COVER: COverage-VErified Roadmaps for Fixed-time Motion Planning in Continuous Semi-Static Environments

Niranjan Kumar Ilampooranan, Constantinos Chamzas

Abstract— Having the ability to answer motion-planning queries within a fixed time budget is critical for the widespread deployment of robotic systems. Semi-static environments—where most obstacles remain static but a limited set can vary across queries, such as bins or packages in warehouses—exhibit structured variability that can be systematically exploited to provide stronger guarantees than in general motion-planning problems. However, prior approaches in this setting either lack formal guarantees or rely on restrictive discretizations of obstacle configurations, limiting their applicability in realistic domains. This paper introduces COVER, a novel framework that incrementally constructs a coverage-verified roadmap in semi-static environments. By partitioning the obstacle configuration space and solving for feasible paths within each partition, COVER systematically verifies feasibility of the roadmap in each partition and guarantees fixed-time motion planning queries within the verified regions. We validate COVER with a 7-DoF simulated Panda robot performing table and shelf tasks, demonstrating that COVER achieves broader coverage with higher query success rates than prior works.

I. INTRODUCTION

Motion planning [1] is a core function of robotics, enabling manipulators to plan collision-free, reliable trajectories in complex workspaces. In industrial domains, robots often operate in semi-static environments—settings where the majority of the workspace remains fixed, while smaller obstacles such as bins and packages vary across tasks. These environments are characterized by the repetitive nature of the tasks performed, such as moving similar objects between shelves or tables. In such scenarios, robots must generate motions within strict time budgets while minimizing downtime.

For instance, consider the scenario illustrated in Fig. 1, where a robot must retrieve a cylindrical object from a shelf while two additional movable obstacles can occupy arbitrary positions within the same shelf region. Such semi-static variability frequently arises in real-world settings, particularly in warehouse and industrial environments, where bins and packages are repeatedly rearranged between tasks.

Preprocessing-based methods exploit this structured variability to accelerate query-time planning and, in some cases, achieve fixed-time performance. Experience-based frameworks [2]–[5] improve efficiency by reusing prior solutions but provide no coverage guarantees—that is, they cannot certify whether a valid obstacle arrangement will admit a solution. Fixed-time motion planners [6], [7] do provide formal guarantees, but require restrictive assumptions such as discretizing the obstacle configuration space and assuming all movable obstacles to be identical. These assumptions limit

All authors are affiliated with the Department of Robotics Engineering, Worcester Polytechnic Institute (WPI), Worcester, MA 01609, USA {nilampooranan, cchamzas} @ wpi.edu.
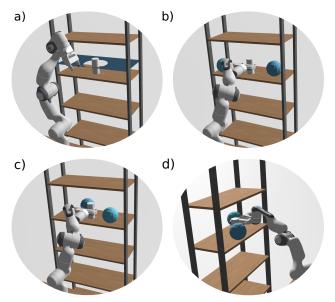
Fig. 1: **(a)** Semi-static motion-planning problem: the robot must grasp a white cylinder while two obstacles may lie anywhere in the dark-blue region. **(b)**–**(d)**) Three sampled instances. COVER guarantees that for all covered placements, a valid motion plan can be retrieved.

their applicability to realistic domains, where obstacles vary continuously in both size and placement.

In this work, we propose COVER, a roadmap-coverage framework within the fixed-time motion planning domain that extends beyond discretized formulations to handle continuous problem spaces while still affording fixed-time guarantees. Given a semi-static environment represented as a continuous set of possible obstacle placements, COVER incrementally constructs a roadmap in the robot's configuration space. By explicitly associating robot motions on the roadmap with the corresponding obstacle configurations that invalidate them, COVER systematically verifies coverage of the problem space. This enables fixed-time query resolution even when the obstacle configurations are defined in a continuous domain.

We evaluate COVER in simulation using a 7-DoF manipulator performing object-picking tasks from a shelf and a table. Our experiments show that COVER achieves broader coverage in continuous domains than prior methods, while maintaining reasonable preprocessing times. COVER naturally extends to heterogeneous obstacle sizes, demonstrating its flexibility beyond uniform setups.

The contributions of this work are threefold: (i) we introduce the COVER roadmap-coverage framework that provides fixed query-time guarantees in continuous obstacle spaces, (ii) demonstrate its ability to handle obstacles of varying sizes and continuous placements, and (iii) develop a problem-space coverage estimator that systematically verifies the exact

coverage ratio of precomputed paths or roadmaps.

II. RELATED WORK

There have been several motion planning approaches that leverage preprocessing to accelerate query-time performance. The most well-known are roadmap planners [8]–[10], which construct a roadmap of collision-free configurations during an offline phase and then answer multiple start–goal queries on the same graph. While highly effective in static environments, these methods degrade when obstacles change, as the roadmap may no longer reflect the true free space and must be recomputed or repaired.

To address this limitation, the class of dynamic roadmaps was proposed. Dynamic roadmap methods extend multi-query planning to environments that change between queries by maintaining mappings from workspace regions to configuration-space edges [11]. When a new environment is encountered, these mappings allow for rapid invalidation of affected edges, enabling efficient online updates. This has been implemented using hashing techniques for fast collision detection [12], [13] or hierarchical decompositions for scalable updates [14]. Although highly efficient, these methods are designed for arbitrary dynamic environments—any workspace subregion can potentially be occupied — and therefore do not target the structured variability of semi-static settings. As a result, they emphasize fast updates rather than certifying coverage.

In the class of learning-based methods, experience-based planners accelerate planning by reusing information from previously solved problems. This information may take the form of a library of paths [3], [5], a roadmap enriched with prior experience [2], [15], or a collection of local samplers that bias future queries [4], [16]. These approaches generally assume that planning problems are drawn from an unknown distribution and attempt to maintain a representative set of solutions or sampling strategies that can be adapted to new queries [5], [17]. While effective in practice, they do not explicitly exploit the structured variability of semi-static environments. Instead, they focus on statistical generalization across distributions of problems, without offering guarantees of full coverage in the space of obstacle configurations.

Fixed-time motion planning methods [6], [7] are most closely related to our work. These approaches specifically target semi-static environments by assuming known apriori distributions of movable obstacles and guarantee that queries can be answered within a fixed-time budget through extensive offline preprocessing. However, existing formulations discretize the obstacle arrangement space, which restricts their applicability in realistic scenarios where obstacles can vary continuously. Moreover, recent formulations [6] impose strong homogeneity assumptions, such as requiring all obstacles to occupy equivalent locations or share identical shapes, further limiting their use in heterogeneous settings.

III. PROBLEM STATEMENT AND NOTATION

A. Geometric Motion Planning

Let $q \in \mathcal{C}$ denote a configuration and configuration space ($\mathcal{C}$-space) of the robot. The robot configurations in collision with obstacles in the workspace is given by

$$\mathcal{C}_{\text{obs}} = \{q \in \mathcal{C} \mid R(q) \cap \mathcal{WO} \neq \emptyset\}$$

where $R(q) \subseteq \mathbb{R}^3$ is the workspace occupancy of the robot at configuration q, and $\mathcal{WO} \subseteq \mathbb{R}^3$ denotes the set of workspace obstacles.[1] Conversely, valid robot configurations in $\mathcal{C}$-space is defined as $\mathcal{C}_{\text{free}} = \mathcal{C} \setminus \mathcal{C}_{\text{obs}}$.

Given $q_{\text{start}} \in \mathcal{C}_{\text{free}}$ as the start configuration, $\mathcal{C}_{\text{goal}} \subseteq \mathcal{C}_{\text{free}}$ as the goal region[2], and $\mathcal{WO}$ as the workspace obstacles, a geometric motion-planning problem instance is specified by $p = (q_{\text{start}}, \mathcal{C}_{\text{goal}}, \mathcal{WO})$. The task is to find a continuous path $\pi : [0, 1] \rightarrow \mathcal{C}_{\text{free}}$ such that $\pi(0) = q_{\text{start}}$ and $\pi(1) \in \mathcal{C}_{\text{goal}}$, if one exists.

B. Geometric Motion Planning in Semi-Static Workspaces

A semi-static workspace contains two disjoint sets of obstacles: fixed (static) obstacles O_{f} and movable obstacles $O_{\text{m}} = \{o_1, o_2, \ldots, o_n\}$ whose configurations may vary across planning queries but remain fixed during execution. Each arrangement m of the movable obstacles from the space of possible arrangements $\mathcal{M}$ induces a distinct collision-free space, $\mathcal{C}_{\text{free}}(m)$ and thus a distinct geometric motion-planning instance given in Equation 1.

$$p(m) = (q_{\text{start}}, \mathcal{C}_{\text{goal}}, O_{\text{f}}, O_m(m)) \tag{1}$$

Here, $O_m(m)$ indicates the occupancy of the movable obstacles after assuming an arrangement $m \in \mathcal{M}$. Now, given a start configuration q_s, and a goal region, $\mathcal{C}_{\text{goal}}$ the *semi-static motion-planning problem* is to find, for every $m \in \mathcal{M}$, a continuous path $\pi_m : [0, 1] \rightarrow \mathcal{C}_{\text{free}}(m)$ such that $\pi_m(0) = q_{\text{start}}$ and $\pi_m(1) \in \mathcal{C}_{\text{goal}}$, if one exists.

IV. METHODOLOGY

Exhaustively solving the motion-planning problems for every possible arrangement $m \in \mathcal{M}$ for a semi-static workspace is infeasible, since the space of movable obstacle configurations is continuous and thereby, infinite. Instead, COVER builds on the key intuition that a single solution path can remain valid across infinitely many obstacle arrangements. For instance, in Fig. 1 (b) and Fig. 1 (c), the same precomputed path is feasible despite variations in obstacle placement. Leveraging this observation, COVER introduces a framework for exactly quantifying the portion of $\mathcal{M}$ that a given roadmap can solve, which we define as problem-space coverage.

Section IV-A formalizes this concept and outlines the key ideas behind the computation for roadmaps in semi-static environments. Building on this foundation, Section IV-B presents our coverage-informed roadmap construction algorithm, which incrementally maximizes problem-space coverage by systematically addressing uncovered regions of the obstacle configuration space. Finally, in Section IV-C, we describe how the resulting COVER roadmap enables fixed-time query resolution for arbitrary arrangements of movable obstacles.

---

[1]Self-collisions or kinematic constraints can be encoded similarly.
[2]In our setting the goal region is always a finite set of configurations.

A. Problem Space Coverage Estimation

Formally, a *roadmap* is defined as a graph $\mathcal{G} = (V, E)$ defined in the configuration space $\mathcal{C}_{\text{free}}$ of the robot [18]. Each vertex $v \in V$ corresponds to a robot configuration $q \in \mathcal{C}_{\text{free}}$, while each edge $e \in E$ represents a collision-free local (straight-edge) path connecting two vertices. Roadmaps provide a compact representation of paths and form the basis of multi-query motion planning. [2], [8], [19].

In static environments, where $\mathcal{C}_{\text{free}}$ remains unchanged, the validity of roadmap vertices and edges persists, ensuring that any previously feasible path between a start configuration q_{start} and goal region $\mathcal{C}_{\text{goal}}$ remains valid. However, in semi-static environments, $\mathcal{C}_{\text{free}}$ (m) depends on the specific arrangement of movable obstacles O_{m}. For a given roadmap $\mathcal{G}$, we define $\mathcal{M}_{\text{cov}}$ as the subset of arrangements that always admit a collision-free path from q_{start} to $\mathcal{C}_{\text{goal}}$. Its complement, $\mathcal{M}_{\text{uncov}} = \mathcal{M} \setminus \mathcal{M}_{\text{cov}}$, consists of arrangements that invalidate every collision-free path on $\mathcal{G}$.

Thus, the *problem-space coverage* of $\mathcal{G}$ (Equation 2) is defined as the proportion of obstacle arrangements, out of all possible arrangements in $\mathcal{M}$, that do not invalidate all paths.

$$\text{Coverage}(\mathcal{G}) = \frac{\text{Vol}(\mathcal{M}_{\text{cov}})}{\text{Vol}(\mathcal{M})}, \tag{2}$$

where $\text{Vol}(\cdot)$ denotes the measure of a subset of obstacle configurations. This metric captures the proportion of the semi-static problem space that is guaranteed to be solvable with the given roadmap. We note that it is possible that certain regions of $\mathcal{M}$ might generate infeasible problems [20].

To compute $\mathcal{M}_{\text{uncov}}$ for a given roadmap $\mathcal{G}$, we first construct a decomposition of $\mathcal{M}$ based on the swept volumes of the edges in $\mathcal{G}$ in Sec. IV-A.1, and then determine the decomposed partitions covered by $\mathcal{G}$ in Sec. IV-A.2.

Algorithm 1: PartitionObstacleSpace

Input: Roadmap $\mathcal{G} = (V, E)$, One movable obstacle o_i, obstacle configuration space $\mathcal{M}_i$
Output: Decomposition tree T_i for obstacle o_i

1 $\mathcal{E} \leftarrow \emptyset$
2 **foreach** $e \in E$ **do**
    // Compute swept volume of robot motion e
3      $SV \leftarrow \texttt{SweptVolume(e)}$
    // Locations of o_i that invalides edge e
4      $\mathcal{E} \leftarrow \mathcal{E} \cup \texttt{MinkowskiSum}(SV, o)$
    // Build Decomposition Tree for o_i
5 $T_i \leftarrow \texttt{BinarySpacePartitionTree}(\mathcal{E})$
6 **return** T

---

1) Partitioning the Obstacle Space: We begin by computing binary partitions of the configuration space for each edge of the roadmap induced by a single movable obstacle, denoted $\mathcal{M}_i$, which we represent with a decomposition tree T_i (a binary space partitioning tree), shown in Fig. 2. Each tree T_i compactly encodes how different placements of an obstacle o_i affect the validity of the edges[3] of $\mathcal{G}$.

---

[3]The decomposition trees are not restricted to edges. The same procedure in Alg. 1 can also be applied to start and goal vertices to identify obstacle configurations that invalidate them.
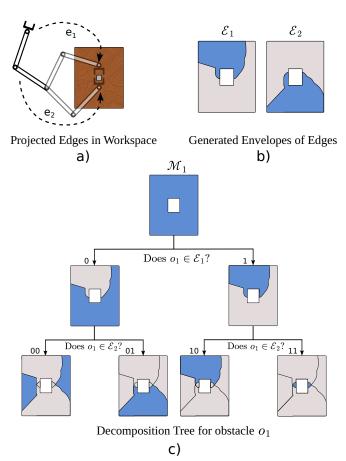

Fig. 2: **(a)**: Simple example with a roadmap containing two edges. Dashed lines indicate the approximate end-effector trajectory of the robot along each edge. **(b)**: Envelopes generated for the two edges. **(c)** Envelopes of the edges used for partitioning the obstacle configuration space of o_i into disjoint regions. These partitions are organized in a binary space partitioning tree, with each leaf corresponding to an equivalent binary signature.

The central idea is that every edge $e \in \mathcal{G}$ dichotomizes the obstacle configuration space $\mathcal{M}_i$ into two complementary subsets: those placements of o_i that invalidate e, and those that do not. To construct the partition, we follow the procedure in Alg. 1. In particular, lines 3–4 compute, for a specific obstacle o_i and edge e, the subset of configurations that render e invalid.

For each edge $e = (q_1, q_2) \in E$, we approximate the swept volume of the robot's motion between its endpoints as $SV = \bigcup_{q \in [q_1, q_2]} R(q)$, where $R(q) \subseteq W$ denotes the workspace occupied by the robot at configuration q. We then compute the Minkowski sum of o_i with SV, which yields the set of obstacle configurations $\mathcal{E}_i$ that cause a collision with the swept volume. We refer to these sets as *envelopes*, following [6]. Examples of $\mathcal{E}_1$ and $\mathcal{E}_2$ are shown in Fig. 2 (a).

The envelopes $\mathcal{E} = \{\mathcal{E}_1, \mathcal{E}_2, \ldots\}$ serve as splitting surfaces that recursively partition $\mathcal{M}_i$. The resulting decomposition tree subdivides $\mathcal{M}_i$ into disjoint regions, with the left child of each node representing placements that invalidate an edge and the right child representing placements where the edge remains valid. This subdivision yields a hierarchical representation of $\mathcal{M}_i$, where the leaves correspond to regions

of obstacle placements that exhibit *identical* edge-validity patterns, as illustrated in Fig. 2 (b).

The leaves of this decomposition tree naturally admit a binary encoding, where each leaf corresponds to a unique *signature*. Each signature records the edges of the roadmap obstructed when placing the obstacle within that region as a binary vector $\mathbf{b}_i \in \{0, 1\}^{|E|}$. For example, the signature `11` indicates that both edges are invalidated, whereas `00` implies that neither is affected. The j-th entry of $\mathbf{b}_i$ specifies whether edge $e_j \in E$ is blocked (1) or remains valid (0). This signature-based representation enables us to reason about obstacle placements algebraically rather than geometrically, forming the basis for combining partitions across multiple obstacles and ultimately identifying the covered and uncovered subsets of the arrangement space.

2) Calculating $\mathcal{M}_{uncov}$: We can now leverage these binary signatures from the decomposition trees to determine which obstacle arrangements render the entire roadmap invalid. This procedure is outlined in Alg. 2. When multiple movable obstacles are considered, the per-obstacle signatures are combined into a *composite signature* (line 6). Specifically, given signatures $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n \in \{0, 1\}^{|E|}$ for n obstacles, the composite signature is defined as

$$\mathbf{b} = \mathbf{b}_1 \vee \mathbf{b}_2 \vee \cdots \vee \mathbf{b}_n,$$

where $\vee$ denotes the element-wise logical OR. Thus, the j-th entry of $\mathbf{b}$ equals 1 if edge $e_j \in E$ is invalidated by at least one selected obstacle placement. Each composite signature corresponds to the subset of obstacle arrangements that invalidate the same set of edges.

Having obtained a compact decomposition of $\mathcal{M}$, we can now identify which partitions (i.e., obstacle arrangements) invalidate the roadmap $\mathcal{G}$. To this end, we construct a binary representation (lines 7–9) of all s–t paths in the roadmap $\Pi = \{\pi_1, \pi_2, \ldots, \pi_p\}$.

Each path $\pi_j \in \Pi$ can be similarly encoded by a binary vector $\mathbf{p}_j \in \{0, 1\}^{|E|}$, where the k-th entry equals 1 if edge $e_k \in E$ belongs to the path and 0 otherwise (lime 8). Collecting these vectors, we define the *path–edge incidence matrix*

$$\mathbf{P} = \begin{bmatrix} \mathbf{p}_1^\top \\ \mathbf{p}_2^\top \\ \vdots \\ \mathbf{p}_p^\top \end{bmatrix} \in \{0, 1\}^{p \times |E|}.$$

Now consider a composite signature $\mathbf{b} \in \{0, 1\}^{|\mathcal{E}|}$, where $\mathbf{b}_k = 1$ indicates that edge e_k is invalidated by the corresponding obstacle arrangement. To determine whether this arrangement invalidates all paths on the roadmap, we evaluate the dot product of column vector $\mathbf{b}$ with the path–edge incidence matrix as given in Equation 3.

$$\mathbf{v} = \mathbf{P}\mathbf{b}. \tag{3}$$

If $\mathbf{v}_j > 0$ for every path $\pi_j \in \Pi$, then all paths are blocked by at least one invalidated edge, and the region corresponding to $\mathbf{b}$ is classified as *uncovered*. Otherwise, if there exists some j with $\mathbf{v}_j = 0$, path π_j remains feasible in that region, the

---

Algorithm 2: DetectUncoveredPartitions

Input: Roadmap $G = (V, E)$,
      Movable Obstacles O_m

Output: Uncovered Obstacle Arrangements C

  // Get paths from the graph
1   $\Pi = \{\pi_1, \pi_2, \ldots\} = \texttt{DepthFirstSearch}(G)$
2   $T \leftarrow \emptyset$
  // Get decomposition tree for each obstacle
3   **foreach** $o_i, M_i \in O_m$ **do**
4     $T \leftarrow T \cup \texttt{PartitionObstacleSpace}(G, o_i)$
5   $C \leftarrow \texttt{GetAllCombinations}(T)$
6   $B \leftarrow \texttt{GenerateAllCompositeSignatures}(C)$
7   **foreach** $\pi_i \in \Pi$ **do**
    // Getting a vector that represents which edges constitute path
8     $p_i \leftarrow \texttt{BinaryVector}(\pi_i, E)$
    // Adding these vectors to path-edge incidence matrix
9     $P[i] \leftarrow p_i$
10   **foreach** $b \in B$ **do**
    // If dot product is non-zero, the combination of partitions invalidates π_i
11     $v[i] \leftarrow Pb$
12   **if** $v[i]$ *is not an all-ones vector* **then**
      // arrangements in $C[i]$ covered by $\mathcal{G}$
13   remove $C[i]$ from C

14 **return** C

---

roadmap continues to provide a collsion-free path (lines 12–13).

By iterating over all composite signatures and classifying each as covered or uncovered, we can explicitly compute $\mathcal{M}_{\text{cov}}$ and $\mathcal{M}_{\text{uncov}}$, which in turn yield the problem-space coverage defined in Equation 2.

Additionally, if $\mathcal{G}$ can be informed of arrangements in $\mathcal{M}_{\text{uncov}}$, efforts can be spent toward searching for collision-free paths in $\mathcal{C}$ for problems induced by these arrangements. For each unresolved arrangement set $c \in C$ that remains after filtration (lines 10–13), we initiate a composite motion-planning problem where the movable obstacles assume all arrangements in c simultaneously ($O_m(c)$). Formally, this problem is denoted as $P = \{p(m) \mid m \in c\}$, where $p(m)$ is a semi-static motion planning problem when O_m assumes an arrangement m (Equation 1). For each successful attempt, $\mathcal{G}$ is augmented with additional vertices and edges that help to resolve these composite problems and improve its problem space coverage. This is illustrated in Fig. 3.

B. Building the Roadmap

COVER computes a coverage-informed roadmap through an iterative procedure to maximize its problem-space coverage. The pipeline for computing $\mathcal{G}$ is summarized in Alg. 3.

We begin by initializing the roadmap $\mathcal{G}$ with the start and goal configurations as vertices in line 1. For each movable obstacle $o_i \in O_m$, the corresponding decomposition tree T_i is constructed using Alg. 1. This subroutine partitions the configuration space M_i into multiple regions according to the
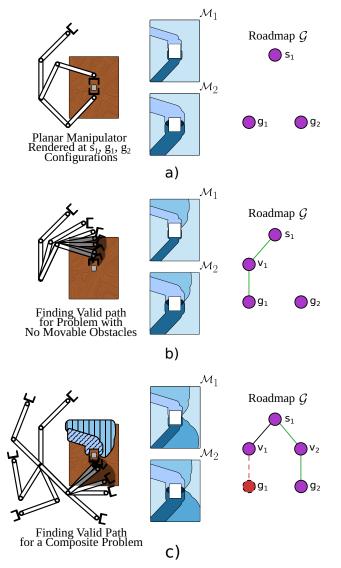
Fig. 3: Demonstration of COVER with two movable obstacles o_1 and o_2. **(a)** The roadmap $\mathcal{G}$ is initialized with start and goal configurations. $\mathcal{M}_1$ and $\mathcal{M}_2$ are partitioned into subregions (Alg. 1), resulting in distinct subdivisions due to different obstacle sizes. **(b)** A path (green edge) is added under the assumption of no movable obstacles, after which the decomposition trees of $\mathcal{M}_1$ and $\mathcal{M}_2$ are refined with the new edges. **(c)** A composite motion planning problem is formed by o_1 assuming configurations that invalidate the new edge and o_2 assuming configurations that block goal configuration g_2. A feasible path is then added to $\mathcal{G}$. The process repeats until $\mathcal{M}_{\text{uncov}}$ is exhausted or a set number of consecutive failures occurs.

Algorithm 3: Computing COVER

Input: Start configurations V_s, Goal configurations V_g, Static obstacles O_f, Movable obstacles O_m

Output: Roadmap $\mathcal{G} = (V, E)$, Decomposition trees T, Unverified obstacle arrangements C

1 $\mathcal{G} \leftarrow \texttt{InitializeRoadmap}(V_s, V_g)$
    // Generate Decomposition trees for obstacle configuration spaces
2 $T \leftarrow \texttt{PartitionObstacleSpace}(G, O_m)$
    // Motion-planning problem with no movable obstacles
3 $\pi \leftarrow \texttt{RepairRoadmap}(G, O_m(\emptyset), O_f)$
4 $\texttt{AddPath}(G, \pi)$
    // Get all arrangements in $\mathcal{M}_{\text{uncov}}$
5 $C \leftarrow \texttt{DetectUncoveredPartitions}(G, O_m)$
6 $failures \leftarrow 0$
7 **while** $C \neq \emptyset$ **do**
8     $c \leftarrow \texttt{Dequeue}(C)$
        // Remove invalid edges due to obstacle arrangements in c
9     $\mathcal{G}' \leftarrow \texttt{RemoveInvalidEdges}(\mathcal{G}, c)$
        // Initialize motion-planning problem with obstacles assuming all arrangements in c
10     $\pi \leftarrow \texttt{RepairRoadmap}(\mathcal{G}', O_m(c), O_f)$
        // Path found if planning is successful
11   **if** $\pi \neq \emptyset$ **then**
12         $\texttt{AddPath}(G, \pi)$
        // Update the partitions for new edges
13         $T \leftarrow \texttt{PartitionObstacleSpace}(\mathcal{G}, O_m)$
        // Remove arrangements covered by $\mathcal{G}$ after augmenting
14         $C \leftarrow$
        $\texttt{DetectUncoveredPartitions}(\mathcal{G}, O_m)$
15         $failures \leftarrow 0$
16   **else**
        // Motion planning failed for $O_m(c)$
17   add c back to C
18         $failures \leftarrow failures + 1$
19   **if** $failures == |C|$ **then**
            // Cannot resolve any arrangements
20   **return** $\mathcal{G}, T, C$

    // Paths found for all obstacle arrangements
21 **return** $\mathcal{G}, T, \emptyset$

---

invalidation of roadmap edges, start, and goal configurations (Fig. 3 (a)). At this stage, the arrangements that render all start or goal configurations invalid could be obtained. These arrangements correspond to infeasible problems and can be excluded to prevent expending computational effort on unsolvable cases.

As an initialization step, a path is computed in the environment with no movable obstacles ($O_m(\emptyset)$) and added to $\mathcal{G}$ in lines 5—6. The decomposition trees are updated to incorporate the new edges, leading to further subdivisions in M_1 and M_2 as shown in Fig. 3 (b). Finally, Alg. 2 is used to extract the set of arrangements C that invalidate all the paths currently in $\mathcal{G}$.

After the unverified arrangement set C is obtained, the algorithm enters an iterative repair loop. In each iteration, we dequeue an arrangement set $c \in C$, constructed by choosing one leaf node from each obstacle's decomposition tree. The edges that are invalidated by the arrangements in c can be obtained through its composite signature. Then in line 11, we derive a restricted roadmap $\mathcal{G}'$ by removing all edges of $\mathcal{G}$ that are invalidated by the placements in c. Since the arrangements in c disconnect the start vertices from the goal vertices, we attempt to reconnect the disconnected components in line 12. A motion planner is invoked for the composite problem in

which the movable obstacles assume all the arrangements in c $(O_m(c))$. If a feasible path π is found for this problem, it is added to $\mathcal{G}$, whose addition ensures that $\mathcal{G}$ contains a valid path for any arrangement in c. To check if π resolves arrangements $\notin c$, Alg. 2 is invoked again to filter out these arrangements that is covered by the updated $\mathcal{G}$.

Conversely, if no path can be found for the composite problem during repair, the arrangement set c remains unresolved. This is recorded as a failure before proceeding to the other sets in C for subsequent iterations. This process repeats until either all obstacle arrangements are covered, or it terminates after a fixed number of consecutive failures, indicating that the remaining set C. This termination condition prevents COVER from running indefinitely.

C. Querying the roadmap

After computing the coverage-informed roadmap $\mathcal{G}$ and the decomposition trees for the obstacle configuration spaces, collision-free paths can be retrieved from the roadmap for a given arrangement of obstacles $m \in \mathcal{M}_{\mathrm{cov}}$. Since edge invalidation is already encoded during the offline computation of the decomposition trees (Alg. 1), the collision-checking in the online phase reduces to inexpensive binary space partitioning tree traversals. For each movable obstacle $o_i \in O_{\mathrm{m}}$, the respective decomposition tree T_i is traversed to locate the partition containing the current configuration. The binary signature $\mathbf{b}_i$ associated with this partition directly specifies the set of edges invalidated by o_i. Then, the signatures from the partition of each tree are then merged into the composite signature $\mathbf{b}$, which encodes the complete set of edges blocked by the obstacle arrangement m.

Using the composite signature $\mathbf{b}$ together with the path–edge incidence matrix $\mathbf{P}$ computed during roadmap construction, we use Equation 3 to obtain $\mathbf{v}$ that identifies which precomputed paths are invalidated by the arrangement m. Any path $\pi_j \in \Pi$ satisfying $\mathbf{p}_j \wedge \mathbf{b} = 0$ remains collision-free and can be returned directly as the solution to the query.

V. THEORETICAL GUARANTEES

A. Fixed-Time Guarantees

Querying the roadmap can be performed within a fixed-time budget t_{query}, offering the same guarantee as in [6]. As outlined earlier, two operations are required to recover a collision-free path for a new obstacle arrangement m.

First, we identify the composite binary signature associated with m, which requires traversing the decomposition trees. Each traversal has complexity $O(\log |E|)$, where $|E|$ is the number of edges in the roadmap, and must be performed n times for n movable obstacles. Second, the composite signature is tested against the path set Π via a matrix–vector multiplication between a matrix of size $|P| \times |E|$ (the path–edge incidence matrix) and a binary vector of size $[E] \times 1$ (the signature). This operation has complexity $O(|P| \cdot |E|)$. Thus, the overall complexity of answering a query is given by

$$O(n \log |E| + |P| \cdot |E|)$$

Since the roadmap is precomputed and remains fixed across all obstacle arrangements, the query time can be deterministically upper-bounded by a constant t_{query}.

B. Completeness Guarantees

Unlike [6], which guarantees that any problem solvable by the underlying planner will yield a corresponding path in Π, the current implementation of COVER cannot provide such a strong completeness guarantee, since roadmap construction terminates once all generated composite problems cannot be resolved.

Therefore, the only completeness guarantee that COVER can provide is the following: if an arrangement $m \in \mathcal{M}_{\mathrm{cov}}$, then a valid collision-free path is guaranteed to exist in the roadmap. For arrangements outside $\mathcal{M}_{\mathrm{cov}}$, no paths exist on the roadmap.

VI. EXPERIMENTS

In this section, we present the evaluation of COVER in simulation on object-picking tasks using a Franka Emika Panda manipulator. The problem space coverage achieved by the generated roadmaps for these environments is compared against the Alternative Paths Planner (APP). [6].

A. Experimental Setup

All experiments were conducted in Genesis [21] simulation on a workstation with 4 NVIDIA A100 GPUs, 56-core Intel Xeon CPU, and 128 GB RAM. For motion planning and roadmap-based operations, we leverage the sampling-based motion planning framework of Open Motion Planning Library (OMPL) [22]. Furthermore, we obtain the spherical approximation of the robot links using [23] and compute the corresponding swept-spheres convex hulls as an approximation of the volume swept by the robot trajectory. Since the robot occupancy and obstacle configuration spaces are modeled as polygonal meshes, we employ the methods in MeshLib [24] for computing their mesh representation and for the 3D Boolean operations for subdividing of obstacle configuration spaces.

The following environments were used as the primary testbeds for evaluating performance on object-picking tasks. Fig. 4 shows the three environments of increasing complexity. In Table-Pick (Fig. 4 (a)), the robot is placed on a rectangular table with the cylindrical target object also on the table surface, and the movable obstacles can be located anywhere on that surface. In Shelf-High (Fig. 4 (b)), the target object is placed on an upper rack of the shelf, with two movable obstacles positioned within the same rack. In Shelf-Low (Fig. 4 (c)), the target and obstacles are placed on a lower rack, which introduces narrow passages that make planning more challenging.

Each environment contains two spherical movable obstacles: one fixed at a radius of 0.1 m, and another that varies across $0.025, 0.05, 0.075, 0.1$ m. This yields four obstacle-size pairs per environment — (0.025, 0.1), (0.05, 0.1), (0.075, 0.1), and (0.1, 0.1). We also ensure that, for every movable obstacle, configurations that collide with the target object are removed from its configuration spaces, so that the picking task always admits at least one valid grasp.
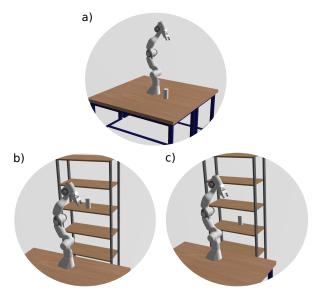
Fig. 4: Semi-static environments with 2 movable obstacles, used for evaluating the object-pick tasks. **(a)** `Table-Pick` with target object and movable obstacles on a table surface. **(b)** `Shelf-High` with target and obstacles on the upper rack. **(c)** `Shelf-Low` with target and obstacles on the lower rack.

For all the setups, we consider four pre-grasp poses for the panda end-effector—{`pick-left`, `pick-right`, `pick-front`, `pick-behind`}. We do not include top grasps, as they would trivialize the table scenario. We attempt to find valid inverse kinematic solutions for each pre-grasp pose. For `Table-Pick`, three grasps are valid, while in `Shelf-High` and `Shelf-Low`, only two of the four pre-grasps have a valid IK-solution.

B. Baseline Comparison

We use APP as our main baseline. APP constructs $n+1$ disjoint paths for n movable obstacles during preprocessing. If not enough disjoint paths can be found, APP handles the remaining composite problems by splitting them into smaller subproblems and generating overlapping paths, ensuring that at least one path remains valid for any arrangement of the movable obstacles. Since the original implementation targets discrete placements and focuses on fast retrieval using hash tables, we adapt APP to continuous spaces by redefining how its envelopes are computed: rather than using occupancy grids, we follow the same process as in Alg. 1, computing the Minkowski sum of each path with the obstacles to obtain the set of configurations that invalidate it. This adaptation is done solely to compare preprocessing performance of APP in generating paths against that of COVER.

We set a preprocessing timeout of 10 minutes for both APP and COVER, which is sufficient for planning in these environments with two movable obstacles and ensures that preprocessing does not continue indefinitely. To enable a direct comparison, we precompute and store disjoint paths [6] for each run and use them to initialize both APP and COVER. Using the same set of disjoint paths to warm-start both methods ensures that they begin from an identical base of solutions.

C. Results and Discussion

We present the problem-space coverage of the resultant roadmaps generated using COVER and the set of paths with APP. Each experiment is run for five trials per obstacle-size pair to ensure consistency, with the median value reported across trials.

A fundamental assumption of APP is that all movable obstacles are the same size. Thus even in the cases that the obstacles are different size, APP needs to assume that they both have the size of the bigger obstacle [6]. In contrast, COVER builds a decomposition tree per movable obstacle, allowing us to capture the contributions of heterogeneous obstacles individually.

For the paths returned by APP, we use the proposed coverage estimation method Alg. 2 to quantify its problem space coverage. The resulting median coverage values are shown in Fig. 5 (a) for each experiment and obstacle-size pair. Across all environments, the green bar marks the upper bound on achievable coverage, obtained by filtering out problem instances where no valid start or goal configuration exists. To complement the results in Fig. 5 (a), we also report the coverage achieved by both APP and our method relative to the initial coverage provided by the disjoint paths used to warm-start them in Fig. 5 (b).

For the object-picking tasks in the `Table-Pick`, both COVER and APP record near-perfect coverage for the $(0.1, 0.1)$ obstacle pair. For all smaller obstacle-size pairs, maximum coverage is consistently achieved across all trials by COVER. This outcome is supported by the availability of more valid grasp solutions and the largely-free space in the table environment.

In the `Shelf-High` setting, increasing the size of the second obstacle reduces both optimal and achieved coverage, as the narrow shelf sections make even single arrangements challenging, let alone composite ones. Despite this, our roadmaps achieve markedly higher coverage than the overlapping paths generated by APP, as evident in Fig. 5. For `Shelf-Low`, APP shows slightly higher coverage for $(0.1, 0.1)$ in a small subset of trials due to its recursive splitting of unresolved composites, but in most cases (see Fig. 5 (b)) it fails to improve beyond the initial disjoint paths. By contrast, COVER augments its roadmap in for the $(0.025, 0.1)$ movable obstacle pair, yielding noticeable coverage gains.

VII. DISCUSSION

We have presented COVER-a roadmap building method that reasons about problem-space coverage in semi-static environments. By associating roadmap edges with obstacle-space partitions, COVER can explicitly certify subsets of the arrangement space that are feasible, moving beyond heuristic or discretization-based approaches. This treatment of coverage allows the framework to handle heterogeneous obstacle sizes and placements while still providing fixed-time guarantees at query stage.

At the same time, several limitations remain. The preprocessing cost grows with the number of obstacles and
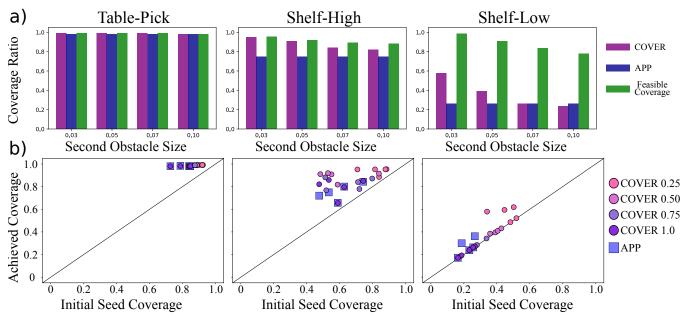
Fig. 5: Problem-space coverage across environments. **(a)** Median coverage of APP and COVER over five trials for each obstacle-size pair. Bars show median values, with green bars denoting the maximum feasible coverage after filtering problems with all starts or all goals invalid. **(b)** Coverage relative to the initial disjoint paths used to warm-start both methods. Dots indicate individual trial results for COVER and Squares correspond to trial results of APP.

roadmap edges, which may hinder scalability to larger environments. Moreover, the current implementation does not recursively split unresolved composite problems, and thus cannot guarantee that uncovered partitions are truly infeasible except when all starts or all goals are invalid. Incorporating infeasibility-detection techniques and principled recursive refinement strategies are promising directions. Beyond the settings evaluated here, we see potential for COVER to serve as a diagnostic tool for roadmap coverage and as a foundation for Task and Motion Planning frameworks, where verified low-level coverage can reduce high-level planning effort.

REFERENCES

[1] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, and W. Burgard, *Principles of robot motion: theory, algorithms, and implementations*. MIT press, 2005.

[2] D. Coleman, I. A. Sucan, M. Moll, K. Okada, and N. Correll, "Experience-Based Planning with Sparse Roadmap Spanners," Oct. 2014, arXiv:1410.1950 [cs].

[3] D. Berenson, P. Abbeel, and K. Goldberg, "A robot path planning framework that learns from experience," in *2012 IEEE International Conference on Robotics and Automation*. St Paul, MN, USA: IEEE, May 2012, pp. 3671–3678.

[4] C. Chamzas, A. Cullen, A. Shrivastava, and L. E. Kavraki, "Learning to Retrieve Relevant Experiences for Motion Planning," in *2022 International Conference on Robotics and Automation (ICRA)*. Philadelphia, PA, USA: IEEE, May 2022, pp. 7233–7240.

[5] H. Ishida, N. Hiraoka, K. Okada, and M. Inaba, "CoverLib: Classifiers-Equipped Experience Library by Iterative Problem Distribution Coverage Maximization for Domain-Tuned Motion Planning," *IEEE Transactions on Robotics*, vol. 41, pp. 2911–2930, 2025.

[6] F. Islam, C. Paxton, C. Eppner, B. Peele, M. Likhachev, and D. Fox, "Alternative Paths Planner (APP) for Provably Fixed-time Manipulation Planning in Semi-structured Environments," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. Xi'an, China: IEEE, May 2021, pp. 6534–6540.

[7] J. Van Den Berg, D. Nieuwenhuisen, L. Jaillet, and M. Overmars, "Creating robust roadmaps for motion planning in changing environments," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Edmonton, Alta., Canada: IEEE, 2005, pp. 1053–1059.

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

[9] R. Bohlin and L. Kavraki, "Path planning using lazy PRM," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*. San Francisco, CA, USA: IEEE, 2000, pp. 521–528 vol.1.

[10] T. Marcucci, M. Petersen, D. Von Wrangel, and R. Tedrake, "Motion planning around obstacles with convex optimization," *Science Robotics*, vol. 8, no. 84, p. eadf7843, Nov. 2023.

[11] M. Kallman and M. Mataric, "Motion planning using dynamic roadmaps," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*. New Orleans, LA, USA: IEEE, 2004, pp. 4399–4404 Vol.5.

[12] P. Leven and S. Hutchinson, "A Framework for Real-time Path Planning in Changing Environments," *The International Journal of Robotics Research*, vol. 21, no. 12, pp. 999–1030, Dec. 2002.

[13] J. Van Den Berg and M. Overmars, "Roadmap-based motion planning in dynamic environments," *IEEE Transactions on Robotics*, vol. 21, no. 5, pp. 885–897, Oct. 2005.

[14] Y. Yang, W. Merkt, H. Ivan, Z. Li, and V. Vijayakumar, "HDRM: A Resolution Complete Dynamic Roadmap for Real-Time Motion Planning in Complex Scenes," *IEEE Robotics and Automation Letters*, vol. 3, no. 1, pp. 551–558, Jan. 2018.

[15] M. Phillips, B. Cohen, S. Chitta, and M. Likhachev, "E-Graphs: Bootstrapping Planning with Experience Graphs," *Proceedings of the International Symposium on Combinatorial Search*, vol. 3, no. 1, pp. 188–189, Aug. 2021.

[16] C. Chamzas, Z. Kingston, C. Quintero-Pena, A. Shrivastava, and L. E. Kavraki, "Learning Sampling Distributions Using Local 3D Workspace Decompositions for Motion Planning in High Dimensions," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. Xi'an, China: IEEE, May 2021, pp. 1283–1289.

[17] E. Pairet, C. Chamzas, Y. R. Petillot, and L. Kavraki, "Path Planning for Manipulation Using Experience-Driven Random Trees," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3295–3302, Apr. 2021.

[18] A. Orthey, C. Chamzas, and L. E. Kavraki, "Sampling-Based Motion Planning: A Comparative Review," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 7, no. 1, pp. 285–310, Jul. 2024.

[19] A. Dobson and K. E. Bekris, "Sparse roadmap spanners for asymptotically near-optimal motion planning," *The International Journal of Robotics Research*, vol. 33, no. 1, pp. 18–47, Jan. 2014.

[20] S. Li and N. T. Dantam, "A sampling and learning framework to prove motion planning infeasibility," *The International Journal of Robotics Research*, vol. 42, no. 10, pp. 938–956, Sep. 2023.

[21] G. Authors, "Genesis: A Universal and Generative Physics Engine for Robotics and Beyond," Dec. 2024.

[22] I. A. Sucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning

Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, Dec. 2012.

[23] S. Coumar, G. Chang, N. Kodkani, and Z. Kingston, "Foam: A Tool for Spherical Approximation of Robot Geometry," Mar. 2025, arXiv:2503.13704 [cs].

[24] "MeshLib - 3D mesh processing library C++, Python, C#, C," Feb. 2025.