Online High-Level Model Estimation for Efficient Hierarchical Robot Navigation

Martina Stadler, Katherine Liu, and Nicholas Roy

***Abstract*— We would like to enable a robot to navigate efficiently and robustly in known, structured environments that are large enough to cause traditional planning approaches to incur considerable computational cost. Hierarchical planners are a promising way to increase planning efficiency in such environments because high-level abstract plans can be used to reduce the size of the search space over which detailed planning occurs. However, useful high-level representations of planning problems can be challenging to generate without prior domain knowledge. In this work, we propose a high-level planning representation which can be learned from previous plans considered in the environment and used online during hierarchical, multi-query robot navigation. We treat previous planning results as noisy measurements of high-level navigation properties, then update these properties over time using recursive estimation. We test our approach in standard and risk-aware hierarchical planning schemes, and demonstrate up to an 86% decrease in the number of nodes expanded and a 66% decrease in wallclock time as compared to a baseline A* planner while finding plans that are only 2-10% more expensive.**

I. Introduction

We would like to enable a robot to navigate efficiently and robustly in large, structured environments. Discrete navigation approaches, such as planners that run the A* algorithm over a fixed-size occupancy grid [1], can be inefficient in large environments due to the *curse of scale*, or the exponential relationship between plan length and planning complexity in discrete environments. Variants of the A* algorithm can use heuristics based on local environmental geometry to improve planning performance, but are often tuned to specific types of environmental structure [2]. Sampling-based motion planners [3]–[5] are efficient in large environments when naive sampling strategies generate random graphs that are representative of the set of possible traversals in the environment. However, these approaches can struggle to generate connected graphs in structured environments without employing problem-specific sampling strategies [6]–[9], which can be challenging to specify. Multi-query planners reduce computational cost across multiple planning trials by precomputing, recording and reusing planning artifacts during online planning, but are mainly focused on minimizing duplicate computations locally and do not fundamentally reduce the dimensionality of the planning problem or enable enhanced reasoning about the global quality of candidate plans [10], [11].

All authors are with the Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology in Cambridge, USA. {mstadler,katliu,nickroy}@mit.edu

This research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-17-2-0181. Their support is gratefully acknowledged.

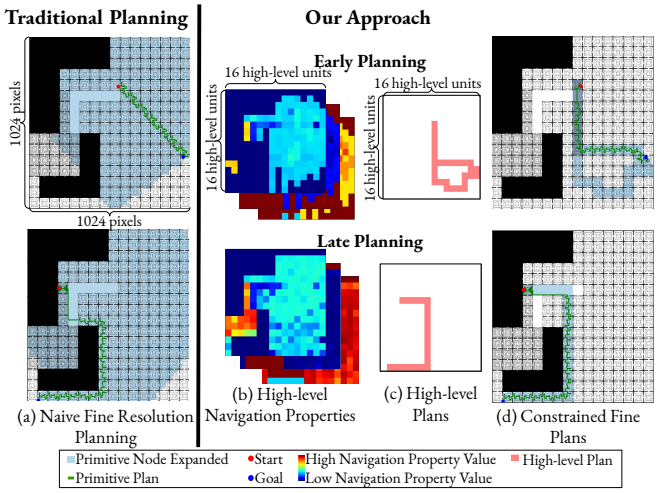

Fig. 1: **Online High-Level Function Estimation for Efficient Hierarchical Robot Navigation.** Rather than planning at a single fine resolution (a), we present an approach which estimates high-level navigation properties (b) online, using only previous planning computations, to find high-level plans (c) in a coarse representation of the planning problem. Then, we use high-level plans to constrain planning in the fine resolution planning problem (d). We demonstrate that our approach increases planner efficiency while ensuring planner robustness over the course of multiple queries to the planner.

Hierarchical planners can minimize the curse of scale by generating coarse representations of the planning problem whose solutions can be used to guide a detailed planner to regions of the environment that are likely to contain low-cost detailed solutions. However, coarse planning is only useful if the coarse planning problem efficiently captures properties of the environment necessary to inform intelligent detailed planning. Some approaches use geometry-based decompositions of the environment, such as convex regions [12], to generate abstract representations of the planning problem which are guaranteed to accurately summarize all possible detailed plans in the environment, but these approaches can be overly restrictive when geometry (e.g., the set of convex regions) is poorly aligned with the geometric features of the environment needed to compute a feasible motion plan efficiently. Other approaches use techniques such as wavelet decompositions [13] or the information bottleneck [14] to compress known cost maps for multi-resolution planning, but do not explicitly capture the structure of the paths agents use when traversing in the environment. Recently, some approaches have turned to learning to generate coarse representations of planning problems based on offline datasets of primitive plans and additional environmental cues, such as semantically segmented overhead images [15] or

height maps [16]; however, such additional environmental cues can be challenging to acquire in real-world environments.

While existing hierarchical planners can plan efficiently using static, coarse representations that are high quality and a good approximation of select environments, these planners still fail to capture the often lossy nature of plan abstraction for planning in complex environments. Risk-aware planning is a well studied technique that has been used to improve planning performance when the model of a planning problem is known to be incomplete or noisy, such as risk-aware planning over probabilistic cost maps [17], which enables planners to trade off between planning efficiency and expected plan quality in a principled manner. Similar approaches have been extended to the online setting, where dynamically discovered plan costs influence future risk-based planning decisions [18]. We hypothesize that explicitly representing the uncertainty in coarse or hierarchical representations can enable efficient, robust planning, even when the coarse representation is not initially well aligned with the true costs of plans in the underlying detailed planning environment.

The act of recovering a coarse planning representation from previous experience has been studied in the reinforcement learning community as the macro-action learning problem. However, in most approaches, such as [19], experience from online executions is used to learn a model-free representation of the environment via a Q function. In this work, we formulate macro-action discovery for navigation as a model-based estimation problem which uses previous planning computations, not past plan executions, to update a coarse representation of the planning problem.

In this work, we propose a hierarchical planning representation which updates properties of a coarse representation using planning results generated in a detailed representation of the planning problem, and which can be used during uncertainty-aware hierarchical multi-query robot navigation. We select a simple environmental decomposition to generate a coarse planning representation, then recognize that previous planning results in the environment can be treated as noisy measurements of coarse navigation properties. We use the measurements to improve estimates of the navigation properties over time. We test our approach in standard and risk-aware hierarchical planning schemes, and demonstrate increased planning efficiency over time as compared to a baseline A* planner while maintaining planner robustness.

II. Online High-Level Model Estimation for Efficient Hierarchical Robot Navigation

In this section, we review the definition of the hierarchical planning problem, then discuss the online generation and use of a coarse representation for the navigation problem. Finally, we discuss two variants of a hierarchical planner which use the coarse representation.

A. Problem Formulation: Hierarchical Planning

Consider a robot with state x defined in some continuous configuration space $\mathcal{X} \in \mathbb{R}^d$, where d is the dimensionality of the state. The goal of the motion planning problem is to find a valid (i.e., non-colliding and dynamically feasible) plan that takes the robot from an initial state x_s to a goal state x_g, where $x_s, x_g \in \mathcal{X}$, while minimizing a cost function of the plan, such as time or distance. We can approximate a continuous plan as a sequence of discrete *primitive* actions $a \in \mathcal{A}$, where we define each action as an explicitly encoded trajectory with a specific beginning state $x_i = b(a)$ and a specific ending state $x_j = e(a)$, where $x_i, x_j \in \mathcal{X}$. Formally, we define a sequence of primitive actions as a primitive plan, $p = \{a_0, a_1, ..., a_{n-1}\}$, where n is the number of actions in the plan, and $p \in \mathcal{P}$, where $\mathcal{P}$ is the space of all possible discretized primitive plans. We will also refer to the sequence of actions in a plan as $a_{0:n-1}$. We define the discrete optimal planning problem,

$$\begin{aligned} p^* = \arg\min_{p \in \mathcal{P}} \ & \sum_{i=0}^{n-1} c(a_i; \Gamma) \\ \text{s.t.} \ \ & f(a_i; \Gamma) = 1 \ \forall i \in \{0, 1, ..., n-1\} \\ & b(a_i) = e(a_{i-1}) \ \forall i \in \{1, ..., n-1\} \\ & b(a_0) = x_s, e(a_{n-1}) = x_g, \end{aligned} \tag{1}$$

where f is a partition function that maps every $a \in \mathcal{A}$ to feasibility, $f : \mathcal{A} \to \{0, 1\}$, where $f(a; \Gamma) = 0$ means the trajectory from the start state in a to the end state in a is infeasible, c is a scalar cost function defined over primitive actions, $c(a; \Gamma) \in \mathbb{R}$, and Γ is additional inputs, such as the kinematic model and the map. Throughout this work, we will refer to the tuple, $\{c(a_i; \Gamma),\ f(a_i; \Gamma)\}$, as the primitive *navigation properties* of action a_i.

In practice, optimizing over all plans $p \in \mathcal{P}$ can be intractable as the length of the largest dimension of the configuration space and the potential length of any plan increases, since discrete planning complexity scales exponentially with plan length. Hierarchical planning algorithms increase the tractability of motion planning at long length scales by representing the plan space using a combination of primitive and *high-level* actions $\mathbf{a} \in \mathbf{A}$, which are actions based on simplified models of the planning problem that do not necessarily capture all planning constraints[1]. While various hierarchical planning methods have been proposed (e.g., [20]–[23]), one approach is to use high-level actions to generate high-level plans $\mathbf{p} \in \mathbf{P}$ which approximate solutions to (1) [23], where $\mathbf{p} = \{\mathbf{a}_0, \mathbf{a}_1, ..., \mathbf{a}_{\mathbf{n}-1}\}$, $\mathbf{n}$ denotes the number of high-level actions in the high-level plan, and $\mathbf{P}$ denotes the high-level plan space. Formally, we write the high-level planning problem,

$$\begin{aligned} \mathbf{p}^* = \arg\min_{\mathbf{p} \in \mathbf{P}} \ & \sum_{i=0}^{\mathbf{n}-1} \mathbf{c}(\mathbf{a}_i; \mathbf{\Gamma}) \\ \text{s.t.} \ \ & \mathbf{f}(\mathbf{a}_i; \mathbf{\Gamma}) = 1 \ \forall i \in \{0, 1, ..., \mathbf{n}-1\} \\ & \mathbf{e}(\mathbf{a}_{i-1}) \subseteq \mathbf{b}(\mathbf{a}_i) \ \forall i \in \{1, ..., \mathbf{n}-1\} \\ & x_s \in \mathbf{b}(\mathbf{a}_0), x_g \in \mathbf{e}(\mathbf{a}_{\mathbf{n}-1}), \end{aligned} \tag{2}$$

where $\mathbf{f}$ is a partition function that maps every high-level action in $\mathbf{A}$ to its validity, $\mathbf{f} : \mathbf{A} \to \{0, 1\}$, $\mathbf{c}$ is a scalar cost

[1]Throughout this work, we use italic type to denote primitive variables and functions, and bold type to denote high-level variables and functions.

function defined over high-level actions, $\mathbf{c}(\mathbf{a};\mathbf{\Gamma}) \in \mathbb{R}$, $\mathbf{\Gamma}$ is additional inputs, and $\mathbf{b}(\mathbf{a}_i)$ and $\mathbf{e}(\mathbf{a}_i)$ denote the set of valid start states and terminal states for high-level action $\mathbf{a}_i$.

While there are multiple ways to solve the optimizations in (1) and (2), one common approach is to represent the optimizations as two separate graph search problems, a high-level graph search problem and a primitive graph search problem, where graph nodes represent states and weighted, directed graph edges represent actions in the respective optimizations. Planning begins with high-level graph search, and continues until a low-cost high-level plan $\mathbf{p}$ is found. The high-level plan is then used to increase the efficiency of primitive graph search by guiding planning to primitive subgraphs which are likely to contain low-cost primitive solutions (e.g., by using high-level plan costs as heuristics [24], or by constraining primitive search to subgraphs that are consistent with the high-level plan [21], see Figure 1(c)-(d)), reducing the total computation required to find such solutions. When a search process uses a high-level plan $\mathbf{p}$ to guide the search for a primitive plan p, we call the resulting primitive plan p a primitive *refinement*.

While hierarchical planning can be efficient, it is obvious that the quality of the high-level planning problem, or the combination of $\mathbf{A}$, $\mathbf{c}(\mathbf{a};\mathbf{\Gamma})$, and $\mathbf{f}(\mathbf{a};\mathbf{\Gamma})$, dictates the ability of the high-level search process to effectively guide primitive planning. Intuitively, a good choice of representation is one that ensures *plan similarity*[2], or that low-cost, feasible plans in $\mathbf{P}$ have low-cost, feasible primitive refinements in $\mathcal{P}$. Unfortunately, generating planning representations which exhibit plan similarity can be challenging, and in practice such representations are often hand-designed for specific problems [24], or require significant prior domain knowledge to be generated [12], [15], [16]. Instead, we define a simple $\mathbf{A}$ and use online planning results to infer estimates of $\mathbf{c}(\mathbf{a};\mathbf{\Gamma})$ and $\mathbf{f}(\mathbf{a};\mathbf{\Gamma})$ over the course of a multi-query planning trial. An overview of our approach is given in Figure 2.

B. The High-Level Action Space

The goal of the high-level action space is to generate a small (e.g., low-resolution) representation of the planning problem that enables efficient planning without compromising plan quality. In this work, we recognize that different optimal primitive plans in structured environments often share common primitive *subplans*, or partial sequences of primitive actions $a_{l:m}$, when navigating through the same region in the environment. By grouping these subplans into high-level actions $\mathbf{a}$, we can generate a compact, flexible high-level action space $\mathbf{A}$ which uses the primitive navigation properties of similar, past planning queries to improve the high-level estimates of navigation properties for use in current and future planning queries. Formally, we define the high-level action space by assuming that the configuration space $\mathcal{X}$ can be decomposed into a finite set of regions[3] $\mathcal{R}$ that an agent can traverse between, and define a high-level action $\mathbf{a}_{ij}$ as the set of primitive subplans that begin in and remain in region $\mathcal{R}_i$ until terminating in region $\mathcal{R}_j$, similar to the definition given by Vega-Brown and Roy[4] [12],

$$\mathbf{a}_{ij} = \{p | p \in \mathcal{P},\ a_k \in p,\ b(a_k) \in \mathcal{R}_i\ \forall k \in \{0, 1, ..., n-1\}, \\ e(a_{n-1}) \in \mathcal{R}_j\}. \quad (3)$$

C. Primitive Subplans as Instances of High-Level Actions

By representing high-level actions as sets of primitive plans, we can use results of planning in the primitive action space $\mathcal{A}$ (e.g., occupancy map) to induce costmaps over the high-level action space $\mathbf{A}$. When we generate a primitive refinement p of high-level plan $\mathbf{p}$, where $p \in \mathcal{P}$, we observe a sequence of primitive actions in the environment, $p = \{a_0, a_1, ..., a_n\}$, where $a_i \in \mathcal{A}$, subject to the high-level planning constraints, and we also observe whether or not the refinement meets the constraints in (1). However, because we define high-level actions $\mathbf{a} \in \mathbf{A}$ as sets of primitive subplans, we can also post-process a primitive refinement p into a sequence of primitive subplans, $p = \{p_0^s, p_1^s, ..., p_{\mathbf{n}}^s\}$, where each subplan p^s is a sequence[5] of primitive actions $a_{l:m}$ that obeys the constraints of a high-level action $\mathbf{a}$ as defined in Equation 3,

$$p_{ij}^s = \{a_{l:m} | a_k \in p,\ b(a_k) \in \mathcal{R}_i\ \forall k \in \{l, l+1, ..., m\}, \\ e(a_m) \in \mathcal{R}_j\}. \quad (4)$$

This formulation allows us to treat a primitive subplan p_{ij}^s collected in an environment as an example high-level action $\mathbf{a}_{ij}$ which transitions from region $\mathcal{R}_i$ to $\mathcal{R}_j$. We can consolidate these example high-level actions into a dataset for inference of high-level function values. We will also refer to the primitive subplan p_{ij}^s as an *instance* of a high-level action[6] $\mathbf{a}_{ij}$.

Once instances p_{ij}^s of high-level actions $\mathbf{a}_{ij}$ are identified, we can calculate the navigation properties of the instances using the known primitive cost and traversability functions, $c(a)$ and $f(a)$. We calculate the cost $c(p_{ij}^s)$ of instance p_{ij}^s by summing the primitive cost function over all primitive a in example action p_{ij}^s,

$$c(p_{ij}^s) = \sum_{a \in p_{ij}^s} c(a). \quad (5)$$

Similarly, we can calculate the feasibility of a high-level action instance by evaluating the feasibility of each primitive action in the instance. If any of the primitive actions in the instance are infeasible, the instance cannot be executed in the environment, and is labeled as infeasible. Formally, we calculate the feasibility $f(p_{ij}^s)$ of a high-level action instance

[2]This is an extension of *cost similarity* [16] which considers plan feasibility.

[3]For simplicity of implementation, we assume that regions are defined by decomposing the environment into low-resolution grid cells, but we note that other decomposition functions could be used.

[4]This definition implies that $\mathbf{a}_{ij}$ is a set of sequences of primitive actions.

[5]In a mild abuse of notation, each $\mathbf{a}_{ij}$ is an unordered set of plans $\mathbf{a}_{ij} = \{p_{ij}\}$. Each p_{ij} is an ordered sequence of actions $p_{ij} = \{a_{l:m}\}$.

[6]In environments that induce regions $\mathcal{R}_i$ which are not internally fully connected, an agent may enter and exit $\mathcal{R}_i$ multiple times to access the disconnected subsets of $\mathcal{R}_i$. This can result in the agent executing the same high-level action $\mathbf{a}_{ij}$ more than once in a single optimal trajectory. In this case, we record each execution of the high-level action separately.

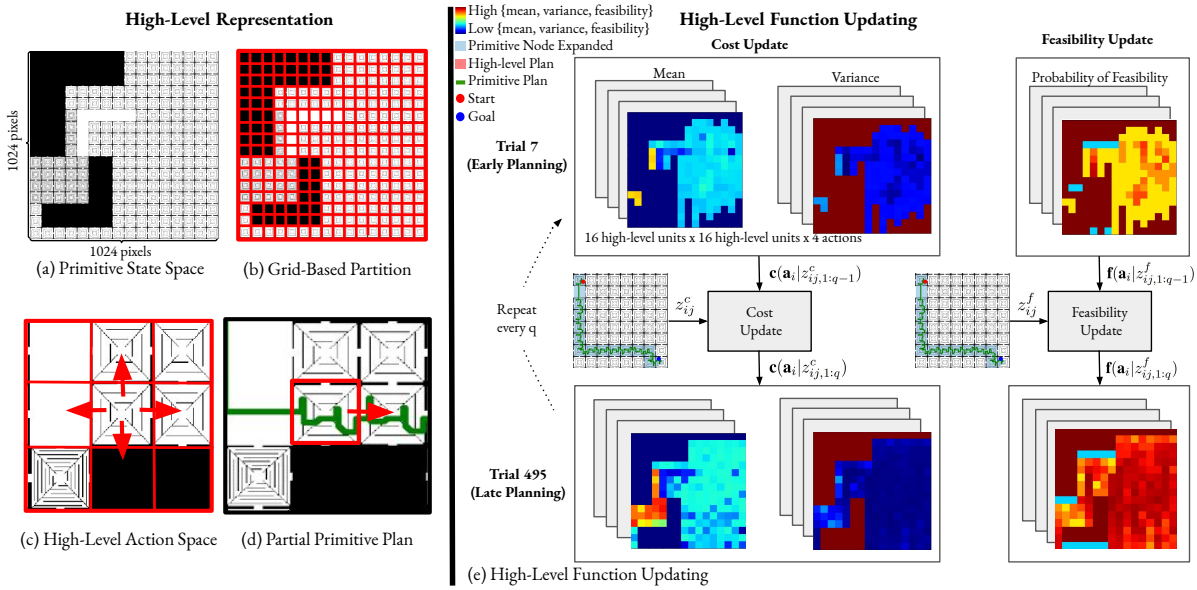

Fig. 2: **Overview of the approach.** Rather than planning at a single resolution in a large primitive state space (a), we partition primitive state spaces into high-level regions using a grid (b) and define high-level actions as transitions between high-level regions (c). Then, primitive plans induce navigation properties over the high-level action space, like the partial primitive plan in green (d), which shows the left high-level action is feasible and has a cost equal to the plan length in the region. Finally, we use online datasets of primitive plan navigation properties to estimate high-level functions over time via two recursive estimation techniques, *Averaging* (not shown) and *Bayes Filtering* (e). Results are shown for the *Medium-Risk* planner (see Section III-B).

p^s_{ij} as the minimum feasibility of the primitive actions in the instance,

$$f(p^s_{ij}) = \min_{a \in p^s_{ij}} f(a), \tag{6}$$

where we note that $f(p^s_{ij})$ is a binary value.

Using this technique, each time that a primitive refinement is generated in an environment, we generate a dataset of high-level action instances and their navigation properties,

$$\mathcal{D} = \{(p^s_{ij}, c(p^s_{ij}), f(p^s_{ij})), (p^s_{jk}, c(p^s_{jk}), f(p^s_{jk})), ...\}. \tag{7}$$

D. The High-Level Functions

While high-level action instances and their navigation properties provide information about previous plans in the environment, it is not obvious how to incorporate a dataset of example action instances and navigation properties $\mathcal{D}_{ij} = \{(p^s_{ij}, c(p^s_{ij}), f(p^s_{ij}))^0, (p^s_{ij}, c(p^s_{ij}), f(p^s_{ij}))^1, ...\} \subseteq \mathcal{D}$ for a high-level action $\mathbf{a}_{ij}$ into approximations of high-level navigation properties that improve planning performance in novel queries over time. In this work, we would like to generate compact high-level functions $\mathbf{c}(\mathbf{a}_{ij})$ and $\mathbf{f}(\mathbf{a}_{ij})$ which estimate the navigation properties of high-level actions $\mathbf{a}_{ij}$ over the course of a multi-query planning trial,

$$\mathbf{c}(\mathbf{a}_{ij} | c((p^s_{ij})_{1:q})) = g_c(\mathcal{D}_{ij}) \tag{8}$$

$$\mathbf{f}(\mathbf{a}_{ij} | f((p^s_{ij})_{1:q})) = g_f(\mathcal{D}_{ij}), \tag{9}$$

where g_c and g_f are cost- and feasibility-specific functions and q is the number of observed instances of a high-level action $\mathbf{a}_{ij}$. We recognize that we can treat the plan cost and feasibility results $c(p^s_{ij})$ and $f(p^s_{ij})$ in dataset $\mathcal{D}_{ij}$ as *streamed measurements* z^c_{ij} and z^f_{ij} of the high-level functions,

$$z^c_{ij} = c(p^s_{ij}) \tag{10}$$

$$z^f_{ij} = f(p^s_{ij}), \tag{11}$$

where for computational efficiency we assume that measurements of different high-level actions in a plan are independent (e.g., $z^c_{ij} \perp z^c_{jk}$). By treating the navigation properties of high-level action instances as streamed measurements, we can reduce the multi-query high-level function construction problem to a recursive estimation problem.

E. Online High-Level Function Updating as Recursive Estimation

The goal of the recursive estimation problem is to use streaming measurements to approximate the current value of a function. However, recursive estimation approaches often rely on a model of the data collection process to define a measurement update function, and a prior to determine an initial, uncertain belief over the function value. While agents with additional environmental information, such as data from a perception system, may have the necessary information to approximate such models, other agents with less information may not be able to use the models. In this work, we consider two approaches to recursive estimation. The first assumes no information about the data collection process or initial belief and uses online averaging to update function values. The second assumes a user can make informed guesses about the data collection process and can generate uninformative prior beliefs for the functions, and uses a Bayes filter to improve function value estimates over time.

1) Averaging: When we cannot make any assumptions about the data collection process or prior function values, we approximate high-level function values as a rolling average of the observed measurements,

$$\mathbf{c}(\mathbf{a}_i | z^c_{ij,1:q}) = \mathbf{c}(\mathbf{a}_i | z^c_{ij,1:q-1}) + \frac{z^c_{ij} - \mathbf{c}(\mathbf{a}_i | z^c_{ij,1:q-1})}{q} \tag{12}$$

$$\mathbf{f}(\mathbf{a}_i|z^f_{ij,1:q}) = \mathbf{f}(\mathbf{a}_i|z^f_{ij,1:q-1}) + \frac{z^f_{ij,q} - \mathbf{f}(\mathbf{a}_i|z^f_{ij,1:q-1})}{q} \quad (13)$$

where $\mathbf{c}(\mathbf{a}_i|z^c_{ij,1:q})$ and $\mathbf{f}(\mathbf{a}_i|z^f_{ij,1:q})$ are scalars.

2) Bayes Filtering: The averaging approach is simple, but it is unable to represent uncertainty in high-level action values. Representing uncertainty is important for high-level online estimation, as high-level representations of primitive planning problems are often lossy and initial high-level function value estimates can be inaccurate. However, accurately describing uncertainty from streaming measurements often requires additional information about the measurement process and environment. When we know or can approximate properties of the measurement process and prior beliefs over high-level navigation properties, we can explicitly represent uncertainty in high-level function values over time by maintaining a belief over each function value. Then, we can apply Bayes filtering to recover the belief update equations,

$$\Pr(\mathbf{c}(\mathbf{a}_{ij})|z^c_{1:q}) \propto \Pr(z^c_q|\mathbf{c}(\mathbf{a}_{ij}))\Pr(\mathbf{c}(\mathbf{a}_{ij})|z^c_{1:q-1}) \quad (14)$$

$$\Pr(\mathbf{f}(\mathbf{a}_{ij})|z^f_{1:q}) \propto \Pr(z^f_q|\mathbf{f}(\mathbf{a}_{ij}))\Pr(\mathbf{f}(\mathbf{a}_{ij})|z^f_{1:q-1}). \quad (15)$$

In practice, we find that simple modeling decisions can lead to uncertainty-aware high-level functions which enable robust planning. In the experiments that follow, we model the cost function belief distribution and cost measurement probability distribution as Gaussian distributions, an approach used by the planning under uncertainty community [17], [25]. We also model the feasibility function belief distribution as a Beta distribution, and the feasibility function measurement probability distribution as a Bernoulli distribution. By selecting distributions which are conjugate, we ensure the computational efficiency of the update step. Additionally, we assume that the variance σ^s of the cost measurement process for the measurement z^c can be modeled as a hand-tuned constant, $\sigma^s = \lambda$. We note that we initially assume uninformative priors for $\mathbf{c}(\mathbf{a}_{ij})$ and $\mathbf{f}(\mathbf{a}_{ij})$ (high-variance and uniform, respectively).

F. Hierarchical Planning using High-Level Functions Estimated Online

Finally, we discuss the hierarchical planner used to find solutions to the optimization in (1), guided by solutions to the optimization in (2), which are calculated using the high-level representation developed in Sections II-B to II-E. At its core, the hierarchical planner is a multi-level graph-based forward search algorithm which iterates between planning in graphs which approximate the high-level and primitive optimizations. Planning begins in the high-level graph, and nodes are expanded according to a priority queue until a goal node is expanded and a high-level plan $\mathbf{p}$ is recovered. Once $\mathbf{p}$ is found, we constrain the search space of the primitive graph to contain only the subgraph of primitive edges which could be possible refinements of the high-level plan $\mathbf{p}$ by temporarily marking all other edges as infeasible. We search in the constrained graph until a primitive plan p is found, or until all reachable nodes in the subgraph have been expanded.

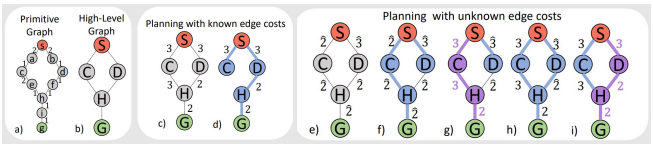

Fig. 3: **Planning in graphs with known and unknown edge costs.** The goal of the problem is to generate a path from the red start node to the green goal node using Dijkstra's algorithm. The true costs of navigating in the environment are given in the primitive graph (a), and the high-level graph (b) represents planning in the proposed high-level action space, where each region is a node and each high-level action is an edge. When the costs of high-level actions are known, edge costs are deterministic (c), and expanding nodes (blue shading) in order of cost-to-come ensures that the first expansion of the goal node is optimal (d). When the costs of high-level actions are unknown and therefore approximated (represented by the $\hat{}$ symbol, as in e), the first expansion of the goal node (f) may not be optimal after plan refinement (g, where the refined plan is shaded purple). Instead, additional paths to the goal must be expanded and refined (h-i) to ensure search optimality.

If the primitive plan is feasible and lower cost than existing primitive plans, it is stored. Then, the result of the constrained primitive planning problem is converted into the dataset in Equation 7 using the method in Section II-C, and the high-level functions are updated according to Sections II-E.1 and II-E.2. Planning returns to the high level graph, and continues until a termination condition is met. However, because the costs of high-level graph edges can change over time and may be probabilistic, some modifications to standard forward search are required for the high-level planner.

1) Queue Prioritization: For both the averaging and filtering approaches, we order nodes v on the high-level queue according to the A* evaluation function $f^\dagger$ [1],

$$f^\dagger(v) = h_{cc}(v) + h_{cg}(v) \quad (16)$$

where $v = \mathbf{e}(\mathbf{a}_{\mathbf{n}-1})$ is the terminal node of the last action $\mathbf{a}_{\mathbf{n}-1}$ in high-level subplan $\mathbf{p}$, and h_{cc} and h_{cg} are the cost-to-come and Euclidean cost-to-go of node v. However, when the properties of graph edges are probabilistic (i.e., for the filtering approach), it is not obvious how to determine the current cost of the subplan $\mathbf{p}$ which terminates in node v on the queue. We choose to approximate the cost-to-come of v as the expected cost of the subplan $\mathbf{p}$,

$$h_{cc}(v) = \mathbb{E}[\mathbf{c}(\mathbf{p})] = \sum_{\mathbf{a}_{ij} \in \mathbf{p}} \mu_{ij}, \quad (17)$$

where μ_{ij} is the mean of $\mathbf{c}(\mathbf{a}_{ij})$, subject to the constraint that the subplan feasibility is above a hand-tuned threshold γ,

$$\mathbf{f}(\mathbf{p}) = \arg\min_{\mathbf{a}_{ij} \in \mathbf{p}} \mathbf{f}(\mathbf{a}_{ij}) \geq \gamma \quad (18)$$

Plans with feasibility values less than γ are *deferred* (see Section II-F.3).

2) Termination Conditions: Because the costs and feasibilities of high-level graph-edges are uncertain, we cannot assume that we have found the lowest cost high-level plan after expanding a goal node in the high-level graph. For the averaging approach, we choose to terminate when the known cost of the current best primitive plan p is lower than the

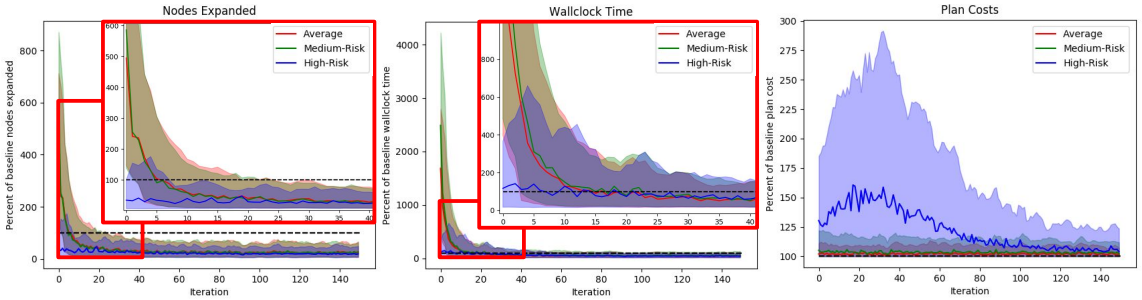

Fig. 4: **Planning metrics for the hierarchical planners over time as a percentage of *Primitive* planning metrics.** Mean values are plotted as solid lines, and the 5th-95th interquartile range, smoothed using a 4-iteration sliding window for improved visualization, is shaded. We show that all hierarchical planners resulted in increased planning efficiency over time, demonstrated by a decreased number of nodes expanded (a) and decreased wallclock time (b) as compared to the *Primitive* baseline (black dashed line). We also demonstrate that the approaches achieved planner robustness over time, demonstrated by a reduction in plan costs or a maintenance of low-cost plans over time (c). Finally, we show that the risk-based planning formulation enabled risk-aware planning during early planning; for example, the conservative *Average* planner (red) was inefficient during early trials but found low-cost plans, while the *High-Risk* planner (blue) was efficient but found high-cost plans.

A* evaluation function value of the lowest-cost node on the high-level queue, $f^{\dagger}(v)$. For the filtering approach, we use the risk-aware termination condition defined by Pearl and Kim [25], which terminates when the expected risk of terminating with a primitive plan p of cost $c(p)$ is less than a pre-defined risk threshold δ. We calculate the expected risk of terminating with a given primitive plan cost $c(p_i)$,

$$R(c(p_i)) = \int_{\tau=-\infty}^{c(p_i)} (c(p_i) - \tau) p(\tau|\mu_i, \sigma_i) d\tau, \tag{19}$$

where μ_i and σ_i are the mean and variance of $\mathbf{c}(\mathbf{a}_i)$ and we select a risk threshold as a percentage of the cost of the primitive plan, as in [25]

$$\delta = \frac{R(c(p))}{c(p)}. \tag{20}$$

3) Soft Pruning of Dominated Nodes from a Priority Queue with Uncertain Costs: In deterministic graph search, planners rely on admissible heuristics and visited lists to ensure that graph nodes are expanded exactly once, and that when a node is expanded, the optimal cost-to-come for the node is recorded. However, when the costs of nodes on a queue are uncertain, we cannot assume that the first expansion of a node is optimal (or even feasible) in terms of cost-to-come, as demonstrated in Figure 3. In uncertain environments, this implies that we must maintain all possible paths to all possible nodes in a queue to avoid pruning a path that may be optimal, but this is computationally infeasible. Instead, we introduce the notion of a *dominated* plan, or a plan which is suboptimal given the current beliefs of plan costs and feasibilities, and store dominated plans in a list of *deferred* plans, similar to Vega-Brown and Roy [12], from which plans are not considered for expansion. Then, when the high-level functions are updated, we reevaluate the navigation properties of plans on the priority queue and in the dominated list, and move plans between the queue and the list as necessary. This technique allows the priority queue to focus search on plans in the environment which are likely to lead to low cost high-level solutions, without compromising the ability to reconsider previously high-cost high-level plans, if high-level function values change.

Planner	Average	Medium-Risk	High-Risk
Nodes Expanded, Early Planning	40%	36%	**17%**
Nodes Expanded, Late Planning	19%	19%	**13%**
Total Nodes Expanded	23%	22%	**14%**
Wallclock Time, Early Planning	106%	112%	**51%**
Wallclock Time, Late Planning	37%	41%	**30%**
Total Wallclock Time	51%	55%	**34%**
Plan Cost, Early Planning	**102%**	104%	131%
Plan Cost, Late Planning	**102%**	103%	104%
Total Plan Cost	**102%**	103%	110%

TABLE I: **Planning performance for the hierarchical planners during different stages of planning.** Metrics were calculated by summing the nodes expanded, wallclock time, and plan costs for all trials in a planning stage; early planning was defined as the first 100 iterations, while late planning was defined as the remaining 400 iterations. Total metrics were the sum of plan metrics for all planning iterations. We demonstrated that the hierarchical planners were efficient and robust over time, and that the risk-aware planner could be used to elicit different balances between planner efficiency and robustness during early planning.

III. Experiments

In this section, we describe the experimental procedure used to benchmark our hierarchical representations and planners against a primitive planner running the A* algorithm in a simulated outdoor environment, and report aggregate metrics.

A. Experimental Setup

We demonstrated our approach in a simulated outdoor environment comprised of 100 1024x1024 pixel maps of randomly generated environments consisting of four terrain elements: pavement, sand, forest, and water. Each terrain element was associated with a 64x64 pixel 2D occupancy map, resulting in environments with 256 terrain elements. Terrain element occupancy maps were randomly rotated and flipped to create more diverse scenes. The environment was modeled such that all terrain elements except water were traversable, and on average, due to the geometric structure of the environment, it was least expensive to navigate in a pavement element and most expensive to navigate in a forest element. An example occupancy map of the environment is shown in Figure 1(a). We assumed a holonomic vehicle with a discrete action set: {up, down, left, right} one pixel with no

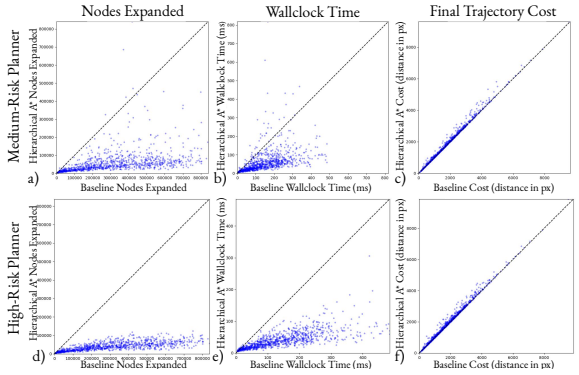

Fig. 5: **Scatter plots of steady state planning results.** Scatter plots of nodes expanded, wallclock time in milliseconds, and plan cost in simulation units for (a-c) *Medium-Risk* vs. *Primitive* and (d-f) *High-Risk* vs. *Primitive* (blue circles) for the last 10 queries in each of the 100 environments. The black dotted line is an equal value reference.

pose estimate noise. To generate the hierarchical action space, we discretized each environment into 256 equally sized, non-overlapping regions (64x64 pixels per region), resulting in a high-level action space with 1024 actions.

B. Simulation Evaluation Results

To evaluate our approach, we simulated a robot completing sequences of 500 randomly generated feasible planning tasks in each of the 100 environments using three variants of the hierarchical approach. The first variant (*Average*) used the averaging approach in Section II-E.1 to update the high-level functions, and used the averaging termination condition in Section II-F.2. The second (*Medium-Risk*) and third (*High-Risk*) variants of the approach used the Bayes filtering approach in Section II-E.2 to update the high-level functions, and used the risk-aware termination condition in Section II-F.2 with medium ($\delta = 0.5$) and high ($\delta = 1.0$) risk thresholds, respectively. All hierarchical planners used the feasibility threshold $\gamma = 0.5$, and the risk-aware planners used the constant variance $\lambda = 0.1$. If any of the hierarchical planners expanded 10,000 high-level nodes during the course of a planning trial, the planner timed out and the current lowest cost primitive plan was returned; this occurred for a single *Average* trial. Additionally, one *High-Risk* planning trial failed to find a plan; this trial was removed from aggregate metrics. We compared each variant of our approach to a non-hierarchical planner (*Primitive*) which generated optimal plans by running the A* search algorithm over the primitive, unconstrained occupancy map. We evaluated planner efficiency by comparing the total number of high-level and primitive nodes expanded and the wallclock time taken by the different planners, and we evaluated planner robustness by comparing the final plan costs of the different planners.

We demonstrated that the online hierarchical planners were capable of increasing planning efficiency while minimizing loss of plan robustness for multi-query planning problems. Planning performance metrics for the hierarchical planners are compared over time in Figure 4 as percentages of *Primitive* performance metrics[7]. In Table I, we compare the total number of nodes expanded, the total wallclock time, and the total plan cost accumulated by the planners during all trials. Across all planning trials, planning with the online representations and hierarchical planners resulted in a 77-86% decrease in the total number of nodes expanded and a 45-66% decrease in the total wallclock time as compared to the *Primitive* baseline while incurring only a 2-10% increase in plan costs.

We also showed that the risk-based formulation enabled the planner to balance between planning efficiency and robustness in a principled manner during early planning, which we defined as the first 100 planning iterations. We computed planning metrics for early planning only; results are shown in Table I. As expected, the *Average* planner demonstrated the most conservative behavior of the planners, while the *High-Risk* planner demonstrated the most risky behavior. For example, the *High-Risk* planner was most efficient, and expanded 17% of the nodes and took 51% of the time as the *Primitive* planner during early planning, but found plans which were 31% more expensive. Conversely, the average planner was the most robust of the hierarchical planners during early planning, and found plans which were only 2% more expensive than the optimal *Primitive* planner; however, *Average* was less efficient, expanding 40% of the nodes and taking 106% of the time as compared to *Primitive* during early planning. The *Medium-Risk* planner achieved an intermediate risk profile as compared to the other high-level planners, and expanded 36% of the nodes as compared to *Primitive*, while incurring 104% of the plan cost. However, due to the cost of maintaining and using the risk-based representation in a risk-averse way, *Medium-Risk* was slower than the other planners, finding plans in 112% of the time as *Primitive*.

We also analyzed the late planning performance of the high-level planners, which we defined as planner performance for the last 400 queries of a 500 query trial. Performance metrics are given in Table I; Figure 5 includes example scatter plots comparing nodes expanded, wallclock time, and plan costs for *Medium-Risk* and *High-Risk* as compared to *Primitive* for the last 10 queries in each of the 100 planning environments. We demonstrated that the hierarchical approaches expanded 13-19% of the nodes and executed in 30-41% of the wallclock time as compared to the *Primitive* baseline while incurring only a 2-4% increase in plan cost during late planning. These results indicate that the online representations and hierarchical planners are capable of significantly improving planning efficiency during late planning as compared to the baseline while minimizing loss of planner robustness, as demonstrated by the minimal increase in plan cost.

Finally, we directly evaluated the plan similarity of high-level and primitive plans over time to analyze the ability of the high-level representation to accurately represent primitive navigation properties. When a high-level plan was refined using the primitive planner, we compared the predicted navigation properties of each high-level action $\mathbf{a}_{ij}$ to the

[7]The planners approach steady state around the 150th trial in the test environments; additional trials are excluded for visual clarity.

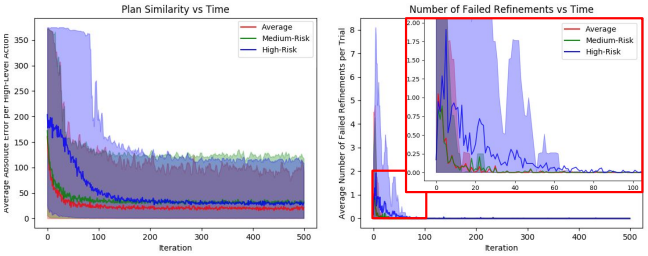

Fig. 6: **Plan similarity between high-level and primitive plans over time.** Average absolute error between predicted high-level plan costs and primitive plan costs and the average number of failed refinements over time for the three hierarchical planners. Mean values are plotted as solid lines, and the 5th-95th interquartile range, smoothed using a 4-trial sliding window, is shaded. Average absolute error and the average number of failed refinements decreased over time for all planners, indicating that the online planners were capable of learning useful high-level representations over time; *High-Risk* converged least quickly, as it prioritized efficient termination over exploration in the environment.

measured navigation properties p_{ij}^s of the resulting example primitive plan. We compared the average absolute error per high-level action and the number of incorrect feasibility assignments for the different planners in Figure 6. All planners demonstrated a decrease in average absolute cost error and number of incorrect feasibility assignments over time. Additionally, we noted that high-level function error decreased most slowly for the *High-Risk* planner, which was expected given that *High-Risk* prioritizes efficient planning over planner robustness during early planning. Overall, we demonstrated that the high-level representations were able to use online planning results to increase plan similarity over multi-query planning trials.

IV. Conclusion

We have presented a novel method for improving hierarchical planning efficiency over a multi-query planning trial by estimating properties of high-level representations online using only previous primitive planning computations. We demonstrated that using the estimated functions increased hierarchical planning efficiency over time as compared to a primitive planner running the A* search algorithm when navigating in a simulated outdoor environment, while only slightly increasing average plan cost. We introduced average-based and risk-aware versions of our approach, and showed that the risk-aware estimated functions allowed an agent to trade off between planning efficiency and robustness during multi-query planning trials.

References

[1] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[2] D. Harabor and A. Grastien, "Online graph pruning for pathfinding on grid maps," in *Proc. AAAI 2019*.

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

[4] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[5] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *IJRR*, vol. 30, no. 7, pp. 846–894, 2011.

[6] D. Hsu, Tingting Jiang, J. Reif, and Zheng Sun, "The bridge test for sampling narrow passages with probabilistic roadmap planners," in *Proc. ICRA 2003*.

[7] B. Ichter, J. Harrison, and M. Pavone, "Learning sampling distributions for robot motion planning," in *Proc. ICRA 2018*.

[8] K. Liu, M. Stadler, and N. Roy, "Learned sampling distributions for efficient planning in hybrid geometric and object-level representations," in *Proc. ICRA 2020*.

[9] D. Molina, K. Kumar, and S. Srivastava, "Learn and link: Learning critical regions for efficient planning," in *Proc. ICRA 2020*.

[10] D. Berenson, P. Abbeel, and K. Goldberg, "A robot path planning framework that learns from experience," in *Proc. ICRA 2012*.

[11] M. Phillips, B. J. Cohen, S. Chitta, and M. Likhachev, "E-graphs: Bootstrapping planning with experience graphs.," in *Robotics: Science and Systems*, vol. 5, 2012, p. 110.

[12] W. Vega-Brown and N. Roy, "Admissible abstractions for near-optimal task and motion planning," in *27th International Joint Conference on Artificial Intelligence*, 2018.

[13] P. Tsiotras, D. Jung, and E. Bakolas, "Multiresolution hierarchical path-planning for small uavs using wavelet decompositions," *Journal of Intelligent & Robotic Systems*, vol. 66, no. 4, pp. 505–522, 2012.

[14] D. T. Larsson, D. Maity, and P. Tsiotras, "Q-tree search: An information-theoretic approach toward hierarchical abstractions for agents with computational limitations," *IEEE Transactions on Robotics*, vol. 36, no. 6, pp. 1669–1685, 2020.

[15] M. Everett, J. Miller, and J. P. How, "Planning beyond the sensing horizon using a learned context," in *Proc. IROS 2019*.

[16] T. Klamt and S. Behnke, "Towards learning abstract representations for locomotion planning in high-dimensional state spaces," in *Proc. ICRA 2019*.

[17] L. Murphy and P. Newman, "Risky planning on probabilistic costmaps for path planning in outdoor environments," *IEEE Transactions on Robotics*, vol. 29, no. 2, pp. 445–457, 2013.

[18] J. J. Chung, A. J. Smith, R. Skeele, and G. A. Hollinger, "Risk-aware graph search with dynamic edge cost discovery," *The International Journal of Robotics Research*, vol. 38, no. 2-3, pp. 182–195, 2019.

[19] A. Bai, S. Srivastava, and S. J. Russell, "Markovian state and action abstractions for MDPs via hierarchical MCTS.," in *25th International Joint Conference on Artificial Intelligence*, 2016.

[20] C. Thorpe and L. Matthies, "Path relaxation: Path planning for a mobile robot," in *OCEANS 1984*, 1984, pp. 576–581.

[21] R. Bohlin, "Path planning in practice; lazy evaluation on a multi-resolution grid," in *Proc. IROS 2001*.

[22] S. Behnke, "Local multiresolution path planning," in *Robot Soccer World Cup*, Springer, 2003, pp. 332–343.

[23] A. Botea, M. Müller, and J. Schaeffer, "Near optimal hierarchical path-finding.," *J. Game Dev.*, vol. 1, no. 1, pp. 1–30, 2004.

[24] T. Klamt and S. Behnke, "Planning hybrid driving-stepping locomotion on multiple levels of abstraction," in *Proc. ICRA 2018*.

[25] J. Pearl and J. H. Kim, "Studies in semi-admissible heuristics," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-4, no. 4, pp. 392–399, 1982.