**Toward Asymptotically-Optimal Inspection Planning via Efficient Near-Optimal Graph Search**

Mengyu Fu∗, Alan Kuntz ∗, Oren Salzman†, and Ron Alterovitz∗

∗Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, NC 27599, USA
Email: {mfu,adkuntz,ron}@cs.unc.edu

†Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA
Email: osalzman@andrew.cmu.edu

**Abstract**—Inspection planning, the task of planning motions that allow a robot to inspect a set of points of interest, has applications in domains such as industrial, field, and medical robotics. Inspection planning can be computationally challenging, as the search space over motion plans grows exponentially with the number of points of interest to inspect. We propose a novel method, Incremental Random Inspection-roadmap Search (IRIS), that computes inspection plans whose length and set of successfully inspected points asymptotically converge to those of an optimal inspection plan. IRIS incrementally densifies a motion planning roadmap using sampling-based algorithms, and performs efficient near-optimal graph search over the resulting roadmap as it is generated. We demonstrate IRIS’s efficacy on a simulated planar 5DOF manipulator inspection task and on a medical endoscopic inspection task for a continuum parallel surgical robot in cluttered anatomy segmented from patient CT data. We show that IRIS computes higher-quality inspection plans orders of magnitudes faster than a prior state-of-the-art method.

**I. INTRODUCTION**

We consider the problem of inspection planning, or coverage planning [1, 19]. Here, we consider the specific setting where we are given a robot equipped with a sensor and a set of points of interest (POI) in the environment to be inspected by the sensor. The problem calls for computing a minimal-length motion plan for the robot that maximizes the number of POI inspected. This problem has a multitude of diverse applications, including industrial surface inspections in production lines [43], surveying the ocean floor by autonomous underwater vehicles [3, 20, 26, 52], structural inspections of bridges using aerial robots [4, 5], and medical applications such as inspecting patient anatomy during surgical procedures [32], which motivates this work.

Naïvely-computed inspection plans may enable inspection of only a subset of the POI and may require motion plans orders of magnitude longer than an optimal plan, and hence may be undesirable or infeasible due to battery or time constraints. In medical applications, physicians may want to maximize the number of POI inspected for diagnostic purposes. Additionally, the procedure should be completed as fast as is safely possible to reduce costs and improve patient outcomes, especially if the patient is under anesthesia during the procedure. For example, a robot assisting in the diagnosis of the cause of a pleural effusion (a serious medical condition which causes the collapse of a patient’s lung) will need to visually inspect the surface of the collapsed lung and chest wall inside the body in as short a time as possible (see Fig. 1).

We note that it may not be possible to inspect some POI due to obstacles in the anatomy and the kinematic constraints of the robot. Our goal is to compute high-quality inspection plans that maximize the number of POI inspected, and of the motion plans that inspect those POI we compute a shortest plan that avoids obstacles.

Inspection planning is computationally challenging because the search space is embedded in a high-dimensional config-
uration space $\mathcal{X}$ (space of all parameters that determine the shape of the robot) [10, 33, 34]. Even finding the shortest plan between two points in $\mathcal{X}$ that avoid obstacles (without reasoning about inspection) is computationally hard. If we want a minimum-length motion plan that maximizes the number of POI inspected, our problem is accentuated as we have to simultaneously reason about the system’s constraints, motion plan length, and POI inspected.

There are multiple approaches to computing inspection plans. Optimization-based methods locally search over the space of all inspection plans [4, 7]. Decoupled approaches first independently select suitable viewpoints and then determine a visiting sequence, i.e., a motion plan for the robot that realizes this sequence [11, 17]. Finally, recent progress in motion planning [28] has enabled methods to exhaustively search over the space of all motion plans [6, 27, 41] thus guaranteeing asymptotic optimality, a key requirement in many applications, including medical ones. Roughly speaking, asymptotic optimality for inspection planning means these methods produce inspection plans whose length and number of points inspected will asymptotically converge to those of an optimal inspection plan, given enough planning time.

Of all the aforementioned methods, only algorithms in the latter group provide any formal guarantees on the quality of the solution. This guarantee is achieved by attempting to exhaustively compute the set of Pareto-optimal inspection plans embedded in $\mathcal{X}$. In our setting, the set of Pareto-optimal inspection plans is the minimal set of inspection plans such that each plan is either shorter or has better coverage of the POI than any other inspection plan. Unfortunately, this comes at the price of very long computation times as the search space is exponential in the number of POI.

To this end, we introduce Incremental Random Inspection-roadmap Search (IRIS), a new asymptotically-optimal inspection-planning algorithm. IRIS incrementally constructs a sequence of increasingly dense roadmaps—graphs embedded in $\mathcal{X}$ wherein each vertex represents a collision-free configuration and each edge a collision-free transition between configurations—and computes an inspection plan on the roadmaps as they are constructed (see Fig. 2). Unfortunately, even the problem of computing an optimal inspection plan on a graph (and not in the continuous space) is computationally hard. To this end, our key insight is to compute a near-optimal inspection plan on each roadmap. Namely, we compute an inspection plan that is at most $1 + \varepsilon$ the length of an optimal plan while covering at least $p$-percent of the number of POI (for any $\varepsilon \geq 0$ and $p \in (0, 1]$). This additional flexibility allows us to improve the quality of our inspection plan in an anytime manner, i.e., the algorithm can be stopped at any time and return the best inspection plan found up until that point.

We achieve this by incrementally densifying the roadmap and then searching over the densified roadmap for a near-optimal inspection plan—a process that is repeated as time allows. By reducing the approximation factor between iterations, we ensure that our method is asymptotically optimal.

The key contribution of our work is a computationally-efficient algorithm to compute provably near-optimal inspection plans on graphs. By pruning away large portions of the search space, this algorithmic building block enables us to dramatically outperform Rapidly-exploring Random Tree Of Trees (RRTOT) [6]—a state-of-the-art asymptotically-optimal inspection planner. Specifically, we demonstrate the efficacy of our approach in simulation for a continuum robot with complex dynamics—the needle-diameter Continuum Incisionless Surgical Parallel (CRISP) robot [2, 38], working in a medically-inspired setting involving diagnosis of a pleural effusion (see Fig. 1).

II. RELATED WORK

A. Sampling-based motion planning

Motion planning aims to compute a collision-free motion for a robot to accomplish a task in an environment cluttered with obstacles [22, 34, 37]. A common approach to motion planning is by sampling-based algorithms that construct a roadmap. Examples include the Probabilistic Roadmaps (PRMs) [29] (for solving multiple queries) and the Rapidly-exploring Random Trees (RRTs) [35] for solving single queries. These methods, and many variations thereof, are probabilistically complete—namely the likelihood that they will find a solution, if one exists, approaches certainty as computation time increases.

Recent variations of these methods, such as PRM* and RRT* [28], improve upon this guarantee by exhibiting asymptotic optimality—namely that the quality of the solution obtained, given some cost function, approaches the global optimum as computation increases. Roughly speaking, this is achieved by increasing the (potential) edge set of roadmap vertices considered as its size increases [28, 50]. One such algorithm is the Rapidly-exploring Random Graphs (RRGs) [28] which will be used in our work. RRG combines the exploration strategy of RRT with an updated connection strategy that allows for cycles in the roadmap. It requires solving the two-point boundary value problem [34], which is only available for some robotic systems (including ours).
Guaranteeing asymptotic optimally can come with heavy computational cost. This inspired work on planners that trade asymptotic optimality guarantees with asymptotic near optimality (e.g., [36, 39, 47]). Asymptotic near optimality states that given an approximation factor $\varepsilon \geq 0$, the solution obtained converges to within a factor of $(1 + \varepsilon)$ of the optimal solution with probability one, as the number of samples tends to infinity. Relaxing optimality to near optimality allows a method to improve the practical convergence rate while maintaining desired theoretic guarantees on the quality of the solution.

### B. Inspection planning

Many inspection-planning algorithms, or coverage planners, decompose the region containing the POI into multiple sub-regions, and then solve each sub-region separately [19]. These methods have limitations, however, such as when occlusions play a significant role in the inspection [16], or when kinematic constraints must be considered [14].

Other approaches simultaneously consider all POI. One approach decouples the problem into the coverage sampling problem (CSP) and the multi-goal planning problem (MPP), and solves each independently [4, 11, 14, 16, 17]. In CSP, a minimal set of viewpoints that provide full inspection coverage is computed. In MPP, a shortest tour that connects all the viewpoints is computed. These correspond to solving the art gallery problem and the traveling salesman problem, respectively. Several of these variants have been shown to be probabilistically complete [16], but none provide guarantees on the quality of the final solution.

The set of viewpoints and the inspection plan itself can also be generated simultaneously. Papadopoulos et al. propose the Random Inspection Tree Algorithm (RITA) [41]. RITA takes into account differential constraints of the robot and computes both target points for inspection and the trajectory to visit the targets simultaneously. Bircher et al. propose Rapidly-exploring Random Tree Of Trees (RRTOT) which constructs a metatree structure consisting of multiple RRT$^*$ trees [6]. Both methods, which were shown to be asymptotically optimal, iteratively generate a tree, in which the inspection plan is enforced to be a plan from the root to a leaf node. However, each inspection plan does not consider configurations from other branches in the tree which may cause long planning times. This motivates our RRtG-based approach.

### C. Path planning on graphs

Planning a minimal-cost path on a graph is a well studied problem. When the cost function has an optimal substructure (namely, when subpaths of an optimal path are also optimal), efficient algorithms such as Dijkstra [13], A$^*$ [23] and the many variants there-of can be used. However, in certain settings, including ours, this is not the case. For example Tsaggouris and Zaroliagis [53] consider the case where every edge has two attributes (e.g., cost and resource), and the cost function incorporates the attributes in a non-linear fashion.

Inspection planning also bears resemblance to multi-objective path planning. Here, we are given a set of cost functions and are required to find the set of Pareto-optimal paths [42]. Unfortunately, this set may be exponential in the problem size [15]. However it is possible to compute a fully polynomial-time approximation scheme (FPTAS) for many cases [54]. For additional results on path-planning with multiple-objectives or when the cost function does not have an optimal substructure, see e.g., [9, 45] and references within.

### III. Problem Definition

In this section we formally define the inspection planning problem. The robot operates in a workspace $W \subseteq \mathbb{R}^3$ amidst a set of obstacles $W_{\text{obs}} \subseteq W$. The robot’s configuration $q$ is a $d$-dimensional vector that uniquely defines the shape of the robot (including, for example, rotation angles and translational extension of all joints). The set of all such configurations is the configuration space $\mathcal{X}$. The geometry of the robot is a configuration-dependent shape $\text{Shape}(q) \subseteq \mathcal{W}$ and we say that $q \in \mathcal{X}$ is collision free if $\text{Shape}(q) \cap W_{\text{obs}} = \emptyset$. In this work we define a motion plan for the robot as a path $P$ in $\mathcal{X}$, which is represented as a sequence of $n$ configurations $\{q_0, \ldots, q_{n-1}\}$ (vertices) connected by straight-line segments (edges) in $\mathcal{X}$. And we say that $P$ is collision free if all configurations along $P$ (vertices and edges) are collision free. We assume that we have a distance function $\ell : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ and denote the length of a path $P$ as the sum of the distances between consecutive vertices, i.e., $\ell(P) := \sum_i \ell(q_i, q_{i+1})$.

We assume that the robot is equipped with a sensor and we are given a set of $k$ points of interest (POI) $I = \{i_1, \ldots, i_k\}$ in $W$. We model the sensor as a mapping $S : \mathcal{X} \rightarrow 2^I$, where $2^I$ is the power set of $I$ and $S$ denotes the subset of $I$ that can be inspected from each configuration. By a slight abuse of notation, given a path $P$ we set $S(P) := \bigcup_{i=0}^{n-1} S(q_i)$ and note that in our model, we only inspect $I$ along the vertices of a path.

**Definition 1:** A point of interest $i \in I$ is said to be covered by a configuration $q \in \mathcal{X}$ or by a path $P$ if $i \in S(q)$ or if $i \in S(P)$, respectively. In such a setting, we say that $q$ (or $P$) covers the point of interest $i$.

Given a start configuration $q_s \in \mathcal{X}$, POI $I$, and a sensor model $S$, the inspection planning problem calls for computing a collision-free path $P$ starting at $q_s$ which maximizes $|S(P)|$ while minimizing $\ell(P)$. Note that this is *not* a bicriteria optimization problem—our primary optimization function is maximizing the coverage of our path. Out of all such paths we are interested in the shortest one.

### IV. Method Overview

In this section we provide an overview of IRIS—our algorithmic framework for computing asymptotically-optimal inspection plans. A key algorithmic tool in our approach is to cast the continuous inspection planning problem (Sec. III) to a discrete version of the problem where we only consider a finite number of configurations from which we inspect the POI, and a discrete set of feasible movements between those configurations. Thus, we start in Sec. IV-A by formally
defining the graph inspection problem and then continue in Sec. IV-B to provide an overview of how IRIS builds and uses such graphs. We then describe the method in detail in Sec. V, and in Sec. VI show that IRIS’s solution converges to the length and coverage of an optimal inspection path.

A. Graph inspection problem

Similar to the (continuous) inspection problem, a graph inspection problem is a tuple \((G, T, S, \ell, v_s)\) where \(G = (V, E)\) is a motion-planning roadmap (namely, a graph embedded in \(\mathcal{X}\), in which every vertex \(v \in V\) is a configuration and every edge \((u, v) \in E\) denotes the transition from configuration \(u\) to \(v\)), \(T\) and \(S\) are defined as in Sec. III, \(\ell : E \rightarrow \mathbb{R}\) denotes the length of each edge in the roadmap, and \(v_s\) is the start vertex (corresponding to the start configuration \(q_s\)). A path \(P\) on \(G\) is represented by a sequence of vertices \(v_i \in V\) such that \(P = \{v_0, \ldots, v_n\}, v_0 = v_s\) and \((v_i, v_{i+1}) \in E\). It is important to note that there can be loops in a path, so it is possible that \(v_m = v_k\) for \(m \neq k\). The length and coverage of \(P\) are defined as the total length of \(P\)’s edges and the set of all points inspected when traversing \(P\), respectively. Namely, \(\ell(P) := \sum_{i=0}^{n-2} \ell(v_i, v_{i+1})\) and \(S(P) := \bigcup_{v \in P} S(v)\). The optimal graph inspection problem calls for a path \(P^*\) that starts at \(v_s\) and maximizes the number of points inspected. Out of all such paths, \(P^*\) has the minimal length. Finally, a path is said to be near-optimal for some \(\varepsilon \geq 0\) and \(p \in (0, 1]\) if \(|S(P)|/|S(P^*)| \geq p\) and \(\ell(P) \leq (1 + \varepsilon) \cdot \ell(P^*)\).

B. Overview of IRIS

Our algorithmic framework, depicted in Fig. 2, interleaves sampling-based motion planning and graph search. Specifically, we incrementally construct an RRT \(T\) rooted at \(q_s\) which implicitly defines a corresponding RRG \(G\). All edges in \(T\) are checked for collision with the environment during its construction (so the roadmap is guaranteed to be connected) while all the other edges of \(G\) are not explicitly checked for collision. Lazy edge evaluation, common in motion-planning algorithms [8, 24, 12, 46], allows us to defer collision detection until absolutely necessary and reduce computational effort. This is critical in our domain of interest where computing Shape() typically dominates algorithms’ running times [40]

The roadmap \(G = (V, E)\) induces the subset of the POI that can be inspected, denoted as \(I_G := \bigcup_{v \in V} S(v)\). Given two approximation parameters \(\varepsilon \geq 0\) and \(p \in (0, 1]\), we compute a near-optimal inspection path for the graph-inspection problem \((G, I_G, S, \ell, v_s)\) by casting the problem as a graph-search problem on a different graph \(G_S\) (to be defined shortly).

As we add vertices and edges to \(T\) incrementally, the roadmap \(G\) is incrementally densified. In addition, we tighten approximations by decreasing \(\varepsilon\) and increasing \(p\) between iterations. As we will see (Sec. VI), this will ensure that IRIS is asymptotically optimal.

V. METHOD

In this section we detail the different components of IRIS. Sec. V-A and V-B describe how we construct a roadmap and then search it, respectively. After describing in Sec V-C how we modify the approximation parameters used by IRIS, we conclude in Sec. V-D with implementation details.

A. Roadmap construction

We construct a sequence of graphs embedded in \(\mathcal{X}\). Specifically, denote the RRT constructed at the \(i\)th iteration as \(T_i\) defined over the set of vertices \(V_i\). We start with an empty tree rooted at \(q_s\) and at the \(i\)th iteration sample a random configuration, compute it’s nearest neighbor in \(T_i\), and extend that vertex toward the random configuration. If that extension is collision free we add it to the tree. If not, we repeat this process (see [31, 34] for additional details regarding RRT).

The tree \(T_i\) implicitly-defines an RRG \(G_i = (V_i, E_i)\) defined over the same set of vertices where every vertex is connected to all other vertices within distance \(r_i\). Here, we define \(r_i\) as in [49, Thm. IV.5] which will allow us to prove that our approach is asymptotically optimal (see Sec. VI).

B. Graph inspection planning

We use the RRG described in Sec. V-A to define a graph inspection problem, and then compute near-optimal inspection paths over this graph. Before describing how we compute near-optimal inspection paths, we first describe how we compute optimal paths given a graph inspection problem.

1) Optimal planning: Given a graph inspection problem \((G, I_G, S, \ell, v_s)\), we compute optimal inspection paths by formulating our inspection problem as a graph-search problem on an inspection graph \(G_S := (V_S, E_S)\). Here, vertices are pairs comprised of a vertex \(u \in V\) in the original graph and subsets of \(I_G\). Namely, \(V_S = V \times 2^{I_G}\), and note that \(|V_S| = O(|V| \cdot 2^{|I_G|})\). An edge \(e\) between vertices \((u, I_u)\) and \((v, I_v)\) exists if \((u, v) \in E\) and \(I_u \cup S(v) = I_v\). If it exists, its cost is simply \(\ell(u, v)\).

A path in the inspection graph \(G_S\) from the vertex \((v_s, S(v_s)) \in V_S\) to some vertex \((u, I_u) \in V_S\) corresponds to a (possibly non-simple) path in the original graph \(G\) of the same length and the same coverage. Thus, our algorithm will run an \(A^*\)-like search from \((v_s, S(v_s)) \in V_S\) to any vertex in the goal set \(V_{goal} = \{(v, I_G)v \in V\}\). An optimal inspection path is the shortest path between \((v_s, S(v_s))\) and any vertex in \(V_{goal}\). For a visualization, see Fig. 3. Note that here we assume the graph \(G\) is connected and that the set of points to be inspected is \(I_G\). This implies that an optimal inspection path always exists.

We can speed up this naïve algorithm using the notion of dominance, which is used in many shortest-path algorithms (see, e.g., [48]). In our context, given two paths \(P, P'\) in our original roadmap \(G\) that start and end at the same vertices, we say that \(P\) dominates \(P'\) if \(\ell(P) \leq \ell(P')\) and \(S(P) \supseteq S(P')\). Clearly, \(P\) is always preferred over \(P'\). Thus, when searching \(G_S\), if we compute a shortest path to some node \((u, I_u)\) of length \(\ell_u\), we do not need to consider any path of length larger than \(\ell_u\) from all vertices \((v, I_v)\) such that \(I_v \subseteq I_u\). For pseudo-code describing a general \(A^*\)-like search algorithm to optimally solve the graph-inspection problem, see Alg. 1 without lines 17-27.
Fig. 3: Computing optimal inspection paths on graphs by casting a graph-inspection problem (bottom) to a graph-search problem (top). Grey layers correspond to the set of all vertices in $V_S$ that share the same set of points inspected. Edges connecting vertices in the same (different) layer are depicted in dashed (dotted) lines, respectively. The start is $(a,0)$ and the goal set $V_{\text{goal}}$ contains all vertices in the top layer. Notice that the optimal path (blue) visits vertex $a$ twice.

Fig. 4: Visualization of the notion of dominating paths by considering a path $P$ from $v_a$ to some vertex $u$ as a two-dimensional point $((\ell(P),S(P))$. Here $I_G = \{0,1,2,3\}$ and $P$ is depicted using the purple circle with $\ell(P) = 2$ and $S(P) = \{2,3\}$. (a) All paths from $v_a$ to $u$ that are $\varepsilon$-dominated by $P$ (solid red). (b) All paths from $v_a$ to $u$ that are $\varepsilon,p$-dominated by $P$. Paths that are $\varepsilon,1$-dominated by $P$ and that are $0,p$-dominated by $P$ for $p = 60\%$ are depicted in solid blue and dashed red, respectively. Dashed blue lines depict paths that are $\varepsilon,p$-dominated by $P$ for $\varepsilon > 0$ and $p = 60\%$.

While path domination may prune away paths in the open list of the $A^*$-like search, this algorithm is hardly practical due to the exponential size of the search space (recall that $|V_S| = O(|V_S| \cdot 2^{|I_G|})$). In the next sections, we show how to prune away large portions of the search space by extending the notion of dominance to approximate dominance.

2) Near-optimal planning: Let $P, P'$ be two paths in $G$ that start and end at the same vertices and let $\varepsilon \geq 0$ and $p \in (0,1]$ be some approximation parameters.

Definition 2: We say that path $P \varepsilon,p$-dominates path $P'$ if $\ell(P) \leq (1+\varepsilon) \cdot \ell(P')$ and $|S(P)| \geq p \cdot |S(P') \cup S(P')|$.

Note that it is possible that both $P \varepsilon,p$-dominates $P'$ and $P' \varepsilon,p$-dominates $P$. For a visualization of the notions of dominance and the approximate dominance, see Fig. 4.

Intuitively, approximate dominance allows to dramatically prune the search space by only considering paths that can significantly improve the quality (either in terms of length or the set of points inspected) of a given path. When pruning away (strongly-) dominated paths, it is clear that they cannot be useful in the future. However, if we prune away approximate-dominated paths, we need to efficiently account for all paths that were pruned away in order to bound the quality of the solution obtained. We do this using the notion of potentially-achievable paths or PAP’s.

Definition 3: A potentially-achievable path (PAP) $\tilde{P}$ to some vertex $u \in \mathcal{V}$ is a pair $(\ell, S)$ such that $\ell \geq 0$ and $S(u) \subseteq I \subseteq I_G$. By a slight abuse of notation, we extend the definitions of $\ell(\cdot)$ and $S(\cdot)$ such that $\ell(\tilde{P}) = \ell$ and $S(\tilde{P}) = I$.

It may seem that a PAP is simply a path but note (as the name PAP suggests) that we do not require that there exists any path $P$ from $v_a$ to $u$ such that $\ell(P) = \ell(\tilde{P})$ and $S(P) = S(\tilde{P})$. It merely states that such a path may exist.

We now use PAP’s to define the notion of a path pair:

Definition 4: Let $P$ and $\tilde{P}$ be a path and a PAP from $v_a$ to some $v \in \mathcal{V}$ such that $\ell(P) \leq \ell(\tilde{P})$ and $S(P) \supseteq S(\tilde{P})$. Their path pair is $PP := (P, \tilde{P})$ and we call $P$ and $\tilde{P}$ the achievable and potentially-achievable paths of $PP$, respectively.

Let us define operations on PAP’s and on $PP$’s, visualized in Fig. 5. The first operation we consider is extending a PAP $\tilde{P}$ by an edge $e = (u,v)$ with $S(v) = \{2\}$. (b) $PP_1$ subsuming $PP_2$. Note that $P_1$ is the achievable path of $PP_1 \oplus PP_2$ thus only the potentially-achievable path is explicitly marked.

The second operation is subsuming a path pair by another one which can be thought of as constructing a PAP that dominates the PAPs of both path pairs. Formally, Let $PP_1 = (P_1, \tilde{P}_1)$ and $PP_2 = (P_2, \tilde{P}_2)$ be two path pairs such that both connect the start vertex $v_a$ to some vertex $u \in \mathcal{V}$. The path pair defined by $PP_1$ subsuming $PP_2$ is $PP_1 \oplus PP_2 := (P_1, \min(\ell(\tilde{P}_1), \ell(\tilde{P}_2))), S(\tilde{P}_1) \cup S(\tilde{P}_2))$.

We now define the notion of bounding a path pair which will be crucial for ensuring near-optimal solutions:

Definition 5: A path pair $PP := (P, \tilde{P})$ is said to be $\varepsilon,p$-bounded for some $\varepsilon \geq 0$ and $p \in (0,1]$ if $P \varepsilon,p$-dominates $\tilde{P}$.
To compute a near-optimal inspection path (Alg. 1 and Fig. 6), we extend each path considered by our search algorithm to be a path pair and use this additional data to prune away approximately-dominated paths. Similar to $A^*$, our algorithm uses two priority queues OPEN and CLOSED to track the nodes considered by the search. It starts by inserting the start vertex $(v_s, S(v_s))$ to the OPEN list together with the path pair $P P_s = (P_s, 0)$ (here $P_s$ is a path containing only start vertex $v_s$) (line 2).

Our algorithm proceeds in a similar fashion to $A^*$—we pop the most promising node $n = (u, I_u, PP_u)$ from OPEN (line 4) and move it to CLOSED (line 5). If the PAP of this node is in the goal set $V_{goal}$ (line 6), we terminate the search and return the achievable path of $PP_u$ (line 7). If not, we consider all neighboring edges $e$ of $u$ in $G$ and extend the node $n$ (line 9). This is akin to computing $n$’s neighbors in $G_S$.

At this point our algorithm deviates from the standard $A^*$ algorithm. For each newly-created node $(v, I_v, PP_v)$ we check if there exists a node in CLOSED that dominates it. If so, this node is discarded (lines 11-14). If no such node exists in the CLOSED list, we check if there exists a node in OPEN that may subsume it. If so, that node is updated and this node is discarded (line 17-21). Finally, we check if this node can subsume nodes that are in OPEN. If so, such nodes are discarded and this node is updated. (line 24-27).

It is straightforward to see that (i) the first path pair is $\epsilon, p$-bounded and hence by induction (ii) all path pairs in the search are $\epsilon, p$-bounded. Furthermore, when the algorithm terminates, it has found a valid solution whose potentially-achievable path is in the goal set. This yields the following corollary:

**Corollary 1**: Alg. 1 computes a path $P$ that $\epsilon, p$-dominates the optimal inspection path $P^*$. Namely, that $\ell(P^*) \leq (1 + \epsilon) \cdot \ell(P)$ and $\alpha(P) \leq p \cdot [S(P) \cup S(P^*)]$.

### C. Tightening approximation factors

Recall that our algorithm starts with approximation parameters $p_0$ and $\epsilon_0$. We endow our algorithm with a tightening factor $f \in (0, 1]$, and at the $i$'th iteration we set our approximation parameters as $p_i = p_{i-1} + f \cdot (1 - p_{i-1})$ and $\epsilon_i = \epsilon_{i-1} + f \cdot (\epsilon_{i-1} - \epsilon_{i-1})$. As we will see (Sec. VI), the

---

**Algorithm 1 Near-optimal inspection planning**

**Input**: $(G_S, v_s, V_{goal}, \epsilon, p)$

1: CLOSED $\leftarrow \emptyset$
2: OPEN $\leftarrow (v_s, S(v_s), PP_s)$
3: while OPEN $\neq \emptyset$ do
4: $(u, I_u, PP_u) \leftarrow$ OPEN.extract_min()
5: CLOSED.insert$(u, I_u, PP_u)$
6: if $S(PP_u) \in V_{goal}$ then $\triangleright PP_u$ is the PAP of $PP_u$
7: return $PP_u$
8: for $e = (u, v) \in$ neighbors$(u, G)$ do
9: $(v, I_v, PP_v) \leftarrow$ extend$((u, I_u, PP_u), e)$
10: valid $\leftarrow$ True
11: for $(v, I_v', PP_v') \in$ CLOSED do
12: if $PP_v'$ dominates $PP_v$ then
13: valid $\leftarrow$ False
14: break
15: if valid then continue
16: for $(v, I_v', PP_v') \in$ OPEN do
17: if $PP_v'$ is $\epsilon, p$-bounded then
18: $(v, I_v', PP_v') \leftarrow (v, I_v', PP_v' \oplus PP_v)$
19: valid $\leftarrow$ False
20: break
21: if valid then continue
22: for $(v, I_v', PP_v') \in$ OPEN do
23: if $PP_v' \oplus PP_v$ is $\epsilon, p$-bounded then
24: OPEN.remove$(v, I_v', PP_v')$
25: $(v, I_v, PP_v) \leftarrow (v, I_v, PP_v \oplus PP_v')$
26: OPEN$\leftarrow (v, I_v, PP_v)$
27: return NULL
tightening allows our method to achieve asymptotic optimality.

D. Implementation details

1) Lazy computation in graph inspection planning: We run our search algorithm on \( G \) (Alg. 1) without checking if its edges are collision free or not (recall that only the edges of \( T \) were explicitly checked for collision). Once a solution is found, we start checking edges one by one until the entire path was found to be collision free or until one edge is found to be in collision, in which case we remove it from the edge set. This approach is common to speed up motion-planning algorithms when edges are expensive to evaluate [12, 21].

2) Node extension in graph inspection planning: Any optimal inspection path can be decomposed into a sequence of vertices that improve the coverage of the path called milestones. It is straightforward to see that an optimal inspection path will (i) terminate at a milestone and (ii) connect a pair of milestones via a shortest path in \( G \). Following this observation, instead of extending each path \( P \) from a vertex \( u \) by all outgoing edges in \( G \) (Alg. 1, line 8), we run a Dijkstra-like search from \( u \) and collect all first-met vertices that can be milestones.

3) Heuristic computation in graph inspection planning: Recall that \( A^* \) orders nodes in the OPEN list according to their computed cost-to-come added to a heuristic estimate of their cost to reach the goal. The heuristic function that we use for vertex \((u, T_u)\) is computed as follows: we run a Dijkstra search on \( G \) from \( u \) and consider the vertices \( V_u \) encountered during the search. We terminate when \((\bigcup_{v \in V_u} S(v)) \cup T_u = T_G\) and use the maximal distance between \( u \) to any node in \( V_u \) as our admissible [23] heuristic function.

VI. THEORETICAL GUARANTEES

We provide a proof sketch showing that, asymptotically, the length and coverage of the path produced by IRIS will converge to the length and coverage of an optimal inspection path. For ease of exposition, we state our results for the following simplified variant of IRIS where we start with an empty roadmap \( G_0 \) and some initial approximation factors \( \varepsilon_0 \) and \( p_0 \). At each iteration \( i \) we (i) sample a collision-free configuration \( q_i \) uniformly at random from \( \mathcal{X} \), (ii) add \( q_i \) to roadmap \( G_i \) and connect it to all samples within radius \( r_i \), and (iii) compute a near-optimal inspection path on this roadmap with parameters \( \varepsilon_i \) and \( p_i \). (Namely, instead of implicitly constructing an RRG, we implicitly construct a PRM. While not identical, both roadmaps exhibit similar properties which are typically easier to show for PRMs.)

Roughly speaking, the connection radius \( r_i \) was chosen to ensure that as \( i \to \infty \) an optimal inspection path may be traced arbitrarily well by the roadmap \( \{G_i\}_{i=1}^{\infty} \). The approximation factors were chosen such that \( \varepsilon_i > \varepsilon_{i+1}, p_i < p_{i+1}, \lim_{i \to \infty} \varepsilon_i = 0 \) and \( \lim_{i \to \infty} p_i = 1 \). This will ensure that as \( i \to \infty \) the inspection path found will converge to an optimal inspection path in \( G_i \).

A key result that we rely on is probabilistic exhaustivity (see, [49, Thm. IV.5] and [25, 51]). Roughly speaking, it is the notion that given a sufficiently large set of uniformly sampled configurations, any path can be traced arbitrarily well.

![Simulation scenarios. (a) A 5-link planar manipulator (orange) inspects the boundary of a square region (blue) where rectangular obstacles (red) may block the robot and occlude the sensor. The sensor’s field of view (FOV) is represented by the yellow region. \( S(q) \) are the points on the boundary in the sensor’s unobstructed FOV, and are shown in magenta. (b) The pleural effusion inspection scenario involves the CRISP robot (orange) inspecting the inner surface of a pleural cavity, including the POI that are covered (green) and non-covered (blue) from the current robot configuration. Finally, we assume that for every configuration along an optimal inspection path, there exists a neighborhood of configurations that share the same visibility. This is critical as we will not be able to exactly trace an optimal inspection path but only to iteratively approximate it. When this assumption holds we say that our inspection problem is well behaved.

The combination of (i) probabilistic exhaustivity, (ii) the \( \varepsilon, p \)-dominance of our graph inspection algorithm (Cor. 1) (iii) that \( \lim_{i \to \infty} \varepsilon_i = 0 \) and \( \lim_{i \to \infty} p_i = 1 \), and (iv) that our inspection problem is well behaved ensures that asymptotically, our algorithm will converge to an optimal inspection path.

VII. RESULTS

We evaluated IRIS on two simulated scenarios: (1) a planar manipulator inspecting the boundary of a square region (Fig. 7a) and (2) a CRISP robot inspecting the inner surface of a pleural cavity (Fig. 7b). All tests were run on a 3.4GHz 8-core Intel Xeon E5-1680 CPU with 64GB of RAM.

A. Planar manipulator scenario

In this scenario, depicted in Fig. 7a, we have a 5-link planar manipulator fixed at its base that is tasked with inspecting the boundary of a rectangular 2D workspace. We start by evaluating IRIS for fixed \( p \) and \( \varepsilon \) and then compare it with RRTOT using our approach for dynamically reducing the approximation factors. For every set of parameters we ran ten experiments for 1000 seconds and report the average value together with the standard deviation.

When \( p = 1 \) and we vary \( \varepsilon \) (Fig. 8a), we can see that even small approximation factors (e.g., \( \varepsilon = 0.5 \)) allow to dramatically increase the coverage obtained as each search episode takes less time and more configurations can be added to the RRT tree. While using \( \varepsilon = 0 \) did not result in 80% coverage even after 1000 seconds, this was achieved within one second for \( \varepsilon \geq 1.0 \). This comes at the price of slightly longer inspection paths. When \( \varepsilon = 0 \) and we vary \( p \) (Fig. 8b), we get roughly the same coverage per time but at the price of much longer paths for higher values of \( p \).

Following the above discussion, when reaching high coverage is the sole objective, one should use large initial values
Fig. 8: Quality of inspection paths computed for the planar manipulator. (a) IRIS running with $p = 1$, $f = 0$ and varying values of $\varepsilon$. (b) IRIS running with $\varepsilon = 0$, $f = 0$ and varying values of $p$. (c) Comparison of IRIS and RRTOT. IRIS running with two input parameter settings, both with $f = 0.03$.

of $p_0$ and $\varepsilon_0$. When we want initial solutions to also be short, one should start with smaller approximation factors. We compared IRIS with different initial approximation factors to RRTOT [6] (Fig. 8c). We can see that our approach allows to produce higher-quality paths than RRTOT. For example, IRIS obtains more than a 2450× speedup when compared to RRTOT when producing the same quality of inspection planning for the case of roughly 83% coverage and path length of 53 units. Final inspection paths obtained by IRIS are both shorter and inspect larger portions of $I$.

B. Pleural effusion scenario for the CRISP robot

The anatomical pleural effusion environment for this simulation scenario was obtained from a Computed Tomography (CT) scan of a real patient suffering from this condition, and a fine discretization of the internal surface of the pleural cavity is used as the set of POI. We also use the internal surface of the cavity as obstacles, and prohibit the robot from colliding with the pleural surface, lung, and chest wall (except at tube entry points). Pleural effusion volumes can be geometrically complex, as the way in which the lung separates from the chest wall can be inconsistent. This results in unseparated regions of the lung’s surface that can inhibit movement and occlude the sensor from visualizing areas further in the volume.

Here we consider a CRISP robot with two tubes, where a snare tube is grasping a camera tube in order to create a parallel structure made of thin, flexible tubes. Each tube can be independently rotated in three dimensions about its entry point into the body, and independently translated into and out of the cavity. The system has 8 degrees of freedom with a configuration space of $SO(3)^2 \times \mathbb{R}^2$, which enables the parallel structure to move in a manner that enables obstacle avoidance as well as precise control of the camera’s pose.

We ran IRIS and RRTOT for this scenario ten different times for 10,000 seconds (Fig. 9). Similar to the planar manipulator scenario, IRIS allows to produce higher-quality paths than RRTOT. For example, IRIS obtains more than a 25× speedup when compared to RRTOT when producing the same quality of inspection planning for the case of roughly 32% coverage and path length of 1.2 units.

Fig. 9: Comparing quality of inspection paths computed for the pleural effusion scenario. IRIS was run with $p_0 = 0.8$, $\varepsilon_0 = 10$, and $f = 0.01$.

VIII. CONCLUSION AND FUTURE WORK

In this work we presented IRIS, an algorithmic framework for computing asymptotically-optimal inspection plans. Our key contribution is an algorithm to efficiently compute near-optimal inspection plans on graphs. We showed IRIS outperforms the prior state-of-the-art, including in a medical application in which a surgical robot inspects a tissue surface inside the body as part of a diagnostic procedure.

We now highlight several avenues for further improving the IRIS framework. The first is accounting for dynamic updates in graph inspection planning. In future versions we would like to reuse information from previous search episodes. We suggest to adapt relevant methods from the graph-search literature (e.g., [18, 44, 30]). The second avenue is to efficiently sample configurations in RRT construction. Currently, in our RRT constructions we sample configurations uniformly at random from $X$. Similar to goal bias commonly implemented in RRT, we suggest to bias sampling towards configurations that increase coverage. The third avenue is to explore the influence of hyper-parameters $p$ and $\varepsilon$ and optimize them. The fourth avenue is to parallelize graph building (including edge validation) and graph searching.

ACKNOWLEDGMENT

This research was supported in part by the National Institutes of Health under award R01EB024864 and the National Science Foundation under award IIS-1149965.
REFERENCES


