Multilevel Incremental Roadmap Spanners for Reactive Motion Planning

Jeffrey Ichnowski[1] and Ron Alterovitz[1]

***Abstract*— Generating robot motions from a precomputed graph has proven to be an effective approach to solving many motion planning problems. After their generation, roadmaps reduce complex motion planning problems to that of solving a graph-based shortest path. However, generating the graph can involve tradeoffs, such as how sparse or dense to make the graph. Sparse graphs may not provide enough options to navigate around a new obstacle or may result in grossly suboptimal motions. Dense graphs may take too long to search and result in an unresponsive robot. In this paper we present an algorithm that generates a graph with multiple sparse levels—the sparsest level can be searched quickly, while the densest level allows for asymptotically optimal motions. With the paired multilevel shortest path algorithm, after the robot computes an initial solution, it can then incrementally refine the shortest-path as time allows. We demonstrate the algorithms on an articulated robot with 8 degrees of freedom, having them compute an initial solution in a fraction of the time required for a full graph search, and subsequently, incrementally refine the solution to the optimal shortest path from the densest level of the graph.**

I. Introduction

For robots to take on tasks in our everyday lives, they must be able to rapidly compute motions that avoid obstacles and lead to a goal configuration. These motions are often subject to additional constraints based upon the robot's design or the task at hand. Unfortunately, computing an optimal plan of constrained obstacle-free motions is computationally complex [1] in the general case, with complexity increasing exponentially with the robot's degrees of freedom. This complexity, combined with the desire to operate the robot in an environment that has moving obstacles, motivates the development of algorithms that either pre-compute a graph (or roadmap) of motions [2] or that can compute motions that approximate solutions rapidly. However, even when using methods that precompute a graph of feasible motions, the robot must be able to compute the shortest path on the graph fast enough to avoid moving obstacles, and ideally with a time-bounded worst-case execution time to allow for safe operation. In this paper, we present algorithms for this two-phase approach. The first algorithm pre-computes a *multilevel* sparse graph of motion (Fig. 1) that, when paired with the second algorithm, a multilevel graph search algorithm, allows a robot to rapidly find an initial path and subsequently use additional compute time to incrementally refine it to an increasingly better path.

To facilitate rapid online computation of motions, graph-based planning algorithms, such as the Probabilistic Road-Map (PRM) [2], RRG [3], and the algorithms presented here, pre-compute a graph of motions that the robot can later use to navigate an environment to accomplish a task. In the graph, vertices represent viable robot configurations, and edges represent viable motions between two configurations. A robot uses this graph to find a path from the robot's current configuration to a goal configuration by computing a graph-based shortest-path algorithm such as Djikstra's or A*. With these two-phase graph-based algorithms, much of the computational complexity of motion planning ideally goes into pre-computing the graph, allowing for fast online

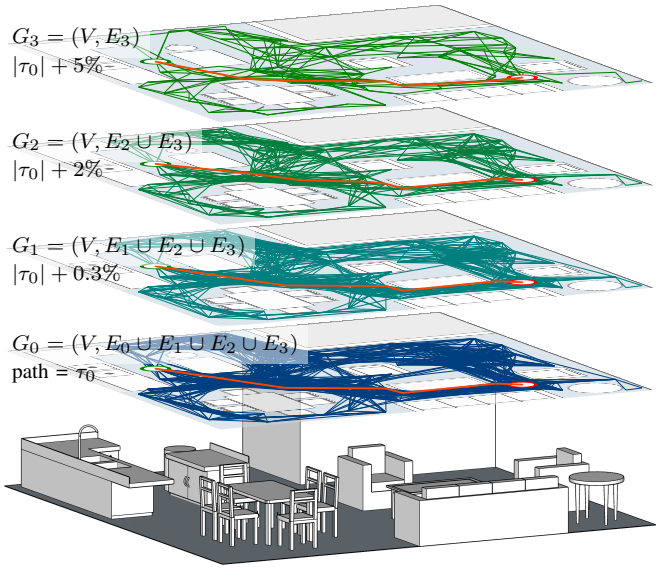

Fig. 1. Robots can use multilevel spare roadmaps to rapidly compute an initial path to a goal, and subsequently, incrementally refine that path to be shorter and shorter. In this example, a holonomic 2D disc robot uses a roadmap to compute a path along the floor while avoiding obstacles such as tables and chairs. Using the sparsest roadmap (top/green), the robot can quickly compute an initial path, and thus minimize the delay before it starts moving or before it reacts to an obstacle blocking part of the roadmap. After computing the initial path, the robot uses the increasingly dense roadmaps (from top to bottom/blue), to incrementally compute a shorter path to follow. The multilevel sparse roadmap graph has a single set of vertices and multiple sets of increasingly dense edges connecting the vertices. In this depiction, the graph has 4 levels of sparsity, with the sparsest graph G_3 at the top and the densest graph G_0 at the bottom. The sparse graph results in an initial path τ_3 that is longer by a small percentage than the optimal path in the dense graph. The algorithm then incrementally improves the path using edges from the increasingly dense graph levels, until it finds the optimal route in the densest graph. Each level increases the graph's edge count by an approximately equal number of edges ($|E_0| \approx |E_1| \approx |E_2| \approx |E_3|$)—a property of the sparse graph generation algorithm. This paper presents algorithms to both generate the multilevel sparse graph and incrementally search it.

[1]Jeffrey Ichnowski and Ron Alterovitz are with the Department of Computer Science, University of North Carolina, Chapel Hill, NC 27599, USA {jeffi,ron}@cs.unc.edu

computation of motions.

A robot that operates in *dynamic environments*, ones in which obstacles change or move over time, must continually react to changes to avoid collisions. Reactivity comes from the robot *sensing* changes to its environment, *planning* motions that take into account the sensed changes, and then *moving* according to the updated plan. This sense-plan-move loop is typically computationally-bounded by the computational complexity of planning—thus in graph-based algorithms, computing the shortest path is one of the critical algorithms that must be re-executed every time a new plan must be computed. When computing the shortest path is slow, the robot cannot react quickly to changes in the environment, and thus, at best, moves suboptimally, and at worst, collides with an obstacle. Thus, for reactivity in a dynamic environment, planning must be fast. With graph-based shortest-path algorithms, computational complexity is dominated by the branching factor of the graph—the more edges per vertex, the slower the shortest-path computation. To allow for fast shortest-path computation, the algorithm presented here generates a multilevel graph in which each level adds more edges to the graph, and the sparsest graph allows for the rapid online computation of shortest paths.

When a robot operates in a *safety-critical* environment, that is, when delays in planning can result in harm to a human, planning must not only be fast but must have an analytically bounded worst-case computation time. It can be the case that planning in the average or expected case is fast, but is (extremely) slow in some cases—and unfortunately, such cases can occur when a collision is imminent. One of the benefits of our proposed method is that the sparsest level of the graph can be analytically bounded to produce a worst-case computation time that is appropriate for safety-critical operation.

While worst-case bounds on computation time are beneficial for guarantees on reactivity and safety, the expected or average case for graph search can often be much faster—meaning that the robot could compute a better path with a denser graph. Using a variant of Djikstra's shortest-path algorithm, the algorithms presented here allow the robot to perform an online incremental refinement of its path as time allows.

This refinement is further enhanced by the asymptotic optimality of the graph generation algorithm presented. Asymptotic optimality means that given enough computation time, the graph will be optimal according to a cost function, with probability 1. While an infinite graph is inappropriate for fast online computation, this convergence property is useful in ensuring that creating larger graphs, and later searching graphs, will result in shorter paths.

II. Related Work

Kavraki et. al proposed Probabilistic Road-Maps (PRM) [2] as a sampling-based method to generate a graph of motions for robots with high-dimensional configuration spaces. This approach has proven effective at generating solutions to complex motion planning problems. Karaman and Frazzoli proposed kPRM* as an asymptotically-optimal variant of PRM, that with enough computation time, produces an optimal graph with probability 1. In the same work, they also propose an incrementally constructed variant of kPRM* called RRG [3].

Marble and Bekris introduced the Incremental Roadmap Spanner (IRS) [4] planning algorithm that follows the outline of kPRM* and RRG, but only includes a subset of edges, thus producing a *sparse* graph. The sparse graph has an asymptotic *near*-optimal guarantee, but with fewer edges in the graph, it allows for faster graph-based shortest path computation. Dobson and Bekris extend this work to building sparse roadmaps by using additional criteria to leave out parts of the graph [5].

In our previous work [6], a cloud-based computer computed a parallelized version of IRS that also kept the underlying dense RRG graph. This graph allows the cloud-based algorithm to selectively send both sparse and portions of the dense graphs based upon the robot's progress through a task. In this paper, we extend beyond the two-levels of the graph in both IRS and prior work to an arbitrary number of levels.

Several algorithms allow for the rapid initial computation of a graph-based shortest path and subsequent refinement. A*, a heuristic-enhanced shortest path algorithm serves as the basis for Anytime Repairing A* (ARA*) [7]. ARA* uses an inadmissible heuristic that finds an initial solution rapidly, and in subsequent iterations, refines the heuristic and thus solution, until it produces a graph-based optimal solution with an admissible heuristic.

D* [8] and a subsequent algorithmic simplification in D*-lite [9] compute a graph-based shortest path and keep around information for a subsequent search. In a subsequent search, after edge weights change on the graph, D* and D*-lite can reuse information from the previous search to speed up their subsequent refinement. Anytime Dynamic A* [10] builds upon D* and ARA* by continually improving a solution while computation time allows.

Hierarchical motion planners speed up planning on large or complex problems by breaking the motion planning problem down into a hierarchy of problems using a variety of approaches. Cowlagi et al. [11] propose wavelet decomposition, Hwang et al. [12] propose a mesh-based simplification, and Multi-Scale LPA* [13] searches a regular grid hierarchy. These approaches often require the planning problem to have a prescribed structure to the problem (e.g., $\mathbb{R}^n$ configurations), whereas the planner proposed herein can be applied to the same broad class of problems as kPRM* and RRG. Other hierarchical motion planners, such as PRM-RL [14], use a hierarchy of different planners to address different problems (e.g., PRM for planning and reinforcement learning for sensing and actuation variance).

III. Problem Statement

Let $q \in C$ be a vector that represent the complete configuration of a robot (e.g, joint angles, position and orientation in space, etc.), where C is the set of all configurations. Let $C_{obs} \subset C$ be the set of configurations that are invalid

e.g., due to obstacles or task- or robot-based constraint violations. The set $C_{\text{free}} = C \setminus C_{\text{obs}}$ is thus the set of valid or obstacle-free configurations. Let $C_{\text{goal}} \subset C_{\text{free}}$ be the set of goal configurations. Let $L(\ ; q_a; q_b) : [0;1] \to C$ be a problem-specific *local planner* that computes a trajectory interpolated over $[0;1]$ as parameterized by configurations q_a and q_b, with $L(0; q_a; q_b) = q_a$ and $L(1; q_a; q_b) = q_b$. The objective of *motion planning* is to compute a path $= \{q_0; q_1; \ldots; q_n\}$ such that q_0 is the robot's initial or current configuration, $L(t; q_i; q_{i+1}) \in C_{\text{free}}$ for all $i \in [0; n)$ and $t \in [0;1]$, and $q_n \in C_{\text{goal}}$.

Let $G = (V; E)$ be a graph, such that $V \subset C_{\text{free}}$ is the set of vertices and E is the set of edges, where $L(t; q_u; q_v) \in C_{\text{free}}$ for all $(q_u; q_v) \in E$ and $t \in [0;1]$. When an *asymptotically-optimal* roadmap motion planner adds additional vertices and edges in G, the lengths of the shortest paths in G remain unchanged or shrink. The objective of a reactive graph-based motion planner is to make the worst-case shortest-path computation fast enough to guarantee a computation within a specified time bound. The objective of the multilevel sparse roadmap algorithms here is to generate an asymptotically-optimal roadmap that allows for reactive motion planning.

IV. Method

The objective of the multilevel sparse roadmap algorithm is to generate a graph that may be used later to plan valid motions for a robot. The vertices in the graph represent valid robot configurations and the edges represent valid trajectories between two vertices. The roadmap generation algorithm in Sec. IV-A generates multiple levels of increasingly dense graphs, each of which shares the same set of vertices, and each level has an increasing number of edges. After generating the roadmap, a robot can follow the shortest valid path on the graph to accomplish a task, by incrementally refining its path using the increasingly dense levels of the graph as outlined in Sec. IV-C.

A. Multilevel Sparse Roadmap Generation

In this section we describe the form and generation of the multilevel sparse roadmap.

The multilevel sparse roadmap algorithm generates a graph of the form $G = (V; (E_L; E_{L-1}; \ldots; E_0))$, in which V is a set of valid robot configurations, and E_i for $i \in [0; L]$ are the sets of valid trajectories between vertex pairs (as computed by the local planner), and L is a configurable parameter that defines the number of sparse graphs to generate in addition to the dense graph. The subscript on E is the sparsity level, where E_L is the set of edges of the sparsest graph, and $E_L \subset E_{L-1} \subset \cdots \subset E_0$ is the set of edges of the densest graph. Note that by construction, each increasingly dense level is a superset of less sparse levels.

Algorithm 1 outlines the generation of the multilevel sparse roadmap. It starts by initializing an empty graph in line 1. While the presented algorithm starts with an empty set of vertices, in practice it may be desirable to initialize the graph with a single start configuration. Adding more than one pre-specified configuration to V (e.g., to force specific waypoints), however, requires that lines 5 to 23 to be executed for each pre-specified configuration beyond the first.

Algorithm 1 MultilevelSparseRoadmap(L)

Require: $L \in \mathbb{Z}^+$ is the number of sparse graph levels to generate

1: $G = (V; (E_L; E_{L-1}; \ldots; E_0)) \quad (?; (?; ?; \ldots; ?))$
2: **while** not done **do**
3: $q_{\text{rnd}} \quad$ random sample
4: **if** $q \in C_{\text{free}}$ **then**
5: $N \quad k$-nearest neighbors of q_{rng} using kRRG
6: $V \quad V \cup \{q_{\text{rnd}}\}$
7: $N' \quad \{q_i \in N : L(t; q_{\text{rnd}}; q_i) \in C_{\text{free}}\ \forall t \in [0;1]\}$
8: $m_{\text{total}} \quad |N'| + \sum_{\ell=0}^{L} |E_\ell|$
9: **for** $\ell \quad L$ down to 1 **do**
10: $m_\ell \quad \min(|N'|; \lfloor m_{\text{total}}/(\ell+1)c \rfloor)$
11: **if** $\ell = L$ **and** $m_\ell = 0$ **then**
12: $m_\ell \quad 1$
13: $m_{\text{total}} \quad m_{\text{total}} \quad m_\ell$
14: **while** $|E_\ell| < m_\ell$ **do**
15: $T \quad$ shortest path lengths from q_{rnd} to N'
16: **if** $\max T = \infty$ **then**
17: $q_n \quad \operatorname{argmin}_{q: T(q) = \infty} \text{distance}(q_{\text{rnd}}; q)$
18: **else**
19: $q_n \quad \operatorname{argmax}_q T(q)$
20: $E_\ell \quad E_\ell \cup \{(q_{\text{rnd}}; q_n)\}$
21: $N' \quad N' \setminus \{q_n\}$
22: **for all** $q_n \in N'$ **do**
23: $E_0 \quad E_0 \cup \{(q_{\text{rnd}}; q_n)\}$

Following the patterns of kPRM* and RRG, after initializing the graph, the algorithm repeatedly generates random samples (line 3) and adds valid samples to the graph, stopping once a termination criterion is met (line 2). For the multilevel sparse roadmap, the termination criterion should be based on the complexity of the sparest graph reaching the maximum tolerable for maintaining reactivity when following the roadmap. Since the time to follow the roadmap is a function of $|V|$ and $|E_L|$ this computation may be a simple polynomial with constants determined by an appropriate analysis tool. Since robots, their tasks, processors, and algorithm implementations vary widely, determining the exact termination criteria for a desired level of reactivity is beyond the scope of this paper.

After generating a valid random sample, the algorithm then adds the sample to the graph. This process starts by computing the k-nearest neighbors in V of the random sample, and then discarding neighbors for which the local planner does not compute a path in C_{free} (line 7). The value for k in line 5 that ensures asymptotic optimality is: $k \quad \lceil k_{\text{RRG}} \log(|V|+1) \rceil$, where $k_{\text{RRG}} = e + e/d$, and d is the dimensionality of C [3]. This follows from kPRM and RRG algorithms. But whereas kPRM* and RRG would add edges from the random sample to k neighbors in a single graph, the multilevel spare roadmap adds edges to all k neighbors

but splits them among $L + 1$ graph levels.

There are multiple possible criteria for distributing the candidate edges among the $L + 1$ levels of the graph. For brevity, we present a criterion that results in an even distribution of edges between levels. Regardless of the distribution, though, the implementation of the algorithm should ensure that at least one edge is added to the sparsest graph per vertex so that the sparsest graph is a connected graph. Also, an edge that leads to connecting disconnected components of the graph should be included in the sparsest level of the graph. The aforementioned criterion is implemented by the loop that starts on line 9. At each level, it computes a fraction of the total edges that the level must have to evenly distribute the edges (lines 10 and 13). The special case of requiring at least one edge in the sparest graph is handled by the conditional in line 12.

Once the algorithm determines the desired number of edges for a level, it then adds one edge at a time to the sparse edge set of that level in the loop on line 14. It adds edges according to the criteria of either increasing connectivity or reducing the longest path to a vertex. To evaluate these criteria, it computes the shortest path from the random sample to all remaining candidate edges (line 15). If any edge cannot be reached, the algorithm will add an edge to the nearest remaining neighbor that cannot be reached (line 17). If all remaining neighbors can be reached, the algorithm will add an edge to the one reachable by the longest path (line 19). In the first iteration, the random sample is not connected to the graph, so there will be no shortest path. It should be noted that this is similar to the Incremental Roadmap Spanner [4], which instead compares the shortest path to a multiple of the length of a direct connection.

As the algorithm adds edges to each increasingly dense level of the roadmap (line 20), it also removes them from the candidate edge list (line 21). After adding edges to all the sparse levels it adds any remaining edges to the densest level E_0 in line 23, thus ensuring that the densest graph is equivalent to the graph generated by RRG.

Once the remaining candidate edges are in the graph, the algorithm proceeds to the next iteration, repeating the process and incrementally growing the graph. During the process, the distribution of edges between the sparsity levels may vary from the desired distribution due to graph connectivity, obstacles, and or other motion constraints. However, in the limit, the proposed algorithm will ensure that the desired sparsity levels are achieved as it generates and adds more highly connectable vertices.

B. Asymptotic Optimality of the Roadmap

The multilevel sparse roadmap algorithm presented here will generate an asymptotically-optimal dense graph. This follows from the construction of the graph following the same principles as kPRM and RRG. With the same random sample sequence, the multilevel sparse roadmap algorithm presented here will generate the same set of edges as the asymptotically-optimal RRG.

Algorithm 2 MultilevelShortestPath($q_{start}; q_{goal}; G$)

Require: $q_{start} \in V$, $q_{goal} \in V$,
$G = (V; (E_L; \ldots; E_0))$ is a multilevel sparse graph
1: $g(q_{goal}) \leftarrow 0$ // length of the shortest path
2: $\ell_{current} \leftarrow L$ // start at the sparsest level
3: **repeat**
4: $Q \leftarrow \{q_{goal}\}$
5: **while** $|Q| \neq 0$ **and** $q_{start} \notin \operatorname{argmin}_{q \in Q} g(q)$ **do**
6: $q_{min} \leftarrow \operatorname{argmin}_{q \in Q} g(q)$
7: $Q \leftarrow Q \setminus \{q_{min}\}$
8: **for** $\ell \leftarrow L$ down to $\ell_{current}$ **do**
9: **for all** $(q_{min}, q') \in E_\ell$ **do**
10: $c \leftarrow g(q_{min}) + \text{cost}(q_{min}, q')$
11: **if** $q' \notin Q$ **or** $c < g(q')$ **then**
12: **if** valid(q_{min}, q') **then**
13: $Q \leftarrow Q \cup \{q'\}$
14: $g(q') \leftarrow c$
15: **if** significant changes in environment **then**
16: $\ell_{current} \leftarrow L$
17: **else**
18: $\ell_{current} \leftarrow \ell_{current} - 1$
19: **until** $\ell_{current} < 0$ **or** maximum time elapsed

C. Multilevel Sparse Roadmap Shortest Path

The goal of the path-finding algorithm is to find an initial path in the sparse graph rapidly. This property allows the robot to react to changes in the environment at a predictable rate. As computation time allows, either because the computation time to find the initial solution was better than worst-case, or because the environment does not change as the robot moves, the path-finding algorithm will incrementally improve the solution by using edges from the increasingly dense graphs.

The algorithm that accomplishes this goal on the multilevel sparse roadmap is presented in Alg. 2 and is based on Djikstra's shortest path algorithm with lazy collision detection. In this algorithm, the `valid( )` method on line 12 checks an edge for collision with a sensed obstacle. (Ideally, obstacles and constraints known at roadmap generation time will have been incorporated into the roadmap, and thus not need to be checked here.) When collision detection is computationally expensive, the implementation of `valid( )` should perform cached lazy evaluation of the collision—that is, the expensive computation will be performed once per unique set of arguments, and thus subsequent evaluation is fast.

The outer loop of the algorithm (re-)evaluates the shortest path at increasingly dense sparsity levels ($\ell_{current}$). The first iteration only considers edges from the sparsest graph level. Sparse graphs have fewer edges, and thus the shortest path algorithm requires fewer computationally expensive calls to `valid` in the inner loop (described next). Each time the outer loop decrements $\ell_{current}$, the inner loop has to consider an increasing number of edges. However, since previous iterations performed the expensive evaluation of the edges from the previous sparsity levels, the inner loop will only

need to perform an incremental number of expensive edge evaluations.

The inner loop of Alg. 2 evaluates the shortest path for the current sparsity level ℓ_{current}. It operates by maintaining a priority queue Q of vertices to evaluate, prioritized based on the shortest path to the vertex. The algorithm initially populated Q with the single vertex q_{goal}, since it searches from the goal to the start. Each iteration of the loop on line 5, removes the vertex with the shortest path from the queue and evaluates the paths of its outgoing edges. This evaluation only considers edges from the current level of the sparse graph, thus the number of edges will proportionally increase as the algorithm increases the sparsity level. Any vertex that has the shortest path through the evaluated edge will be (re-)queued in the priority queue.

In different applications other shortest path algorithms, such as A*, and incremental shortest path algorithms, such as D*-Lite [9], may allow for more efficient operation, e.g., due to the nature of the sensing and collision detection routines.

V. Results

We evaluate our method's ability to enable a robot to quickly compute an initial motion and then incrementally refine it. For this evaluation, we give a Fetch robot the task of carrying an open-top container without spilling its contents. To perform this task, the robot uses its 7-joint articulated arm and single torso prismatic lift joint, for a total of 8 degrees of freedom. The robot and example of this task are shown Fig. 2. We precompute a multilevel sparse roadmap offline. In this precomputation process, the sampling routine generates robot configurations (as vertices) that hold the container upright, and the local planning routine generates trajectories (as edges) that keep the container upright throughout the motion. In some cases, the local planner is unable to generate such a motion (e.g., due to self-collision in the interpolated path), and the edge is left out of the graph. Using the precomputed roadmap, we later generate and have the robot solve random motion planning tasks of varying difficulty and traversed distance. In these tasks, the robot's starts at a random configuration from the graph in which its end effector lies to the right of the robot and computes a path to a goal configuration in which its end effector lies to the left of the robot and above a table (thus no tasks must traverse purely below the table). The robot uses the multilevel shortest path algorithm while performing lazy collision detection on obstacles in its environment.

To evaluate the proposed method's ability to quickly generate solutions, we record and plot the time required for a shortest-path search at each sparsity level over 100 runs, and compare to the time required to search a fully dense graph non-incrementally. From the results plotted in Fig. 3, we can see that each incremental refinement requires a fraction of the time that the full non-incremental search requires. The time savings could then be passed along in the form of allowing the robot to start moving towards its goal sooner.

To demonstrate that the proposed method can incrementally refine the shortest path, we also record the shortest path length and plot the results in Fig. 4. This plot shows that the low-quality initial solution is quickly refined in subsequent iterations by searching the increasingly dense graphs. When the incremental search reaches the densest graph, it produces the same shortest path as a full (non-incremental) search does.

To help visualize the impact of the incrementally refined shortest path, we plot the joint trajectories for a single example task in Fig. 5. The initial path computed on the sparest graphs requires 17.1 seconds to follow. As the multilevel shortest path algorithm refines the shortest path using increasingly dense graph levels, the computed trajectory shortens and smooths out, until it reaches the densest graph and finds a path requiring 8.4 seconds to follow.

The results described above demonstrate an advantage over the default approach of creating a single PRM. When one creates a single PRM, the robot will either have (1) a shortest-path solver that is too slow due to the graph being too large or dense (e.g., "full" in Fig. 3), or (2) find paths of poor quality due to the graph being too small or sparse.

VI. Conclusion

In this paper, we presented an algorithm that generates an asymptotically-optimal roadmap as a graph with multiple levels of sparse edges We also presented an algorithm that allows a robot to use the multilevel sparse roadmap to quickly find an initial shortest path and incrementally refine it later. When compared to a non-incremental shortest-path algorithm, the incremental version finds initial solutions faster, allowing a robot to start moving sooner. Given additional compute time, the incremental shortest path finds shorter and shorter paths until it reaches the densest level—at which point the path is equivalent to one found by a non-incremental search of the densest graph. These properties can allow a robot to react more quickly to changes in its environment while retaining the ability to generate asymptotically-optimal shortest paths. We demonstrated this on an 8 degree of freedom task computed for the Fetch robot.

In future work, we plan to explore varying the number of edges in each sparse level of the graph, as well as dynamically selecting the number of sparse levels. These changes could readily be applied to the presented algorithm and could allow a robot to select a multilevel sparse roadmap setup to match robot or problem-specific requirements—e.g., how quickly the sparsest levels can be searched, and how rapidly subsequent refinement shortens the shortest path.

We also plan to explore speeding up the shortest-path search by integrating it with approaches from other incremental and heuristic graph search algorithms. We believe these approaches would further extend the benefits of the proposed multilevel sparse graph. Related to these other incremental and heuristic-based search algorithms lies the notion that the robot could start moving before it has had enough time to refine the shortest path to the optimal one from the dense graph. In future work, we also plan to explore incrementally refining the motion after the robot starts moving on a sub-optimal path from a sparse graph.