases with the magnification of the maps, covering all eight maps.



mean path length comparison

Fig. 16. These figures present a comparison of path lengths among the four methods when the required number of paths is set to 10. The results from all maps are combined, with a consistent magnification of 1.0 for all maps.

In conclusion, our method demonstrates stability with increasing map scale. It incurs a gradual increase in time cost while maintaining a 100% success rate, contrasting sharply with the notable decline in performance observed in other methods. The resilience of our method as map scale increases can be ascribed to the consistent behavior of the tangent graph, which remains stable in both size and time cost of construction, responding adeptly to variations in scale.

IV. Discussion and Conclusion

In this article, we introduce a tangent graph-based topologically distinctive path planning algorithm. It leverages the property that tangents form locally shortest paths, and any two locally shortest paths belong to different topologies. Compared to existing algorithms, our approach requires no indicator to determine whether two paths belong to the same topology. Furthermore, it eliminates the need to repeat the search for multiple paths, as all distinctive paths can be found in a single search.

To address the challenge of the exponential growth in queue size during breadth-first search, we propose an expansion reduction technique. This approach significantly reduces the time cost of searching for multiple paths while preserving computational complexity.

Our method consists of two main steps: the construction of the tangent graph and the search for multiple paths. We employ the ENL-SVG technique, as introduced by [12], to construct the tangent graph, leveraging the advantages offered by line-of-sight scans. Additionally, to avoid repeating these calculations every time the map is loaded, we save the results to a binary file, which is then loaded during the framework loading process, saving valuable time.

During the search for multiple paths, we introduce loop avoidance to prevent the repetition of waypoints in the path and avoid intersections with itself. Additionally, we apply the taut condition of [12] to ensure every path is locally shortest. It's important to note that as the number of waypoints in the path increases, the time cost of loop detection also increases. Experimental results show that the ratio of no-loop constraint checks in the total cost increases as the required number of paths increases. This occurs because the requirement for more paths often results in paths with a larger number of waypoints.

In comparison with other methods, our approach demonstrates significant efficiency when compared to RHCF, HA*, and HTheta*. Specifically, our method takes approximately 250ms to determine 400 paths from the mentioned public map dataset, while RHCF takes more than 1 second, and HA* and HTheta* take more than 10 seconds. And our method's performance is stable as the scale of map increases. However, it's important to note that our method has a limitation in that it cannot search for multiple paths with the same topology, unlike methods that use indicators like H-signature, which are capable of searching for multiple paths with the same topology. And the current version of our method is not suited for 3D Euclidean space, which limit its application.

In the future, we plan to research how to apply our method

to 3D maps and implement our method using multiple threads, as currently, it operates in a single thread. Since there is no dependence between different unfinish paths during the search process, there will be no loss of complexity when utilizing multiple threads. Additionally, we intend to apply our method to trajectory optimizations. Given that our method can provide hundreds of topologically distinctive paths in real time, it is crucial to study how to speed up trajectory optimizations to enable them to efficiently handle hundreds of paths as input. Our method is specifically applied to tangent graphs generated from static grid maps, limiting its significance. Therefore, we plan to develop a method to dynamically update tangent graphs in the future, enabling our approach to work under dynamically changing grid maps.

ACKNOWLEDGMENTS

The first author thanks Lu Zhu for her encouragement and support during the consummation of this work.

REFERENCES

[1] Subhrajit Bhattacharya. "Search-based path planning with homotopy class constraints". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 24. 1. 2010, pp. 1230–1237.

[2] Subhrajit Bhattacharya, Maxim Likhachev, and Vijay Kumar. "Search-based path planning with homotopy class constraints in 3D". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 26. 1. 2012, pp. 2097–2099.

[3] Subhrajit Bhattacharya, Maxim Likhachev, and Vijay Kumar. "Topological constraints in search-based robot path planning". In: *Autonomous Robots* 33 (2012), pp. 273–290.

[4] Naoki Katoh, Toshihide Ibaraki, and Hisashi Mine. "An efficient algorithm for k shortest simple paths". In: *Networks* 12.4 (1982), pp. 411–427.

[5] Dabin Kim et al. "Topology-guided path planning for reliable visual navigation of MAVs". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 3117–3124.

[6] Soonkyum Kim et al. "Optimal trajectory generation under homology class constraints". In: *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*. IEEE. 2012, pp. 3157–3164.

[7] Soonkyum Kim et al. "Topological exploration of unknown and partially known environments". In: *2013 IEEE/RSJ international conference on intelligent robots and systems*. IEEE. 2013, pp. 3851–3858.

[8] Keshav Kolur et al. "Online motion planning over multiple homotopy classes with Gaussian process inference". In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019, pp. 2358–2364.

[9] Markus Kuderer et al. "Online generation of homotopically distinct navigation paths". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 6462–6467.

[10] Bai Li et al. "Online trajectory replanning for sudden environmental changes during automated parking: A parallel stitching method". In: *IEEE Transactions on Intelligent Vehicles* 7.3 (2022), pp. 748–757.

[11] Y-H Liu and Suguru Arimoto. "Proposal of tangent graph and extended tangent graph for path planning of mobile robots". In: (1991), pp. 312–317.

[12] Shunhao Oh and Hon Wai Leong. "Edge n-level sparse visibility graphs: Fast optimal any-angle pathfinding using hierarchical taut paths". In: *Tenth Annual Symposium on Combinatorial Search*. 2017.

[13] Luigi Palmieri, Andrey Rudenko, and Kai O Arras. "A fast randomized method to find homotopy classes for socially-aware navigation". In: *arXiv preprint arXiv:1510.08233* (2015).

[14] Jacob J Rice and Joseph M Schimmels. "Multi-homotopy class optimal path planning for manipulation with one degree of redundancy". In: *Mechanism and Machine Theory* 149 (2020), p. 103834.

[15] Christoph Rösmann, Frank Hoffmann, and Torsten Bertram. "Integrated online trajectory planning and optimization in distinctive topologies". In: *Robotics and Autonomous Systems* 88 (2017), pp. 142–153.

[16] N. Sturtevant. "Benchmarks for Grid-Based Pathfinding". In: *Transactions on Computational Intelligence and AI in Games* 4.2 (2012), pp. 144 –148. URL: http://web.cs.du.edu/~sturtevant/papers/benchmarks.pdf.

[17] Zhuo Yao et al. "Jump Over Block (JOB): An Efficient Line-of-Sight Checker for Grid/Voxel Maps With Sparse Obstacles". In: *IEEE Robotics and Automation Letters* 8.11 (2023), pp. 7575–7582. DOI: 10.1109/LRA.2023.3320435.

[18] Zhuo Yao et al. "ReinforcedRimJump: Tangent-based shortest-path planning for two-dimensional maps". In: *IEEE Transactions on Industrial Informatics* (2019).

[19] Jin Y Yen. "Finding the k shortest loopless paths in a network". In: *management Science* 17.11 (1971), pp. 712–716.

[20] Boliang Yi et al. "Model predictive trajectory planning for automated driving". In: *IEEE Transactions on Intelligent Vehicles* 4.1 (2018), pp. 24–38.

[21] Daqing Yi et al. "Topology-aware RRT* for parallel optimal sampling in topologies". In: *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE. 2017, pp. 513–518.