Topology-Aware RRT* for Parallel Optimal Sampling in Topologies

Daqing Yi*, Michael A. Goodrich†, Thomas M. Howard‡, and Kevin D. Seppi§
*†§ Department of Computer Science, Brigham Young University, Provo, UT USA 84604
‡ Electrical & Computer Engineering Department, University of Rochester, Rochester, NY USA 14604

***Abstract*—In interactive human-robot path-planning, a capability for expressing the path topology provides a natural mechanism for describing task requirements. We propose a topology-aware RRT* algorithm that can explore in parallel any given set of topologies. The topological information used by the algorithm can either be assigned by the human prior to the planning or be selected from the human in posterior path selection. Theoretical analyses and experimental results are given to show that the optimal path of any topology can be found, including a winding topological constraint wherein the robot must circle one or more objects of interest.**

I. Introduction

Robot path-planning is commonly modeled as path optimization. In path optimization, the topology of the path is often ignored because it can be difficult to quantify, but the topology can be critical to real-world performance. There are many scenarios that require considering the topologies of paths: in collaborative search, the navigation of a robot can be constrained by a human's trajectory [1]; in surveillance, the movement of robot can be constrained to circle regions of interest [2]. In following a demonstration, a robot needs to follow a coarse path topology obtained from a human [3]. Topology can also be used to help find similar paths in re-planning.

In human-robot interaction, specifying path topology is a straightforward way for a human to describe a task. Because a human is better at high-level reasoning, directly expressing a topological requirement simplifies the path planning for a robot. For example, both avoiding risky regions or visiting regions in some temporal sequence define or at least constrain the eligible topological shapes. This kind of topological information often indicates that only part of a region needs to be considered in planning, which reduces the planning cost.

We propose a path-planning algorithm that supports topological constraints, including multi-class topological constraints. This algorithm explores in parallel all paths within each of a set of topological constraints and does so using a shared structure for the identical section of topologies. The topological information can either be assigned prior to the planning or be queried during posterior path selection.

II. Related Work

In path-planning problems, we are interested in comparing the topologies of two paths σ_1 and σ_2 that share the same start position and the same end position. σ_1 and σ_2 are *homotopic* iff one can be continuously deformed into the other without intersecting any obstacle [4]. A *homotopy class* is defined as a set of paths that are homotopic.

An approximation of homotopy is homology, which can be identified by using a complex analysis [5]. In this approach the 2D plane is modeled as a complex plane, and there is a point marked as undefined in each obstacle. By comparing the complex integral values along pairs of paths, homologous paths can be identified. In a similar way, paths are classified by homological equivalence by Delaunay-Čech complex values [6].

By approximating obstacles by polygons, a triangulation can be created to generate lines that divide the map [7], but when obstacles are not polygons, the complexity grows quickly. Another approach is to create parallel, non-intersecting rays from representative points in obstacles [8]. These rays are independent of obstacle shape. In order to support winding paths, which are self-intersecting paths (paths with loops), virtual sensor beams created from obstacles have been used to identify homotopies [2, 9]. In a similar way, a radial structure can be used that generates reference frames connecting obstacles [10].

Sampling methods have been widely used to perform efficient path-planning. RRT* [11] exploits sampling efficiency and guarantees that it will find the optimal path in the limit as samples grow. An RRT* approach has also been combined with the ability to identify completely the homotopic equivalence of two paths in 2D using a homotopic Deterministic Finite Automata (DFA) [12]. In that work the homotopic equivalence of two arbitrary paths could be determined using properties of strings recognized by the DFA, but the RRT* implementation did not fully exploit this capability.

Most of these algorithms lack completeness analyses, that is, there is no guarantee that the homotopy class can be identified for every path. Importantly, many of the homotopy- and homology-based algorithms only support finding the shortest path [13] or a feasible path [10] rather than an optimal path with respect to general objective functions. There is still a need for a complete path-planning algorithm that is capable of exploring any topology class, including winding topologies, and that guarantees that the optimal path within the topology class will be found.

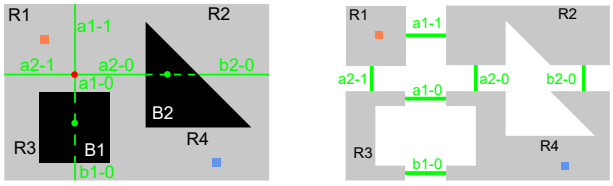
III. Path Homotopy Identification

A *simple path* is defined as a path that does not enclose any obstacle and a *simple homotopy class* is a set of simple

paths that are homotopic to each other. Prior work showed that an equivalent definition of a simple homotopy class is a homotopy class that contains at least a path that has no duplicate character [12]. This paper extends prior work to the identification to all kinds of paths, including non-simple paths. This section briefly reviews the decomposition method in [12] and proves that homotopic equivalence can be identified by comparing strings.

A. Homotopic DFA Strings

The following decomposition method, introduced in [12], uses a homotopic DFA, $\boldsymbol{M}^h$, that converts a path σ into a string v. First, a point b_k is randomly sampled in each obstacle $B_k \in \boldsymbol{B}$ as a representative point. A representative point is not allowed to lie on any line that connects any two other representative points. Second, a center point c is randomly sampled in the non-obstacle region which does not lie on any line that can be created from any two representative points from different obstacles. Third, starting at the center point a ray can be created to each representative point. Fourth, the radial structure is cut into a set of line segments by the obstacles. The set of line segments are used as a set of reference frames $\boldsymbol{R}$, which separate the map into a set of subregions $\boldsymbol{S}$. Figure 1a shows an example decomposition. In this example, the map is cut into four subregions, $R1$, $R2$,$R3$ and $R4$. The green line segments are the reference frames that define how the subregions are connected. A topology about how the subregions are connected is shown in Figure 1b. The start is an orange point and the goal is a blue point.



(a) Decomposition (b) Topology

Fig. 1: Map decomposition and its topology.

An *ID character* is assigned to each reference frame so that a sequence of crossed reference frames can be represented by a *string* of ID characters. The resulting strings can be used to identify homotopic equivalence. The homotopic DFA defined in [12] performs the conversion of a path into a string representation $v = \boldsymbol{M}^h(\sigma)$. Homotopic DFA terminology is as follows: the set of reference frames that connect with the center point c is denoted $\boldsymbol{R}_c$ and the set of subregions that connect with the center point c is denoted $\boldsymbol{S}_c$. A new subregion $\widehat{S_c}$ is created by combining all the subregions in $\boldsymbol{S}_c$. A *string block* Γ_v is the set of all paths that yield the same string v.

We now review a sequence of important properties of strings produced by the homotopic DFA [12]. Two paths in the same string block are homotopic — Property 1.

Property 1. $\forall \sigma_i, \sigma_j \in \Gamma_v, \sigma_i \simeq \sigma_j$.

Because all the paths in a string block are homotopic, each homotopy class Γ consists of infinitely many string blocks — Property 2.

Property 2. $\Gamma = \bigcup_{i=1}^{\infty} \Gamma_{v_i}$ *and* $v_i \neq v_j \Rightarrow \Gamma_{v_i} \cap \Gamma_{v_j} = \emptyset$.

The union in Property 2 is over an infinite number of subsets because a sub-path can visit arbitrarily many subregions and backtrack without enclosing an obstacle, which results in strings of arbitrary lengths. Consider a string constructed as follows: begin with the empty string ε and recursively insert a palindromic substring ww^R, where the R operator reverses the characters in the string, into any position of a string. A string made up of *R*ecursively *E*mbedded *P*alindromic substrings is denoted an *REP string* [12]. Note that ε and strings of the form ww^R are REP strings.

The shortest string in a homotopy class, denoted v^*, is not an REP string. Each homotopy class can be represented using this shortest string — Property 3.

Property 3. $\forall v, \exists v^*, \Gamma_v \simeq \Gamma_{v^*}$

Since the shortest path in a homotopy class does not contain a subpath that visits several subregions and backtracks, the Homotopic DFA represents the shortest path in a homotopy class by the shortest string v^*. Algorithm 2 in [12] removes REP substrings by sequentially pushing characters from the string onto a stack *unless* the character at the top of the stack matches the next character in the string [12]. If the top of the stack and the next character match, the stack is popped to eliminate the palindromic structure.

Consider two paths in the same homotopy class, σ and σ^*. Suppose that σ^* is the shortest path. REPTrim() converts string $v = \boldsymbol{M}^h(\sigma)$ into $v^* = \boldsymbol{M}^h(\sigma^*)$ — Property 4.

Property 4. $v^* = \text{REPTrim}(v)$.

The homotopic DFA converts two concatenated paths into two concatenated strings. Because of the recursive nature of Algorithm 2 in [12], these strings decompose into shortest strings — Property 5.

Property 5. $\text{REPTrim}(v_1 v_2) = \text{REPTrim}(v_1^* v_2^*)$
$= \text{REPTrim}(\text{REPTrim}(v_1)\text{REPTrim}(v_2))$.

Algorithm 2 in [12] will never have two consecutive characters the same in the stack. This implies that there are no two consecutive characters that are the same in the output v^*.

Property 6. $\forall i \in \{1, \ldots, |v^*| - 1\} v^*[i] \neq v^*[i+1]$.

B. Identifying Homotopic Equivalence

We now extend results beyond prior work. We show that we can use the shortest string v^* to determine when two paths are homotopic. This requires two lemmas.

Lemma 1. $\text{REPTrim}(v_i^* v_j^{*R}) = \varepsilon \Leftrightarrow v_i^* = v_j^*$.

Proof. Suppose $\text{REPTrim}(v_i^* v_j^{*R}) = \varepsilon$. When $v_i^* v_j^{*R}$ is input to REPTrim(), v_i^* is pushed onto the stack first. The no duplicate character property of shortest strings, Property 6,

means that the entire string v_i^* will be on the stack. Thus, the length of the stack is $|v_i^*|$. In order to get an empty string ε as the output, the stack needs to be cleared. This requires that $|v_i^*| = |v_j^{*R}|$ and $v_i^{*R} = v_j^{*R}$, which implies that $v_i^* = v_j^*$.

Conversely, when $v_i^* = v_j^*$, simulating stack pushes/pops shows that $\text{REPTrim}(v_i^* v_j^{*R}) = \text{REPTrim}(v_i^* v_i^{*R}) = \varepsilon$. □

We also have Lemma 2 for an equivalence in the path space.

Lemma 2. $\sigma_1 \simeq \sigma_2 \Leftrightarrow \text{REPTrim}(\boldsymbol{M}^h(\sigma_1 \circ \sigma_2^R)) = \varepsilon$.

Proof. Suppose $\sigma_1 \simeq \sigma_2$. Concatenating σ_1 with a reversed σ_2 creates the path $\sigma_1 \circ \sigma_2^R$, which encloses no obstacle by the definition of homotopy. $\sigma_1 \circ \sigma_2^R$ is homotopic to the (closed) path that starts and ends at the same position, $\sigma_1(0)$. Since this closed path encloses no obstacle, $\sigma_1 \simeq \sigma_2 \Rightarrow \sigma_1 \circ \sigma_2^R \simeq \sigma_1(0)$. This means that the closed path $\sigma_1 \circ \sigma_2^R$ recursively visits several subregions and backtracks, returning to the starting point. Applying REPTrim() to such a path yields the empty string.

Conversely, suppose $\text{REPTrim}(\boldsymbol{M}^h(\sigma_1 \circ \sigma_2^R)) = \varepsilon$. We use a proof by contradiction. Assume that $\sigma_1 \not\simeq \sigma_2$. By Property 5, $\text{REPTrim}(\boldsymbol{M}^h(\sigma_1 \circ \sigma_2^R)) = \text{REPTrim}(v_1^* v_2^{*R})$. By Lemma 1, we have $v_1^* = v_2^*$. Because $\sigma_1 \not\simeq \sigma_2$, let $\sigma_1 \in \Gamma_1$ and $\sigma_2 \in \Gamma_2$, we have $\Gamma_1 \not\simeq \Gamma_2$. Let $\text{min\_len}(\Gamma_i) = \arg\min_{\sigma \in \Gamma_i} |\sigma|$ be the shortest path in Γ_i. We have $\text{min\_len}(\Gamma_1) \not\simeq \text{min\_len}(\Gamma_2)$. However, by the definition of the shortest string, $\boldsymbol{M}^h(\text{min\_len}(\Gamma_1)) = \boldsymbol{M}^h(\text{min\_len}(\Gamma_2)) = v_1^* = v_2^*$. By Property 1, it means that $\text{min\_len}(\Gamma_1) \simeq \text{min\_len}(\Gamma_2)$. This is a contradiction. □

By Lemma 2 and Lemma 1, we can derive Theorem 1.

Theorem 1. $\text{REPTrim}(\boldsymbol{M}^h(\sigma_1)) = \text{REPTrim}(\boldsymbol{M}^h(\sigma_2))$ *iff* $\sigma_1 \simeq \sigma_2$.

Proof. Let $v_1^* = \text{REPTrim}(\boldsymbol{M}^h(\sigma_1))$ and $v_2^* = \text{REPTrim}(\boldsymbol{M}^h(\sigma_2))$.

When $\text{REPTrim}(\boldsymbol{M}^h(\sigma_1)) = \text{REPTrim}(\boldsymbol{M}^h(\sigma_2))$, we have $v_1^* = v_2^*$. By Lemma 1, we know $\text{REPTrim}(v_1^* v_2^{*R}) = \varepsilon$. Thus, $\text{REPTrim}(\boldsymbol{M}^h(\sigma_1)\boldsymbol{M}^h(\sigma_2)^R) = \varepsilon$. By Lemma 2, we have $\sigma_1 \simeq \sigma_2$.

When $\sigma_1 \simeq \sigma_2$, we can create a path $\sigma_1 {\sigma_2}^R$ by concatenating σ_1 and a reversed σ_2. Without loss of generality, we can assume that this path starts and ends at the same position. By Lemma 2 and Property 5, we have $\varepsilon = \text{REPTrim}(\boldsymbol{M}^h(\sigma_1 {\sigma_2}^R)) = \text{REPTrim}(v_i^* v_j^{*R})$. By Lemma 1, we know $v_i^* = v_j^*$, which implies $\text{REPTrim}(\boldsymbol{M}^h(\sigma_1)) = \text{REPTrim}(\boldsymbol{M}^h(\sigma_2))$. □

Theorem 1 tells us that we can identify the homotopy of paths by comparing the strings generated by the homotopic DFA after passing them through REPTrim Algorithm [12]. We use this to create an RRT*-based planner that only generates solutions constrained to one or more homotopy classes.

IV. Topology-Aware RRT*

In this section, we propose a topology-aware random sampling algorithm that is derived from the standard RRT* [11]. Assumptions from [11] about RRT* are inherited here. We are interested in both non-simple paths and non-simple homotopy classes, but Property 2 implies that there exists an infinite number of possible string blocks in a homotopy class. If we assume the optimal path is not an infinitely long path (which will be true for cost functions that monotonically increase with path length), the string produced by homotopic DFA for optimal path will not be infinitely long. We state this assumption and use it to limit the number of possible subtrees produced by our algorithm.

Assumption 1. $\forall \sigma^* = \arg\min_{\sigma \in \Gamma} \textsc{Cost}(\sigma), \exists \tau \geq 1, |v| \leq \tau |v^*|$ *and* $\sigma^* \in \Gamma_v$.

In this assumption, the topological constraint is represented by the set Γ that contains all homotopy classes consistent with this constraint. We assume that we can restrict search to paths that (a) are in a homotopy class consistent with an optimal path and (b) do not produce really long strings.

A. Expanding Topology

The basic idea of the Topology Aware RRT* (TARRT*) algorithm is that the tree produced by RRT* is "chunked" into subtrees when the paths in those subtrees produce different string blocks; extension of the trees is restricted to only those string blocks that are consistent with the desired topological constraint(s). The RRT* subtrees find the optimal path within each string block, and the optimal path for the homotopy class is the best of the paths for the string blocks.

We will be talking about two different trees: the tree produced by RRT* and the tree of subtrees produced when we group the RRT* branches into string blocks. To help distinguish between the elements of the tree of subtrees and the elements of the tree, we will refer to the subtrees that belong to the same string block as a *TARRT* node* and the individual elements of the complete tree as *RRT* vertices*.

Figure 2a illustrates a very simple world with four regions. These four regions are separated by four reference frames, "A1,B1,A2,B2", and the homotopic DFA adds the label for these references frames to the string whenever the path crosses the reference frame. The small red square indicates the start position and the small blue square indicates the goal position. Thus, they represent different string blocks and different sequences of TARRT* nodes that can be created.

Each TARRT* node is associated with a subregion, and each edge in TARRT* is associated with a reference frame. There exist similarities between string blocks. For example, all the string blocks start in the same initial subregion "R1". The string block "B1,A2,B2,B2" contains a substructure that is identical with the string block "B1,A2". We can thus use an expanding topology to efficiently express these string blocks, as shown in Figure 2c. The *root TARRT* node* is always associated with the start subregion. Denote a TARRT* node associated with the goal subregion as a *terminal TARRT* node*.

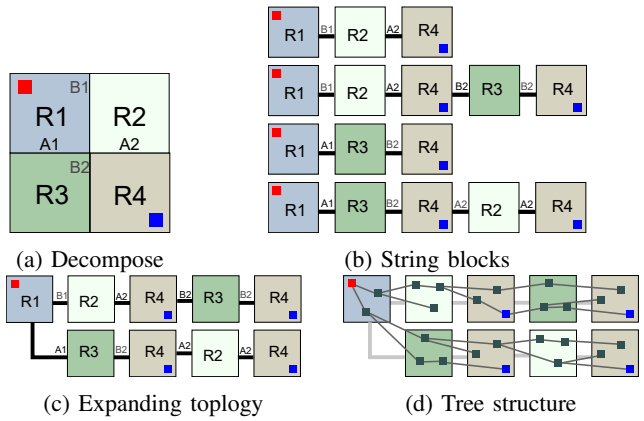


Fig. 2: Expanding Topology.

Any path from the root expanding node to a terminal TARRT* node defines a string block, which is called a *string-block branch* of the TARRT* tree. In Figure 2c, each path from the TARRT* node "R1" to a TARRT* node "R4" is within one of the string blocks in Figure 2b.

RRT* uses directed random sampling to create new possible nodes in the RRT* subtrees. Since each one of the new possible RRT* vertices is located in a subregion, it is possible that the location of the new node can be part of multiple string blocks and their corresponding TARRT* nodes. If we can generate an optimal structure like RRT* but sorted by string blocks, backtracking from a goal position in a terminal expanding node to the root obtains the optimal path of the corresponding string block.

B. Topology-Aware Space Sampling

The TARRT* algorithm is given as Algorithm 1. It inherits optimal spatial sampling from RRT* but the tree generation process is guided by an expanding topology of TARRT* nodes. The branches of the tree are sorted by string-block branches of the expanding topology, like in Figure 2d.

The algorithm enforces a tree-of-subtrees structure by ensuring that the parent RRT* vertex of any RRT* vertex can only be located (a) within the same TARRT* node as the RRT* vertex or (b) in the parent node of that TARRT* node. Moreover, if an existing RRT* vertex is linked with a new RRT* vertex, the edge between those vertices visit reference frames as defined in the expanding topology of TARRT*.

For example, consider the string block "B1, A2" at the top of Figure 2b and an RRT* vertex in the TARRT* node "R4". If the RRT* vertex has a parent in TARRT* node "R2", the edge between child and parent should cross the reference frame "A2". If the RRT* vertex has an edge connecting to a grandparent RRT* vertex in TARRT* node "R1", the edge should cross the reference frames "B1","A2" sequentially. If an RRT* vertex has an edge towards a node in the TARRT* node "R3", the edge violates the requirement of this string block and should not exist.

Because a subregion is associated with multiple TARRT* nodes, for each new position obtained via directed sampling

Algorithm 1 Topology-Aware Rapidly-exploring Random Tree* $G(V, E)$

$V \leftarrow \{x_{init}\}$; $E \leftarrow \emptyset$; $i \leftarrow 0$
while $i < N$ **do**
  $x_{rand} \leftarrow$ SAMPLE (i) ; $i \leftarrow i + 1$
  $x_{nrst} \leftarrow$ NEAREST (G, x_{rand})
  $x_{new} \leftarrow$ STEER $(x_{nrst}, x_{rand}, \eta)$
  if OBSTACLEFREE(x_{nrst}, x_{new}) **then**
    $R =$ REGION (x_{new})
    for each $tarrt\_node$ in TARRTNODES (R) **do**
      $tarrt\_node \leftarrow x_{new}$
      $G \leftarrow$ EXTEND (G, x_{new}, x_{nrst})

in RRT* a new RRT* vertex will be created in each relevant TARRT* node; relevant TARRT* nodes are those associated with the subregion in which the new position lies. This means that when a new position is sampled, there are new RRT* vertices created in several associated TARRT* nodes. For example, a new position (the yellow square) is sampled in the subregion "R3", as illustrated in Figure 3a. Two new RRT* vertices of the position are added to the two TARRT* nodes, one for the top string block topology and another for the bottom string block topology as shown in Figure 3b.

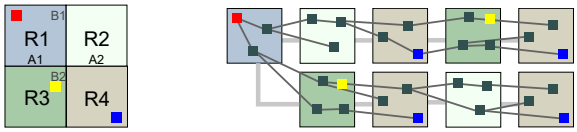


Fig. 3: Sampling and adding new nodes.

We now define several functions, using appropriately modified definitions from the RRT* algorithm in [11].

- SAMPLE(): Returns independent uniformly distributed samples from X_{free}.
- NEAREST(): Returns a position of the vertex whose position is closest to point x. NEAREST$(G = (V, E), x)$ = $\arg\min_{v \in V} \|x - v\|$.
- STEER(): Given two points x and y, returns a point z on the line segment from x to y that is no greater than η from y. STEER(x, y, η) = $\arg\min_{z \in \boldsymbol{R}^d, \|z-x\| \leq \eta} \|z - y\|$.
- OBSTACLEFREE(x, x'): Returns *True* if $[x, x'] \subset X_{free}$, which is the line segment between x and x' lies in X_{free}.
- REGION(x): Returns the subregion that position x is in.
- TARRTNODES(R): Returns all TARRT* nodes from the expanding topology that are associated with subregion R.

The RRT* vertices of the TARRT* tree are created and stored in TARRT* nodes. This provides information for how to add connections between new positions to potential parent RRT* vertices and also how to rewire RRT* vertices so that rewiring honors string block constraints. Thus, the EXTEND procedure of TARRT* is slightly different with that in RRT*. It is stated in Algorithm 2.

The precise definitions of the methods used in the Algorithm 2 are given below.

Algorithm 2 EXTEND($G, x_{new}, x_{nearest}$)

if $x_{new} = x_{nrst}$ **then return** $G = (V, E)$
$V' \leftarrow V \cup \{x_{new}\}$
$x_{min} \leftarrow x_{nrst}$
$X_{near} \leftarrow$ NEAR($G, x_{new}, |v|$)
for each $x_{near} \in X_{near}$ **do**
  if OBSTACLEFREE(x_{new}, x_{near}) **and** HOMOTOPY-ELIGIBLE(x_{new}, x_{near}) **then**
    $c'_k \leftarrow$ COST$_k(x_{near})$ $+c_k($ LINE(x_{near}, x_{new}) $)$
    if $c'_k <$ COST$_k(x_{new})$ **then**
      $x_{min} \leftarrow x_{near}$
$E' \leftarrow E' \cup \{(x_{min}, x_{new})\}$
for each $x_{near} \in X_{near} \setminus \{x_{min}\}$ **do**
  if OBSTACLEFREE(x_{near}, x_{new}) **and** HOMOTOPYELIGIBLE(x_{near}, x_{new}) **then**
    $c'_k \leftarrow$ COST$_k(x_{new})$ $+c_k($ LINE(x_{new}, x_{near}) $)$
    if $c'_k <$ COST$_k(x_{near})$ **then**
      $x_{parent} \leftarrow$ PARENT(x_{near})
      $E' \leftarrow E' \setminus \{(x_{parent}, x_{near})\}$
      $E' \leftarrow E' \cup \{(x_{new}, x_{near})\}$
return $G' = (V', E')$

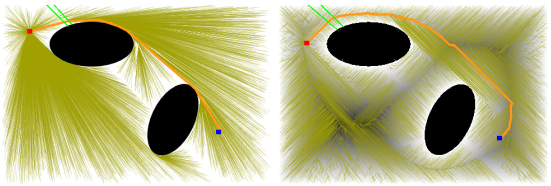
- NEAR($G, x, card$): Returns all vertices within the closed ball of radius $\gamma = \min\{\gamma_{RRT*}(\log(card)/card)^{1/d}, \eta\}$ centered at x, in which $\gamma > (2(1 + 1/d))^{1/d}(\frac{\mu(X_{\text{free}})}{\zeta_d})^{1/d}$ [11].
- HOMOTOPYELIGIBLE(x_{from}, x_{to}): Uses REPTRIM() to return *True* if the sequence of reference frames a line visits is consistent with a required sequence of reference frames. The line is from x_{from} to x_{to}. The required sequence of reference frame is obtained by the sequence of edges from the TARRT* node that x_{from} is in to the TARRT* node that x_{to} is in.
- LINE(x, x') : $[0, s] \leftarrow X_{free}$ denotes the path defined by line segment from x to x'.
- COST(x): Returns cost of the unique path (because G is a tree) from x_{init} to the vertex $x \in V$. COST(x_{init}) = 0.

V. EXPERIMENTS

This section presents a series of informative examples that illustrate how the TARRT* algorithm works, that provide empirical support for the claims in the paper, and that illustrate some useful properties of the algorithm. The examples include some use cases. The first use case is when a human specifies a single homotopy class as the topological constraint and the algorithm returns the optimal path subject to that constraint. The second use case is when the human specifies some things to avoid; these are translated into a topological constraint that includes multiple homotopy classes, and then the human selects from multiple possible paths returned by the algorithm.

A. Optimality and Practicality

Figure 4 illustrates TARRT* for a homotopy class where the path is required to go above the obstacle at the top of the world. Black blobs indicate obstacles. The orange line is a found path from the start (red point) to the goal (blue point). The olive lines visualize the tree structure generated by TARRT*. There are also green line segments that show the reference frames associated with a string block in a homotopy class.



(a) Minimum Distance (b) Maximum Safety

Fig. 4: Optimality. $\tau = 1$.

The algorithm behavior when path length is the objective is given in Figure 4a and when minimizing distance to any obstacle is the objective is given Figure 4b. The gray background shows the cost map distribution; darker means lower cost and the lighter means higher cost. Because the two reference frames happened to be very close together, the algorithm wastes a lot of time exploring to the lower left of the obstacles. Nevertheless, the algorithm returns the shortest path and the minimum cost path, respectively, in the world.

B. Use Case 1

In a human-to-robot approach, a homotopy class is provided a priori by a human. Consider the world illustrated in Figure 5 and two constraints provided by the human. In the first constraint, the human draws a path that goes south of every obstacle. In the second constraint, the human draws a path that goes to the west of the obstacle in the lower center of the world. The human-drawn paths were translated into shortest strings using the REPTIME algorithm, and then string blocks that were consistent with these shortest strings were passed to TARRT*.

Because a homotopy class from a human is given, subregions that are not associated with the homotopy class need not be explored. As a result, the tree extends only to part of the map and consists of fewer subregions. Figure 5 illustrates two examples where the optimal paths for a cost of avoiding near approaches to obstacles can be found even when only part of the map is explored. A tight constraint enhances efficiency.

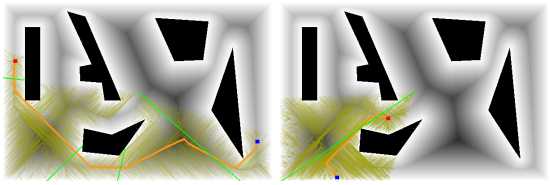
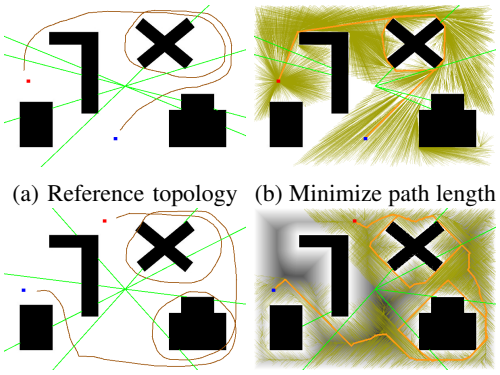


Fig. 5: Non-winding topology.

So far, the examples have considered only simple homotopy classes. TARRT* is able to find the optimal path of a winding topology. Figure 6b shows the optimal path of a homotopy class that contains single windings of two different obstacles using minimum path length as the objective. The reference topology was drawn by a human. Figure 6d uses a topology

constraint that winds around two obstacles for a cost function related to obstacle proximity (darker is better).



(a) Reference topology (b) Minimize path length

(c) Reference topology (d) Maximize safety

Fig. 6: Wwinding topology.

C. Use Case 2

Suppose that the human only specifies a region to avoid, leaving many different homotopy classes that might satisfy this constraint. In the extreme, suppose that the human wants to see the lowest cost path for every possible topology. Finally, suppose that the human specifies an upper bound on how long the path can be, which turns the problem from one of searching through an infinite number of possible homotopy classes to searching the finite number of classes that satisfy the stretching assumption (Assumption 1). Figure 7 illustrates TARRT* exploring different homotopies in parallel. Each subfigure shows the portion of the TARRT* tree relevant for one of the homotopy classes along with the best path from that homotopy class.

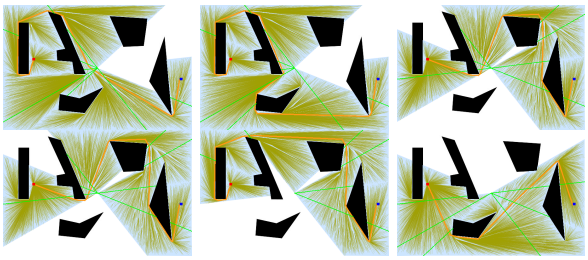


Fig. 7: Multiple string blocks.

VI. Summary

In this paper, we proved that the decomposition method and homotopic DFA can be applied not only to simple paths but also to non-simple paths. We also proposed the TARRT* algorithm, which provides an efficient sampling structure for exploring a topological constraint over multiple homotopy classes. TARRT* enforces samplings that honor a set of possible homotopy classes and rewires the RRT* tree so that it explores multiple homotopy classes in parallel. TARRT* finds a solution for any homotopy class with finite length paths including winding topologies.

Future work should apply TARRT* to different decomposition methods and in higher dimension spaces such as in a robotic manipulation problem using a 3D decomposition method. Practical complexity of the algorithm will likely become an issue for these problems.

References

[1] D. Yi, M. Goodrich, and K. Seppi, "Informative path planning with a human path constraint," in *Systems, Man and Cybernetics (SMC), 2014 IEEE International Conference on*, Oct 2014, pp. 1752–1758.

[2] V. Narayanan, P. Vernaza, M. Likhachev, and S. LaValle, "Planning under topological constraints using beam-graphs," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, May 2013, pp. 431–437.

[3] D. Shah and M. Campbell, "A robust qualitative planner for mobile robot navigation using human-provided maps," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, May 2011, pp. 2580–2585.

[4] 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.

[5] S. Bhattacharya, "Search-based path planning with homotopy class constraints," 2010.

[6] F. T. Pokorny, M. Hawasly, and S. Ramamoorthy, "Multiscale topological trajectory classification with persistent homology," in *Proceedings of Robotics: Science and Systems*, Berkeley, USA, July 2014.

[7] D. Demyen and M. Buro, "Efficient triangulation-based pathfinding," in *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1*, ser. AAAI'06. AAAI Press, 2006, pp. 942–947.

[8] S. Kim, S. Bhattacharya, and V. Kumar, "Path planning for a tethered mobile robot," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, May 2014, pp. 1132–1139.

[9] P. Vernaza, V. Narayanan, and M. Likhachev, "Efficiently finding optimal winding-constrained loops in the plane," in *Proceedings of Robotics: Science and Systems*, Sydney, Australia, July 2012.

[10] E. Hernandez, M. Carreras, J. Antich, P. Ridao, and A. Ortiz, "A topologically guided path planner for an auv using homotopy classes," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, May 2011, pp. 2337–2343.

[11] 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, Jun. 2011.

[12] D. Yi, M. A. Goodrich, and K. D. Seppi, "Homotopy-Aware RRT*: Toward human-robot topological path-planning," in *The Eleventh ACM/IEEE International Conference on Human Robot Interation*, ser. HRI '16. Piscataway, NJ, USA: IEEE Press, 2016, pp. 279–286.

[13] A. Efrat, S. G. Kobourov, and A. Lubiw, "Computing homotopic shortest paths efficiently," *Computational Geometry*, vol. 35, no. 3, pp. 162 – 172, 2006.