Volumetric Tree*: Adaptive Sparse Graph for Effective Exploration of Homotopy Classes

Donghyuk Kim[1], Mincheul Kang[1] and Sung-Eui Yoon[2]

***Abstract*— We present volumetric tree*, a hybridization of sampling-based and optimization-based motion planning. Volumetric tree* constructs an adaptive sparse graph with volumetric vertices, hyper-spheres encoding free configurations, using a sampling-based motion planner for a homotopy exploration. The coarse-grained paths computed on the sparse graph are refined by optimization-based planning during the execution, while exploiting the probabilistic completeness of the sampling-based planning for the initial path generation. We also suggest a dropout technique probabilistically ensuring that the sampling-based planner is capable of identifying all possible homotopies of solution paths. We compare the proposed algorithm against the state-of-the-art planners in both synthetic and practical benchmarks with varying dimensions, and experimentally show the benefit of the proposed algorithm.**

I. Introduction

Path and motion planning problem is a fundamental research area in robotics and has been widely studied for autonomous systems with mobility and manipulability. Among various categories of planning algorithms, sampling-based approaches have attracted considerable attention thanks to its probabilistic completeness [1], [2] and almost-sure asymptotic optimality [3]. Their key concept is to construct a random geometric graph or tree to identify the connection of feasible motions in the configuration-free (C-free) space.

It has been well-known that when applied to practical problems, sampling-based motion planners require a considerable amount of computation cost to check collision for vertices and edges, especially for computing the optimal solution in high dimensions [4], [5]. To this end, there have been a plethora of researches to alleviate the overhead of collision checking. Hauser [4] proposed a lazy collision checking with DSPT (Dynamic Shortest Path Tree [6]) to delay the explicit checking until it is necessary, while preserving the asymptotic optimality and anytime properties. Bialkowski et al. [5] presented a graph associated with safety certificates, i.e., a set of collision-free balls to reduce the amortized complexity of collision checking. Gammell et al. [7] combined sampling-based motion planning and Lifelong Planning A* (LPA*) with a batch sampling technique. Their approach expands the graph using LPA*-style incremental search techniques, performing graph expansion and collision checking on a partial subset of graph components.

Meanwhile, optimization-based planners and the hybridization of sampling and optimization also has been suggested to get synergy in various manner. CHOMP (Covariant Hamiltonian Optimization for Motion Planning) [8], its variants [9], [10] and TrajOpt [11] formulate the trajectory optimization as a convex problem to minimize the local approximation of the objective function sequentially.

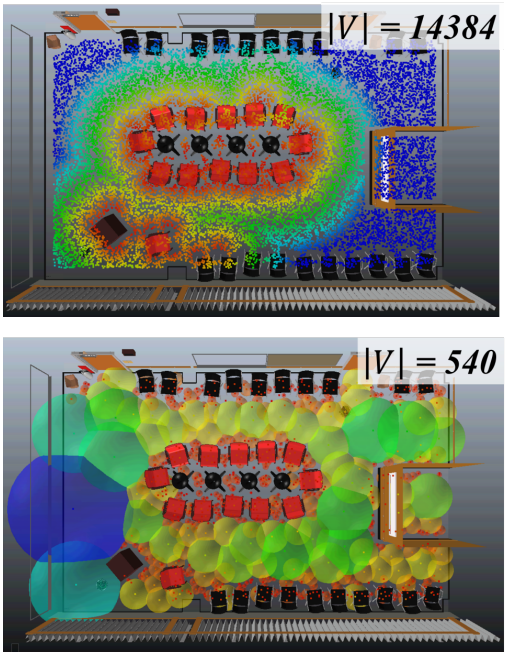


Fig. 1: A heatmap-style visualization of the vertex set V, constructed by a conventional planner (top, $|V| = 14384$) and that of volumetric tree* (bottom, $|V| = 540$) in the same time budget. We can observe the volumetric tree* constructs a sparse graph, while capturing the samples around narrow passages or boundaries. The vertices close to the obstacles are encoded red; otherwise blue.

As a hybridization of sampling and optimization, Choudhury et al. [12] suggested RABIT* (Regionally Accelerated Batch Informed Trees), where a sampling-based planner runs as a global planner, and an optimization-based planner takes over the local planning. Kim et al. [13] proposed a similar hybridization, Dancing PRM*, whose obstacle potential computation is solely done in the configuration space. It directly approximates the C-free space by utilizing the empirical collisions found during the execution. Kuntz et al. [14] presented another hybrid algorithm, which locally refines solution paths using lazy interior point optimization to compute high-quality solutions quickly.

[1]Donghyuk Kim and Mincheul Kang are with the School of Computing ({donghyuk.kim, mincheul.kang}@kaist.ac.kr) and [2]Sung-Eui Yoon is with the Faculty of School of Computing (Corresponding author, sungeui@kaist.edu), Korea Advanced Institute of Science and Technology (KAIST) at Daejeon, South Korea 34141.

When it comes to a scalability issue, recent researches have also pointed out the significance of nearest-neighbor search [15], [16]. Kleinbort et al. [15] analyzed the asymptotic computational bottleneck in sampling-based motion planning. To be specific, the complexity of collision detection is proportional to that of the given workspace, while nearest-neighbor search asymptotically dominates the entire computation time as the number of samples goes to infinity. Varricchio and Frazzoli [16] proposed pruning techniques for the k-d tree to reduce the computational cost of nearest neighbor search.

Main Contributions. In this work, we present Volumetric Tree*, a hybridization of sampling-based and optimization-based motion planning for effectively exploring the homotopy class of solution paths and identifying local optimums in each homotopy class (Sec. III-A). In volumetric tree*, the role of a sampling-based motion planner is used for a homotopy exploration. To be specific, it is designed to construct an adaptive sparse graph, where wide-open areas are covered by a fewer number of volumetric vertices (hyperspheres) while maintaining fine-grained vertices around a boundary of free space or narrow passage (Sec. III-B), instead of constructing a dense graph. On top of that, we can represent a set of paths homotopic to each other into a sequence of volumetric vertices as a compact representation. To complement the coarse-grained paths computed on the sparse graph, we combine an optimization-based motion planner (Sec. III-C) to refine the solution paths with a dedicated shortest path computation technique, dropout (Sec. III-D). As a result, volumetric tree* efficiently identifies initial paths in multiple homotopy classes in a sampling-based manner for the optimization-based planning by complementing each other.

According to conducted experiments, we observe up to 3 times speedup over other tested methods in terms of convergence to the optimum on rigid body and manipulation planning problems (Sec.IV). These results are mainly obtained by the adaptive sparse graph integrated with the optimization-based planning, which allows volumetric tree* to explore the configuration space efficiently.

II. Algorithmic Background

We first formulate the problem we would like to address and review preliminaries of sampling-based motion planning.

A. Problem Definition

Let $\mathcal{X} = \mathbb{R}^d$ be the configuration space, where d is the dimension of a given problem. Let the configuration-obstacle (C-obs) space be $\mathcal{X}_{obs}$, which is a set of states in collision with obstacles. The complement of $\mathcal{X}_{obs}$, $\mathcal{X}_{free}$ ($= \mathcal{X} \setminus \mathcal{X}_{obs}$), becomes the configuration-free (C-free) space. For a given pair of an initial and goal configurations, x_{init} and $x_{goal} \in \mathcal{X}_{free}$, respectively, let $\sigma \in \Sigma : [0,1] \rightarrow \mathcal{X}_{free}$ be a feasible (e.g., collision-free) path, where Σ is a set of all possible paths satisfying $\sigma(0) = x_{init}$ and $\sigma(1) = x_{goal}$.

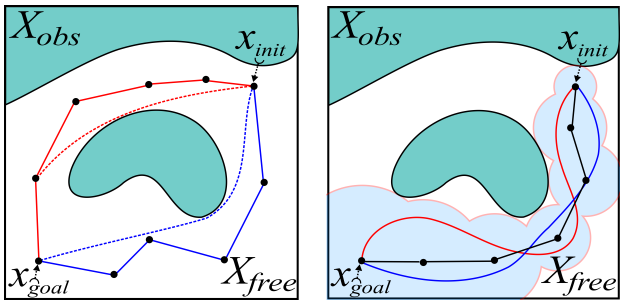


Fig. 2: The left figure illustrates two solution paths, shown red and blue solid lines, in two different homotopy classes with their local optimal paths (dotted). The right figure shows a solution path (solid black) covered by a sequence of collision-free balls and paths homotopic to the solution path (red and blue). These observations suggest that such a coarse-grained graph can be a sufficient representation if we can optimize each path toward the local optimum.

The optimal motion planning problem then can be formulated as:

$$\sigma^* = \underset{\sigma \in \Sigma}{\operatorname{argmin}} \left(c(\sigma)\right), \tag{1}$$

where $c(\cdot)$ is a cost function such that $c : \Sigma \rightarrow \mathbb{R}_{\geq 0}$.

For a sampling-based planner, it is said to be almost-sure asymptotic optimal if the probability that a path computed by the planner at an iteration number i, σ_i, converges to the optimal path is 1, as the number of iteration goes to infinity:

$$\mathbb{P}\Big[\lim_{i \rightarrow \infty} \big(c(\sigma_i) = c(\sigma^*)\big)\Big] = 1 \tag{2}$$

B. Sampling-based Motion Planning

There have been proposed various optimal sampling-based motion planning algorithm such as RRT*, PRM* [3], and BIT* [7]. They sample a set of configurations, $\mathcal{X}_{sample} \in \mathcal{X}$, to construct a search graph $G = \{V, E\}$, where each vertex $v \in V$ represents a collision-free configuration and $e = (v, w) \in E$ is a feasible motion for a pair of configurations v and $w \in V$.

V is initialized with $\{x_{init}, x_{goal}\}$ and E is set to empty, then we sample a random configuration at each iteration until the termination condition is satisfied, e.g., time limit or a maximum number of iterations.

For a sample configuration x_{sample}, the edge connection process is done with its near neighbors found by $Near(\cdot)$. $Near(x)$ returns a set of near vertices in V such that those vertices lie inside of an implicit ball centered at x with a connection radius r. This type of NN search is also called ε-NN search. k-nearest neighbor search can be used alternatively; in either case, both connection radius r and k are determined proportionally to the cardinality of V to achieve the almost-sure asymptotic optimality [3]. Each vertex and edge is then checked for a collision, and only valid components are inserted into G for the graph expansion. Finally, a path connecting x_{init} and x_{goal} becomes a discrete solution path $\sigma \in \Sigma$, and its cost $c(\sigma)$ can be computed by summing up the cost of all edges over the path.

III. ALGORITHM

A. Motivations

We would like to explain motivations of volumetric tree*, a hybridization of sampling and optimization-based planning, with the following two aspects. Firstly, Fig. 2(a) shows two exemplar paths shown in the solid lines that connect x_{init} and x_{goal} in different homotopy classes. Once a solution path is computed, we can apply a local optimal planner such as an iterative optimization-based planners [8] to refine the path toward the local optimal path (dotted lines in Fig. 2(a)) in the same homotopy class. Based on this observation, we suggest using a sampling-based motion planner, which guarantees probabilistic completeness, for a homotopy exploration to identify solution paths in all possible homotopy classes. On top of that, an optimization-based planner then locally optimizes the initial paths computed by the sampling-based planner toward the local optimum.

Once we realize the hybridization of sampling-based and optimization-based approaches, it is unnecessary to construct a dense graph. This is mainly because the cost of an initial path is not that important for the optimization-based planning. Instead, it is more important to find different homotopic classes of initial paths. Moreover, constructing such a dense graph would require a massive computational cost due to graph manipulation operations such as NN-search. As a result, the sparse representation of the given space can be a reasonable choice for our objective.

In particular, we associate each vertex of a graph with a collision-free hyper-sphere volume for locally representing the C-free space, while rejecting samples inside volumes, resulting in a sparse graph. The idea of using the hyper-sphere volume is inspired by the previous literature [5], [13], [17], [18]. Fig. 2(b) shows an exemplar solution path over a sparse graph constructed in the proposed manner, where a set of collision-free hyper-spheres are associated with vertices.

We then utilize the following observation. For a collision-free hyper-sphere S, we can assume that any two given configurations located within S can be connected directly without having any collision. More importantly, it implies that for a given discrete path $\sigma = \{v_0, ..., v_{n-1}\} \in \Sigma$, associated with collision-free hyper-spheres S_{v_i}, centered at v_i, we can then expect any path through the same sequence of vertices located in the hyper-spheres, $\sigma' = \{v'_0 \in S_{v_0}, ..., v'_{n-1} \in S_{v_{n-1}}\}$ is homotopic to σ. We can hence represent plenty of paths homotopic to each other into a single path over the adaptive sparse graph. Our volumetric tree* utilizes this observation for creating a sparse graph and computes the optimal paths with the graph. Fig. 1 shows two different types of graphs, a dense graph computed by the conventional approach and ours in an example scene.

B. Adaptive Sparse Tree Construction

Volumetric tree* constructs a random geometric graph $G = \{V, E\}$, where a vertex $v \in V$ also encodes a collision-free configuration; depending on the context, we can just use v to denote its configuration, and at that case, $v \in \mathcal{X}_{free}$. An edge $e = (v \in V, w \in V) \in E$ represents a continuous motion connecting two configurations. We also define $S_v (\forall v \in V)$ to represent a hyper-sphere centered at a configuration v, associated with a radius of r_v, a distance to the closest empirical collision $o_v \in \mathcal{X}_{obs}$. They are additionally stored in each v.

Algorithm 1: VOLUMETRIC TREE*

```
V ← {x_init, x_goal}; T ← {x_init}; E ← ∅
while Termination condition is not satisfied do
    x_sample ← Sample()
    if IsCollisionFree(x_sample) then
        V_near ← Near(x_sample, V)
        if ¬IsInside(x_sample, V_near) then
            Insert x_sample to V
            foreach v_near ∈ V_near do
                Insert (v_near, x_sample) to E
            PropagateCFreeSpace(x_sample, V_near)
            σ_new ← UpdateDSPT(x_sample, T)
    else
        V_near ← Near(x_sample, V)
        UpdateCFreeSpace(V_near, x_sample)
    if BetterPathFound(σ_new) then
        OptimizePath(T, G)
return SolutionPath(T)
```

Our hyper-sphere based representation is designed for approximately encoding $\mathcal{X}_{free}$, and its construction is inspired by the approximate C-free representation proposed in [13]. The main idea of C-free approximation is to associate each vertex with the closest empirical collision found during the execution, resulting in a set of approximate collision-free hyper-spheres, reducing the approximation error over iterations probabilistically.

For $Near(\cdot)$, we use a distance function specialized for considering radii of vertices to measure a distance between two hyper-spheres associated with those two vertices:

$$dist_{NN}(v \in V, w \in V) = \|v - w\| - r_v - r_w, \tag{3}$$

where $\|\cdot\|$ is the Euclidean norm of a vector. The reason why we use a specialized $dist_{NN}(\cdot)$ is to enable a vertex with a large radius to be better connected to other samples for $Near(\cdot)$, either r-NN or k-NN, during the graph expansion. Note that the cost of an edge $e = (v, w)$ is still defined as conventional, i.e., $c(e) = \|v - w\|$.

We can recognize that $dist_{NN}(\cdot)$ violates non-negativity and triangle inequality, which should be held for the metric space. For this reason, the conventional k-d tree or G-NAT [19] can show sub-optimal performance for near neighbor search in volumetric tree*. We hence use NMSlib [20], a proximity-graph based approximate near neighbor search library for generic non-metric spaces for efficient performance.

Fig. 1 shows an example of the constructed graph. We can observe that the volumetric tree* covers $\mathcal{X}_{free}$ adaptively with

a fewer number of vertices, while the conventional planner maintains a set of tremendous vertices in the same time budget. We further discuss the comparison of the cardinality of graph components in Sec. IV with Table I.

We also maintain a subset of G, $\mathcal{T} = \{V^{\mathcal{T}}, E^{\mathcal{T}} \subset E\}$ for the shortest path computation, where $e \in E^{\mathcal{T}}$ is a set of edges on the shortest paths to all $v \in V^{\mathcal{T}}$ from x_{init} that is constructed by DSPT (Dynmic Shortest Path Tree) [6]. The search tree $\mathcal{T}$ consisting of volumetric vertices $V^{\mathcal{T}}$ in the end contains the solution path we are seeking, and is also used for the homotopy exploration with the dropout technique explained in Sec. III-D.

Overall process. The overall process is depicted in Alg. 1. The proposed volumetric tree* is based on Dancing PRM* [13] in terms of the graph construction and configuration-free space approximation. To be specific, volumetric tree* inherits the lazy collision checking with DSPT and witness propagation with radius compensation from Dancing PRM* for the precomputation-free approximation. Accordingly, we allow G to have edges in collision to reduce unnecessary edge collision checkings and check lazily if they are necessary.

During the iteration in Alg. 1, a collision-free x_{sample} is checked for a inclusion test with its near neighbor set V_{near}, not to fall inside of the hyper-sphere volumes associated with V (Alg. 1, Line: 6). On the other hand, for a $x_{sample} \in \mathcal{X}_{obs}$, we exploit the sample to improve the configuration-free approximation (Alg. 1, Line: 13-14). In $UpdatecFreeSpace$, we update $r_{v_{near}}$ if $\left\|v_{near} - x_{sample}\right\|$ is smaller than $r_{v_{near}}$ to trim the volumes by updating $o_{x_{sample}}$. Likewise, $PropagateCFreeSpace(\cdot)$ initializes $r_{x_{sample}}$ to be $\text{argmin}_{v \in V_{near}}(\left\|o_v - x_{sample}\right\|)$ and then updates r_v to be $\min(\left\|o_{x_{sample}} - v_{near}\right\|, r_{v_{near}})$ for all $v \in V_{near}$ to propagate the empirical collision information locally. For the details, refer to the original work [13].

$UpdateShortestPath(\cdot)$ updates $\mathcal{T}$ to maintain shortest paths for all possible destinations, i.e., $(v \in V^{\mathcal{T}}, x_{goal})$ dynamically. Finally, *BetterPathFound* (Alg. 1, Line: 15) checks whether the cost of the best-so-far path $c(\sigma_{new})$ is updated in $UpdateDSPT(\cdot)$ at this iteration. If so, we lazily check the feasibility of σ_{new} and then $OptimizePath(\cdot)$ (Alg 1, Line: 16) refines the σ_{new} towards the local optimal path, which is discussed in the subsequent subsection.

C. Path Optimization

Our path optimization is based on CHOMP [8], which uses gradient descent techniques to optimize a motion trajectory iteratively.

The original objective function related to the obstacle cost, f_{obs}, contains so-called obstacle potential terms, which can be expressed as a vector to the closest obstacle or the obstacle proximity, usually obtained by the Euclidean distance transformation in the workspace. It however requires additional precomputation and model simplification such as swept-sphere technique and kinematic Jacobian [8]. For this reason, our objective function is designed to be free from such dependencies to achieve a higher applicability and seamless integration with existing sampling-based planners.

At a high level, our objective function used for volumetric tree* is identical to the original form [8]:

$$\mathcal{U}(\xi) = f_{prior}(\xi) + f_{obs}(\xi), \tag{4}$$

where $\xi : [1, n \in \mathbb{N}] \to \mathcal{X}$ is a discrete path consisting of n equidistant intermediate configurations along σ_{new}, i.e., $\xi \in \mathbb{R}^{n \times d}$, and f_{prior} is a sum of squared derivatives, i.e., $\frac{1}{2}\xi^T A \xi + \xi^T b$. $A \in \mathbb{R}^{n \times n}$ and $b \in \mathbb{R}^{n \times d}$ are available in [8].

The obstacle cost for our work is formulated as follows:

$$f_{obs}(\xi) = \sum_{i=1}^{n} \omega(\|\xi^*(i) - \xi(i)\|) \left\| \frac{d}{dt}\xi(i) \right\|, \tag{5}$$

where $\omega(\cdot)$ is a user-defined weighting function; we simply use the identify function $\omega(x) = x$ and $\xi^*(t)$ is the last configuration of $\xi(i)$ without collision. Unlike the problem CHOMP and its variants aim to solve, we can assume that the given initial path σ_{new} is collision-free, and thus that $\xi^*(i)$ exists by initializing ξ^* as σ_{new} during the iteration; therefore, $\xi^*(i)$ can be considered as the closest empirical collision-free state of $\xi(i)$.

The update rule follows the iterative quasi-Newton approach like CHOMP, which can be written as:

$$\xi_{i+1} = \xi_i - \nabla\mathcal{U}(\xi_i), \tag{6}$$

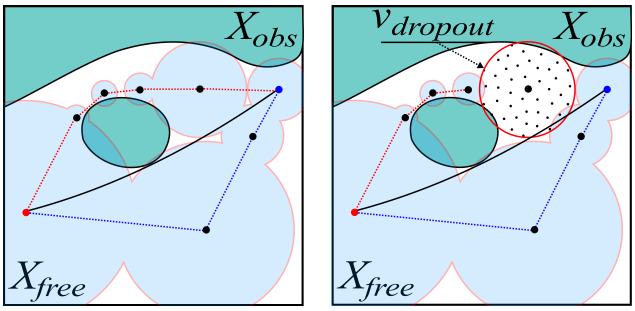
where $\nabla\mathcal{U}(\cdot) \in \mathbb{R}^{n \times d}$ is:

$$\nabla\mathcal{U}(\xi) = \eta A^{-1}(\lambda(A\xi + B) + \nabla f_{obs}(\xi)). \tag{7}$$

In the above equation, η is a user-defined convergence rate, set to be 1, $(A\xi + B)$ is ∇f_{prior} in the expanded form, and λ is a trade-off parameter for the smoothness against the obstacle avoidance, set to be $\frac{n}{2}$, where $n = 50$ is the number of intermediate nodes. The number of iterative optimization of Eq. 6 is fixed to 50 in our implementation. Lastly, ∇f_{obs} is formulated as:

$$\begin{aligned}\nabla f_{obs}(\xi(i)) = \left\|\xi(i)'\right\| (I - \widehat{\xi(i)'}\widehat{\xi(i)'}^T)(\xi(\widehat{i)^* - \xi(i)}) \\ - \|\xi(i)^* - \xi(i)\| \kappa,\end{aligned} \tag{8}$$

where $\xi(i)'$ is the derivative of $\xi(i)$, $\widehat{\xi(i)}$ stands for the normalizing function, i.e., $\hat{x} = \frac{\xi(i)}{\|\xi(i)\|}$, and κ is the curvature of ξ at $\xi(i)$. We also replace the obstacle potential terms used for CHOMP with a function of $\xi(i)^* - \xi(i)$, accordingly to Eq. 5. The underlying meaning of Eq. 8 is that when $\xi(i) \in \mathcal{X}_{obs}$, we push $\xi(i)$ with collision toward its last empirical C-free state $\xi(i)^*$ as a roll-back. In CHOMP, this type of approach is not applicable since a given initial path is assumed in collision, which demands a local workspace obstacle analysis in advance, e.g., the signed distance field, to compute the obstacle potentials efficiently. Our approach, therefore, gets rid of such dependencies and makes the optimization process even intuitive, while combining sampling-based and optimization-based planning seamlessly. Note that $\xi(i) \in \mathcal{X}_{free}$ makes $\nabla f_{obs} = 0$.



(a) (b)

Fig. 3: Naïve shortest path computation can miss to explore a solution path even homotopic to the optimal solution path (the blue dotted one), due to a sparse graph structure. We address this issue by using the dropout of vertices in solution paths observed during the execution. The dotted-filled ball in the right figure stands for an excluded vertex, $v_{dropout}$. The search tree constructed without $V_{dropout}$ allows our approach to find other solution paths (the blue dotted one) that can be homotopic to the optimal solution, the solid black line.

For feasibility of the updated path, it is necessary to check ξ for collision and update ξ^* prior to performing the gradient update in Eq. 6 at each iteration. For each $\xi(i) \in \mathcal{X}_{obs}$ found during the optimization process is also used to improve our configuration-free space approximation (Sec. III-B). For a set of vertices along the path, i.e., $V_\sigma = \{v | v \in \sigma\}$, we perform $UpdateCFreeSpace(V_\sigma, \xi(i))$ (Alg. 1, Line: 14), which assures that each $o_{v \in \sigma}$ is set to the closest empirical collision found locally.

D. Shortest Path Computation with Dropout

As the number of iteration increases, volumetric tree* attempts to optimize multiple best-so-far paths as this method is also based on the sampling-based approach. Nonetheless, we found a technical challenge that arises due to the sparse graph structure with hyper-sphere volumes.

The left figure in Fig. 3 shows two different paths; one containing a vertex associated with a large volume, σ_{blue} (dotted blue), and the other, σ_{red} (dotted red), with a relatively lower cost, i.e., $c(\sigma_{red}) < c(\sigma_{blue})$. If the σ_{red} is found prior to the σ_{blue}, $UpdateDSPT(\cdot)$ (Alg. 1, Line: 14) can fail to return σ_{blue}, which is homotopic to the optimal solution path (the solid black line).

To ensure that volumetric tree* finds all possible homotopy classes of solution paths given the aforementioned challenge, we propose to use a path optimization with *dropout*, which is a randomized vertex exclusion procedure. The core concept is motivated by the Yen's algorithm [21] designed for finding the loop-less k-th shortest path for a graph.

Our dropout technique is summarized as follows:

1) Whenever a solution path σ_{new} is to be optimized in $OptimizePath(\cdot)$ (Alg. 1, Line: 16), insert σ_{new} to Σ_{new}, a set of all solution paths found so far, and insert all vertices in σ_{new} to $V_{\Sigma_{new}}$, a set of all vertices in Σ_{new}.
2) In $UpdateDSPT(\cdot)$ (Alg. 1, Line: 11), compute a set of excluded vertices, $V_{dropout} \subset V$, at a probability (Eq. 9) from V prior to updating $\mathcal{T}$.
3) Update $\mathcal{T}$ with $V \setminus V_{dropout}$ to compute a new shortest path, σ'_{new} from x_{init} to x_{goal}.
4) With dropout, $BetterPathFound(\cdot)$ (Alg. 1, Line: 15) behaves differently; it returns *true* if σ'_{new} has not been observed previously, i.e., $\sigma'_{new} \notin \Sigma_{new}$.
5) Empty $V_{dropout}$ and repeat the iteration.

Note that our approach is to find a solution paths in unrevealed homotopy classes, while the original Yen's algorithm is for finding the next, i.e., $k+1$-th shortest path by excluding an edge in k-th shortest path.

Fig. 3(b) shows an example of the shortest path computation with dropout. Suppose that a vertex associated with the dotted-filled ball is excluded out by dropout, which allows the planner to find σ_{new} as the blue dotted path. Consequently, volumetric tree* with dropout is capable of finding a solution path homotopic to the optimum even with the coarse-grained search graph G.

To realize the dropout approach, we record vertices that have been involved in any σ_{new} previously found, and then randomly exclude recorded ones, followed by updating our search tree $\mathcal{T}$ with remaining vertices, The dropout probability for a vertex v can be defined as follows:

$$\mathbb{P}[v \in V_{dropout}] = \begin{cases} c_{dropout} \frac{1}{|V_{\Sigma_{new}}|} & \text{if } v \in V_{\Sigma_{new}}, \\ 0 & \text{otherwise,} \end{cases} \quad (9)$$

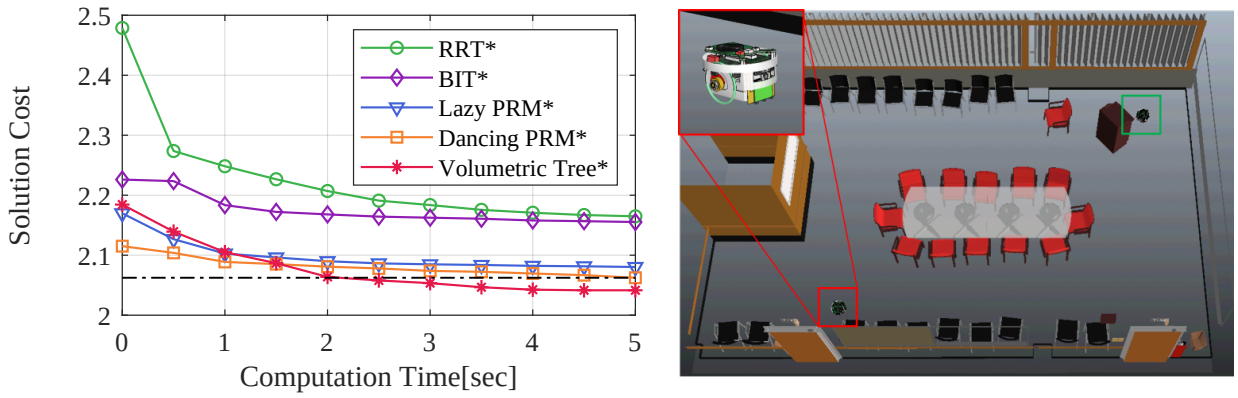
where $c_{dropout}$ is a user-tuned dropping-out parameter and set to be 1.

As an alternative to our dropout approach, one can consider prioritizing a candidate path σ over G. However, this alternative is difficult to be realized, since for an arbitrary vertex $v \in V$, both cost-to-come from x_{init} and cost-to-go to x_{goal} are unknown in advance. Moreover, using an admissible estimator, e.g., the Euclidean distance in the simplest form, can require an excessive amount of computations due to its weak bound.

IV. Experiment

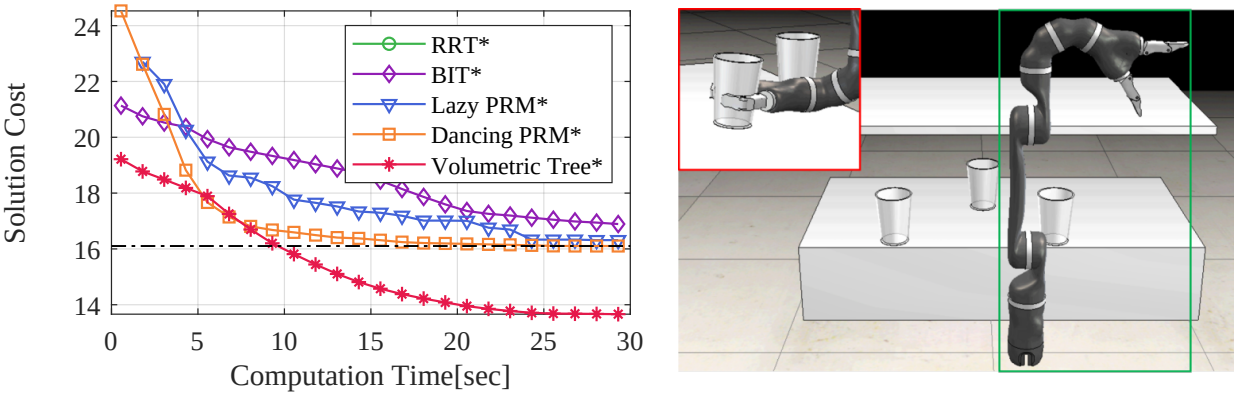
We compare the performance of volumetric tree* against the other almost-sure asymptotic optimal sampling-based planners: RRT* [1], Lazy PRM* [4], BIT* [7], and Dancing PRM* [13]. Volumetric tree* is implemented using OMPL (Open Motion Planning Library) [22] including CHOMP-based path optimizer and DSPT (Dynamic Shortest Path Tree). Volumetric tree* uses NMSlib [20] for nearest neighbor search (Sec. III-B). Other planners use G-NAT [19], which is available in OMPL.

We test our method with a 2-DoF rigid body planning (Fig. 4) and 6-DoF manipulation problem (Fig. 5) using V-REP simulator [23]. We also perform evaluations in two synthetic environments ($\mathbb{R}^2$ and $\mathbb{R}^8$) shown in Fig. 6 to illustrate different behaviors clearly, and a real robot experiment using Hubo with 7-DoF Manipulator [24], which can be found in the video attachment to show the practical benefits.



(a) (b)

Fig. 4: A 2-DoF mobile robot planning problem in a conference room with narrow passages under the chairs, resulting in difficult-to-sample homotopies and surrounding wide-open areas. x_{init} and x_{goal} are depicted in the green and red boxes, respectively.



(a) (b)

Fig. 5: A 6-DoF manipulation planning problem. x_{goal} (in the red box) is a pose of grasping the cup in the middle of the table, avoiding the other cups and the ceiling from x_{init} (in the green box).

Our results are averaged over 30 trials and the time budgets are set to 5, 30, 1 and 10 seconds for 2-DoF rigid body, 6-DoF manipulation problem, $\mathbb{R}^2$, and $\mathbb{R}^8$, respectively. The detail of the problem settings can be seen in the attached video.

Fig. 4 and 5 show 2-DoF rigid body and 6-DoF manipulation planning problems, respectively. In these scenes, planners spend much time, 20% to 80% of the total computation, on collision checking. For our method, collision checking takes about 30% to 50% of the total computation time. For the 2-DoF rigid body problem, the computation time on optimization, $T(OPT)$, becomes the major computation bottleneck in volumetric tree* due to a number of explicit collision checkings during the optimizations. On the other hand, for the 6-DoF manipulation problem, collision checking, $T(CC)$, becomes the main computation bottleneck for all the planners, which is caused by the higher complexity of workspace, i.e., the number of triangles.

Volumetric tree*, nevertheless, outperforms the other planners by reducing the number of vertices ($|V|$) by an order of magnitude. It can be interpreted as that our adaptive sparse graph efficiently captures the homotopy of solution paths, while achieving a better quality of solution paths by the integration with optimization-based planning.

In 6-DoF manipulation planning (Fig. 5), the computational portion of $T(NN)$ becomes comparable to $T(CC)$ in both Lazy PRM* and Dancing PRM*, which also use lazy collision checking. On the other hand, volumetric tree* achieves a noticeable improvement with the fewer number of vertices and the reduced overhead of NN, despite its higher $T(OPT)$. It is because that volumetric tree* efficiently exploits the solution path using optimization-based planning rather than constructing a dense graph to explore the configuration space. Its benefit is expected to go higher as we have a higher dimensional problem since it is required to have denser graphs in that higher space for other methods.

Discussion. We also provide results with synthetic scenes for in-depth analysis. The synthetic benchmarks are designed to

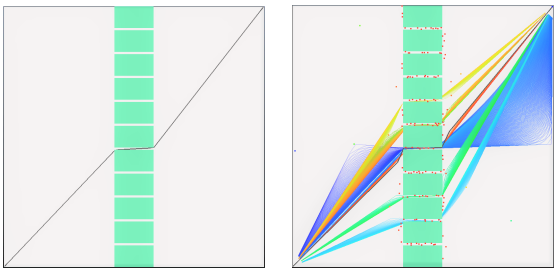


Fig. 6: 2D synthetic benchmark with 10 narrow gaps located in the middle of the hyper-cube. In the left figure, green squares indicate the obstacles and the black lines are the optimal solution path. The right figure shows intermediate configurations (colored differently) during the optimization iterations. The dots indicate vertices of our sparse graph, colored according to their radii (red for smaller values).

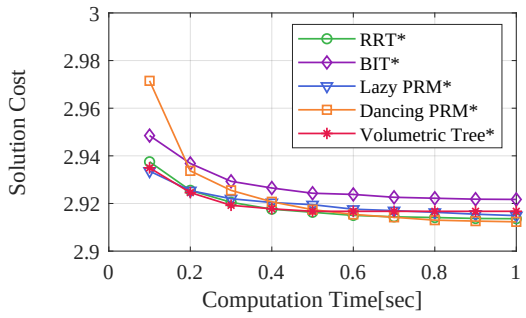
have the narrow passage with multiple homotopy classes of the solution path. It can also be considered *NN-sensitve* [15]; the computational cost of nearest neighbor search is not negligible, since that of collision checking is relatively lighter due to the simple geometry. In both $\mathbb{R}^2$ and $\mathbb{R}^8$, x_{init} and x_{goal} are set to $[-1,...-1]^d$ and $[1,...,1]^d$, respectively, in a hyper-cube of width 2, and the narrow gaps are equidistantly located in the middle wall; the gap has a height of $\frac{1}{6}$ and i-th obstacle is defined by two diagonal points, $[-0.15, i(gap+\frac{gap}{10}),-1,\ldots,-1]$ and $[0.15, i\times(gap+\frac{gap}{10},1,\ldots,1)] \in \mathbb{R}^d$.

Fig. 7 shows the solution cost as a function of computation time measured in synthetic benchmarks and the corresponding statistics are organized in Table I. In $\mathbb{R}^2$, we can observe some of the other planners outperform volumetric tree*. While volumetric tree* identifies the homotopy class of the optimal solution path and optimizes solution paths toward a local optimum within the finite number of optimization iterations, it may result in near-optimal paths depending on the optimizer parameters, yet the performance gap is $<1\%$ in terms of the final solution cost. This issue can show a slowdown for our method, especially for simple, lower dimensional problems. Adopting advanced optimization techniques or automatically tuning the parameters depending on the problem are left for future work.

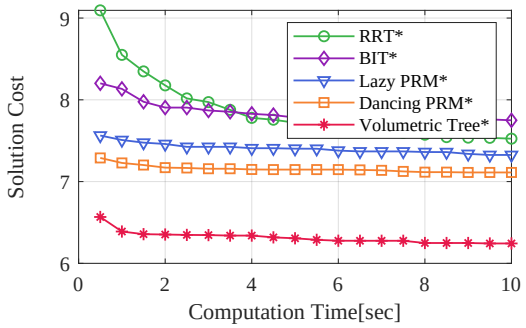
For the case of $\mathbb{R}^8$, volumetric tree* shows exceptional performance improvement. As we can observe in Table I, volumetric tree* checks more vertices ($|VC|$), which can provide a better understanding of the given space, while keeping a fewer number of vertices thanks to our adaptive sparse graph construction with C-free approximation. Furthermore, the hybridization with optimization-based planning allows volumetric tree* not to rely only on sampling for the convergence of solution paths, resulting in a better performance, especially in a higher dimensional problem.

V. CONCLUSION

In this work, we have presented a hybridization of sampling-based and optimization-based planning named volumetric tree*. Our approach constructs a random geometric graph, where each vertex is associated with a hypersphere volume with the C-free approximation, while rejecting other samples falling into the space occupied by the existing volumes, resulting in a sparse graph. Volumetric tree* identifies all possible homotopy classes of solution paths with the dropout technique and refines solution paths found during the execution toward the local optimum using optimization-based planning. Our experiment results have shown meaningful performance improvement in most tested environments, showing higher robustness compared to the other tested methods.



(a) $\mathbb{R}^2$



(b) $\mathbb{R}^8$

Fig. 7: Performance comparison over computation time for different algorithms in synthetic benchmarks.

There are many interesting research directions. When it comes to sampling, volumetric tree* simply relies on the sample rejection for the adaptive sparse graph construction, which can be inefficient, especially in a higher dimensional space [25]. For this reason, it is worth studying to design a dedicated sampler that explores a promising area efficiently.

ACKNOWLEDGEMENT

We appreciate the anonymous reviewers for constructive comments and insightful suggestions, and for the researchers whose contributions have inspired our work. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT, No. 2019R1A2C3002833).

REFERENCES

[1] J. Kuffner and S.M. LaValle, "RRT-connect: An efficient approach to single-query path planning", in *IEEE Int'l. Conf. on Robotics and Automation*, 2000, pp. 995–1001.

TABLE I: Statistics of experiment results.

Benchmark	Planner	Final Cost	$\|V\|$	$\|E\|$	$\|VC\|$	$\|EC\|$	$T(CC)$	$T(NN)$	$T(SP)$	$T(OPT)$
2D Rigid body	RRT*	2.164	5363	185276	7239	22163	97%	<1%	N/A	N/A
	BIT*	2.156	10102	34220	12604	4506	47%	52%	N/A	N/A
	Lazy PRM*	2.081	16274	621267	20297	3825	24%	8%	55%	N/A
	Dancing PRM*	2.062	14834	560427	18553	3562	20%	7%	46%	11%
	Volumetric Tree*	**2.040**	540	11808	13688	1195	29%	2%	<1%	67%
6D Manipulation	RRT*	-	1316	29051	2543	5314	99%	<1%	N/A	N/A
	BIT*	16.799	33588	6336	53064	1179	83%	16%	N/A	N/A
	Lazy PRM*	16.315	24259	771850	38309	265	50%	48%	<1%	N/A
	Dancing PRM*	16.058	22401	706865	35400	335	47%	39%	<1%	1%
	Volumetric Tree*	**13.646**	6242	169534	40788	306	48%	27%	<1%	22%
2D Hyper-cube	RRT*	2.914	44493	1958254	51846	144975	13%	72%	N/A	N/A
	BIT*	2.922	4102	23656	4926	7531	2%	82%	N/A	<1%
	Lazy PRM*	2.916	16550	634554	19200	1885	1%	27%	56%	N/A
	Dancing PRM*	**2.913**	11057	404458	12810	1618	2%	15%	50%	11%
	Volumetric Tree*	2.917	353	6862	62406	935	3%	86%	<1%	3%
8D Hyper-cube	RRT*	7.525	29536	938437	35548	132745	4%	95%	N/A	N/A
	BIT*	7.750	10791	64296	12502	16637	<1%	97%	N/A	N/A
	Lazy PRM*	7.326	25227	779203	29262	1404	<1%	97%	<1%	N/A
	Dancing PRM*	7.156	24737	762916	28670	801	<1%	86%	<1%	3%
	Volumetric Tree*	**6.243**	5050	128114	61409	1547	4%	90%	<1%	<1%

$|V|$ and $|E|$ are the cardinality of the vertex set V and edge set E, respectively. $|VC|$ and $|EC|$ stand for the number of vertex and edge collision checking, and $T(\cdot)$ is a computation time ratio of each component: CC = Collision Checking, NN = Nearest Neighbor search, SP = Shortest Path computation in DSPT and OPT = OPTimization. Note that the collision checking time during the optimization is also included in $T(OPT)$ and the number of NN-query is identical to $|VC|$ in volumetric tree*. In 6D manipulation problem, the result of RRT* is not reported due to a high ratio ($> 90\%$) of unsuccessful attempts.

[2] LE Kavraki, P Svestka, J-C Latombe, and MH 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.

[3] Sertac Karaman and Emilio Frazzoli, "Sampling-based algorithms for optimal motion planning", *Int'l. Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[4] Kris Hauser, "Lazy collision checking in asymptotically-optimal motion planning", in *IEEE Int'l. Conf. on Robotics and Automation*, 2015.

[5] Joshua Bialkowski, Michael Otte, Sertac Karaman, and Emilio Frazzoli, "Efficient collision checking in sampling-based motion planning via safety certificates", *Int'l. Journal of Robotics Research*, vol. 35, no. 7, pp. 767–796, 2016.

[6] Daniele Frigioni, Alberto Marchetti-Spaccamela, and Umberto Nanni, "Fully dynamic algorithms for maintaining shortest paths trees", *Journal of Algorithms*, vol. 34, no. 2, pp. 251–281, 2000.

[7] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot, "Batch informed trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs", in *IEEE Int'l. Conf. on Robotics and Automation*, 2015, pp. 3067–3074.

[8] Matt Zucker, Nathan Ratliff, Anca D Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M Dellin, J Andrew Bagnell, and Siddhartha S Srinivasa, "CHOMP: Covariant hamiltonian optimization for motion planning", *Int'l. Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1164–1193, 2013.

[9] Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal, "STOMP: Stochastic trajectory optimization for motion planning", in *IEEE Int'l. Conf. on Robotics and Automation*, 2011, pp. 4569–4574.

[10] Chonhyon Park, Jia Pan, and Dinesh Manocha, "ITOMP: Incremental trajectory optimization for real-time replanning in dynamic environments.", in *Int'l. Conf. on Automated Planning and Scheduling*, 2012.

[11] John Schulman, Jonathan Ho, Alex X Lee, Ibrahim Awwal, Henry Bradlow, and Pieter Abbeel, "Finding locally optimal, collision-free trajectories with sequential convex optimization.", in *Robotics: science and systems*, 2013.

[12] Sanjiban Choudhury, Jonathan D Gammell, Timothy D Barfoot, Siddhartha S Srinivasa, and Sebastian Scherer, "Regionally accelerated batch informed trees (RABIT*): A framework to integrate local information into optimal path planning", in *IEEE Int'l. Conf. on Robotics and Automation*, 2016, pp. 4207–4214.

[13] Donghyuk Kim, Youngsun Kwon, and Sung-Eui Yoon, "Dancing PRM*: Simultaneous planning of sampling and optimization with configuration free space approximation", in *IEEE Int'l. Conf. on Robotics and Automation*, 2018, pp. 7071–7078.

[14] Alan Kuntz, Chris Bowen, and Ron Alterovitz, "Fast anytime motion planning in point clouds by interleaving sampling and interior point optimization", *Int'l. Symposium on Robotics Research*, 2017.

[15] Michal Kleinbort, Oren Salzman, and Dan Halperin, "Collision detection or nearest-neighbor search? on the computational bottleneck in sampling-based motion planning", *Int'l. Workshop on the Algorithmic Foundations of Robotics*, 2016.

[16] Valerio Varricchio and Emilio Frazzoli, "Asymptotically optimal pruning for nonholonomic nearest-neighbor search", in *IEEE Conf. on Control and Decision*, 2018, pp. 4459–4466.

[17] A. Shkolnik and R. Tedrake, "Sample-based planning with volumes in configuration space", *arXiv preprint arXiv:1109.3145*, 2011.

[18] Sean Quinlan and Oussama Khatib, "Elastic bands: Connecting path planning and control", in *IEEE Int'l. Conf. on Robotics and Automation*, 1993, pp. 802–807.

[19] Sergey Brin, "Near neighbor search in large metric spaces", in *International Conference on Very Large Data Bases*, 1995.

[20] Leonid Boytsov and Bilegsaikhan Naidan, "Engineering efficient and effective non-metric space library", in *Int'l Conf. on Similarity Search and Applications*, 2013, pp. 280–293.

[21] Jin Y Yen, "Finding the k shortest loopless paths in a network", *management Science*, vol. 17, no. 11, pp. 712–716, 1971.

[22] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki, "The Open Motion Planning Library", *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, 2012, `http://ompl.kavrakilab.org`.

[23] M. Freese E. Rohmer, S. P. N. Singh, "V-REP: a versatile and scalable robot simulation framework", in *IEEE/RSJ Int'l. Conf. on Intelligent Robots and Systems*, 2013.

[24] M. Lee, Y. Heo, J. Park, H. Yang, P. Benz, H. Jang, H. Park, I. Kweon, and J. Oh, "Fast perception, planning, and execution for a robotic butler: Wheeled humanoid m-hubo", in *IEEE/RSJ Int'l. Conf. on Intelligent Robots and Systems*, 2019.

[25] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot, "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic", in *IEEE/RSJ Int'l. Conf. on Intelligent Robots and Systems*, 2014, pp. 2997–3004.