Learning Motion Planning Policies in Uncertain Environments through Repeated Task Executions

Florence Tsang, Ryan A. Macdonald, and Stephen L. Smith

Abstract— **The ability to navigate uncertain environments from a start to a goal location is a necessity in many applications. While there are many reactive algorithms for online re-planning, there has not been much investigation in leveraging past executions of the same navigation task to improve future executions. In this work, we first formalize this problem by introducing the Learned Reactive Planning Problem (LRPP). Second, we propose a method to capture these past executions and from that determine a motion policy to handle obstacles that the robot has seen before. Third, we show from our experiments that using this policy can significantly reduce the execution cost over just using reactive algorithms.**

I. INTRODUCTION

In settings ranging from warehouses to restaurants, many tasks, such as pickup and delivery and material transport are highly repetitive and ripe for automation using robots. However, the environment typically contains uncertainty, which poses difficulties for repeated start-to-goal task executions. During one instance of a task, a robot can generate a map of the environment (e.g., using SLAM [1], [2]) to navigate and complete the task. However, the environmental uncertainty makes it difficult to harness this map to complete a future task; resulting in the robot re-mapping during each task execution, or using only the most recent map along with heuristics to navigate around unexpected obstacles. In [3], the authors proposed an alternative solution to this problem, calling it the Reactive Planning Problem (RPP). The idea was to generate a motion policy, that balanced the competing tasks of identifying which environment configuration (or map) the robot was operating in, and efficiently navigating to the goal. However, their solution required the robot to be given the full set of possible configurations of the environment, and their relative likelihoods *a priori*. In practice this information is difficult to obtain and likely to be inaccurate. As a result, their proposed approach lacked robustness in that it cannot adapt to new or unexpected environments. This work builds on the RPP and proposes a new solution in which only one initial map is required *a priori*, and instead the robot learns about the environment configurations and incrementally builds a motion policy through repeated executions of a start-to-goal task.

This research is partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

The authors are with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo ON, N2L 3G1 Canada (f4tsang@uwaterloo.ca; r25macdo@uwaterloo.ca; stephen.smith@uwaterloo.ca)

Related Work: Traveling between positions or configurations is a fundamental problem in mobile robotics. The typical approach is to encode the environment into a map before running a planning algorithm to generate trajectories. To this end, there is a plethora of mapping algorithms for different applications, ranging from complex 3D surroundings [4] to highly dynamic environments [5]. Given a map along with robot dynamics, the task of selecting desirable robot actions *prior* to task execution can be a computationally complex task [6]. To address this, [7] considers obstacle correlations only between neighboring regions dependent on the direction from which the robot enters. The computational burden is further reduced by allowing the robot to re-plan during execution as its map changes. Algorithms like D* Lite [8] and lifelong planning A* [9] provide fast re-planning in order to approach real-time reaction to obstacles. In this work, the mapping objective is to capture only regions of the environment critical to task completion. We discuss conditions to encourage the robot to map only regions that may benefit future tasks.

The topic of reinforcement learning in robotics, reviewed in [10], presents a method to iteratively improve performance of difficult tasks. Q-Learning has been used to solve similar reinforcement learning problems [11], [12]. More recent work has combined these concepts with deep learning called deep reinforcement learning [13]–[15]. For example, [16] uses a deep reinforcement learning strategy to improve the exploration of office buildings. In contrast, our work does not require extensive training data.

Our work is focused on minimizing the total cost for a given number of repetitions of a task (or episodes). For this problem, there is an explicit reward/cost for an action in the current task, but there is also an implicit reward/cost for an action in the current task that will influence future tasks. This is further complicated as the interaction between the current and future tasks may become less important as the robot approaches the final task. Several works discuss the inverse reinforcement learning problem (IRL), which builds a model of the reward function [17]–[19]. Much of this work requires expert examples to learn the underlying reward function [20]. For our work, this is unavailable to the robot. Instead, we focus on predicting the implicit reward of an action on future tasks.

Contributions: The contributions of this paper are three fold. First, we introduce the Learned Reactive Planning Problem. Second, we present an algorithm that condenses past experiences into what we call *super maps* in order to

generate and update a motion policy between tasks. Lastly, we present simulation results showing the strengths and weaknesses of using this approach.

II. NOTATION

A weighted directed graph G is defined by the pair $G = (\mathcal{V}, \mathcal{E})$ with the cost $c : \mathcal{E} \to \mathbb{R}_{\geq 0}$ for traversing each edge $e \in \mathcal{E}$. A path P in a graph is defined by a sequence of vertices $v_1, \ldots, v_k$ that satisfies $(v_i, v_{i+1}) \in \mathcal{E}$ for all $i \in \mathbb{N}_{k-1}$ with cost of traversal defined by $c(P) = \sum_{i=1}^{k-1} c(v_i, v_{i+1})$. With some abuse of notation for $v_i, v_j \in \mathcal{V}$, we let $c(v, u)$ denote the minimum cost of a path from v to u. Given a graph $G = (\mathcal{V}, \mathcal{E})$, the subgraph $G_E = (V, E)$ is induced by $E \subseteq \mathcal{E}$ with $V \subseteq \mathcal{V}$ given by the endpoints of E. An edge $e = (v, u) \in E$ is said to be *incident* with vertices v and u. As the graph is directed, e is outgoing at v and incoming at u. Therefore, e is *incident-in* to u and is *incident-out* to v with the set of edges *incident-out* to v, $I_v \subseteq \mathcal{E}$.

III. PROBLEM SETUP

Consider a single robot that must repeatedly navigate from a start to a goal location in a partially known static environment. It must perform this task $T \geq 1$ times, and obstacles may be added or removed from the environment in between tasks. Our goal is to minimize the total cost to complete these T tasks. In the following subsections we define the environment, robot model, and its motion policy before formalizing the problem.

A. Environment Model

Given a graph G, there are $r = 2^{|\mathcal{E}|}$ edge subsets of $\mathcal{E}$. The robot functions within a graph drawn from the set of subgraphs of G labelled $\mathcal{G} = \{G_1, \ldots, G_r\}$, with a probability mass function (pmf) capturing the likelihood a given graph will be drawn. Contrary to the RPP [3], the robot does not know the pmf over $\mathcal{G}$. The robot experiences a sequence of T random graphs $G_{X_1}, \ldots, G_{X_T}$, where $X_1, \ldots, X_T$ are independent and identically distributed (i.i.d) random variables according to the pmf over $\mathbb{N}_r$ (i.e., $\mathbb{P}(X_t = i) = p_i$ for $i \in \mathbb{N}_r$ and $t \in \mathbb{N}_T$ where $p_1, \ldots, p_r$ is the pmf). We drop the index when referring to the underlying pmf and use random variable X. The robot executes task t in the realization G_{x_t} of G_{X_t} without knowing G_{x_t}.

We are interested in applications where a small (cardinality much less than r) subset of $\mathcal{G}$ dominates the pmf. Thus graphs in this subset are much more likely to be drawn. For cases where each graph is equally likely, namely $\mathbb{P}(X = i) = \frac{1}{r}$ for any $i \in \mathbb{N}_r$, our approach will operate similarly to existing online reactive algorithms. From a practical point of view, we are interested in structured environments (even though that structure is unknown at first), and for which that structure has occasional unexpected modifications. This captures environments where certain areas are often blocked or unblocked (e.g., a doorway) but others are expected to be in a given state (e.g., it is unlikely a wall will suddenly be absent). Our work still reacts to the unexpected case, but we wish to speed up reaction time for the most probable cases.

B. Robot Model

Suppose for some task t the robot functions within the realization $G_{x_t} = (V, E_{x_t})$. If the robot occupies $v \in V$, it may sense an edge $(v, u) \in \mathcal{E}$ to check if it is blocked and thus not traversable. Formally, the sensing action is defined by the mapping $\gamma_v : I_v \to \{\text{blocked}, \text{unblocked}\}$ where $\gamma_v(e) = \text{unblocked}$ for $e \in E_{x_t}$ and $\gamma_v(e) = \text{blocked}$ otherwise, this is the edge's state. If the robot, positioned at $v \in V$, wishes to traverse $e = (v, u) \in \mathcal{E}$, it first senses the edge e. If $\gamma_v(e) = \text{unblocked}$, then the robot will proceed to traverse e and arrive at u, incurring the transition cost $c(e)$. For simplicity, we assume there is no cost to sense the state of e and that the robot is capable of sensing whether e is blocked or not. Although we assume no sensing cost, it can be added without significant changes to the problem or solution.

After the robot performs n actions within the environment, let $E_{t,n} \subseteq \mathcal{E}$ denote the set of edges for which the robot knows the state. We define the robot's understanding, or *map*, of G_{x_t}, after its nth action, as the tuple $M_{t,n} = (E^{\mathsf{b}}_{t,n}, E^{\mathsf{u}}_{t,n})$ for known blocked edges $E^{\mathsf{b}}_{t,n} = \{e \in E_{t,n} | e \notin E_{x_t}\}$ and known unblocked $E^{\mathsf{u}}_{t,n} = \{e \in E_{t,n} | e \in E_{x_t}\}$. Note that $E^{\mathsf{b}}_{t,n}$ and $E^{\mathsf{u}}_{t,n}$ form a partition of $E_{t,n}$. When the task is finished, the robot stores the *map* in the list $\mathcal{M}_t = [M_1, \ldots, M_t]$ for $t \in \mathbb{N}_T$, where n is removed to indicate the task is completed.

C. Complete Policy

Consider a single task $t \in \mathbb{N}_T$, and to reduce notational complexity in what follows, we drop the index t. The robot state space is defined as $\mathcal{V} \times 2^{\mathcal{E}} \times 2^{\mathcal{E}}$ where $v \in \mathcal{V}$ is the robot's position, $E^{\mathsf{b}} \in 2^{\mathcal{E}}$ is the set of known blocked edges and $E^{\mathsf{u}} \in 2^{\mathcal{E}}$ is the set of known unblocked edges. At a given state $(v, E^{\mathsf{b}}, E^{\mathsf{u}})$, the robot selects an action defined by an outgoing edge $e \in I_v$, and a command from the set $\mathcal{C}$. The most primitive commands being $\{\text{move}, \text{terminate}\}$, but in later sections we add an observe and call reactive algorithm command. Formally, a policy maps the robot state space to the set of actions, $\pi : \mathcal{V} \times 2^{\mathcal{E}} \times 2^{\mathcal{E}} \to I_{\mathcal{V}} \times \mathcal{C}$. The move command updates E^{u} if $e \in E_x$ and modifies v since the robot has moved or it only updates the set of blocked edges E^{b} if $e \notin E_x$. The terminate command ends task execution and should only be used when the robot is in a *terminal state*. Given a start and a goal $v_s, v_g \in \mathcal{V}$, a state $(v, E^{\mathsf{b}}, E^{\mathsf{u}})$ is said to be *terminal* if $v = v_g$ or the graph $\underline{\mathrm{G}} = (\mathcal{V}, \mathcal{E} \setminus E^{\mathsf{b}})$ has no path from v_s to v_g. We now define a complete policy.

Definition III.1 (Complete Policy)**.** *A policy π is complete for a graph G if it produces a finite sequence of actions that ends in a* terminal state *for any subgraph in the set $\mathcal{G} = \{G_1, \ldots, G_r\}$.*

Given a graph G_j with $j \in \mathbb{N}_r$, consider the sequence of actions $A_{G_j} = a_1, \ldots, a_z$ produced by π for some $z \in \mathbb{N}$.

Each action a has a cost, which is the sum of all edges travelled during the action; we denote this set of edges as E_a. The total cost of A_{G_j} would be given by $\text{cost}(A_{G_j}) = \sum_{i=1}^{z} \sum_{e \in E_a} c(e)$. Therefore, the expected cost to complete a task is

$$\mathbb{E}_X[\text{cost}(\pi)] = \sum_{j \in \mathbb{N}_r} \mathbb{P}(X = j)\text{cost}(A_{G_j}) \,. \tag{1}$$

For a complete policy to exist it is sufficient that the component containing v_s is strongly connected for each graph in $\mathcal{G}$ that has non-zero probability. This holds as long as the robot can exit each region that it can enter.

D. Learned Reactive Planning Problem (LRPP)

We consider a sequence of T tasks where the robot wishes to minimize the summed cost of completing each task. When considering a sequence of T tasks, note that all prior tasks affect the way in which the robot completes the current task. Therefore, the policy for task t may use the information collected during all prior tasks. Formally, we define this as the Learned Reactive Planning Problem.

Problem 1 (Learned Reactive Planning Problem (LRPP))**.** *Given a graph G with unknown pmf over all subgraphs $\mathcal{G}$, a start and goal $v_s, v_g \in \mathcal{V}$ and number of tasks T, find a sequence of T complete policies, $\pi_1, \ldots, \pi_T$, that minimizes $\sum_{t=1}^{T} \mathbb{E}_{X_t}(\text{cost}(\pi_t))$, where π_t may depend on the observations made in tasks $1, \ldots, t-1$.*

We now characterize the complexity of this problem for the special case when the pmf over subgraphs is completely known, which occurs as $T \to \infty$.

Proposition 1. *Even if the pmf over subgraphs $\mathcal{G}$ is known, the Learned Reactive Planning Problem is PSPACE-hard.*

Proof. Consider an instance of the stochastic Canadian Travelers problem (CTP). This consists of a graph $G_{\text{CTP}} = (V, E)$, a cost on each edge $c_{\text{CTP}} : E \to \mathbb{R}_{>0}$, and a probability for each edge $p : E \to [0, 1]$, giving the probability $p(e)$ that the edge $e \in E$ is unblocked. The goal is to find a policy that minimizes the expected cost from start to goal. To reduce this problem to LRPP with a known pmf, we create the following instance of the LRPP: We set $G = G_{CTP}$, $c = c_{\text{CTP}}$, and for each subgraph $G_i = (V, E_i) \in \mathcal{G}$, we define its probability $\mathbb{P}(X_t = i) = p_i$ as

$$p_i = \prod_{e \in E_i} p(e) \prod_{e \in E \setminus E_i} \big(1 - p(e)\big).$$

An optimal policy for this instance of LRPP then minimizes the expected cost of completing a task. This policy then is also optimal for the CTP. Since the CTP is PSPACE-hard [21], the LRPP with a known pmf is also PSPACE-hard. □

Remark 1. *Notice that in the first task $t = 1$, the pmf is completely unknown, and thus minimizing the expected cost with an unknown distribution is equivalent to minimizing the worst-case cost. Thus, the first task is an instance of the non-stochastic version of the Canadian Travelers problem [22]. In this problem, the goal is to compute a policy that minimizes the competitive ratio, defined as the worst-case ratio over all subgraphs between the cost to navigate from start to goal using the policy, and the cost of the optimal path from start to goal in the subgraph. This problem is also known to be PSPACE-complete [22].*

IV. Solution Approach

There are three key challenges to address when considering the approach in [3] to solve the LRPP. First, the subgraph set $\mathcal{G}$ and its pmf is unavailable to the robot for planning. We propose a Map Memory Filter in Section IV-A to estimate $\mathcal{G}$ and its pmf by efficiently storing the robot's map $M_{t,n}$ for every task t into $\mathcal{M}_t$. Second, we need a method for the robot to reach the goal when it encounters an environment it has not experienced before. In Section IV-B we introduce the idea of calling a *reactive algorithm* to handle such environments, and adding this command to the set C that can be used in a policy. Third, the robot needs to be able to update its navigation strategy from v_s to v_g as its estimate of $\mathcal{G}$ and its pmf changes between task executions. In Section IV-E we explain how the policy generating algorithm proposed by [3] can be utilized to generate and update a complete policy π that can react to all the realizations it has experienced before. This approach also introduces the command *observe* to set C.

A. Map Memory Filter

After the nth action during task t, the robot's knowledge is defined by the tuple $(\mathcal{R}_{t,n}, \mathcal{M}_{t-1})$ where $\mathcal{R}_{t,n} = (v_{t,n}, E^{\text{b}}_{t,n}, E^{\text{u}}_{t,n})$ is the robot state after the nth action. The robot would only need to store map $M_{t,n} = (E^{\text{b}}_{t,n}, E^{\text{u}}_{t,n})$ if it did not *agree* with a map stored from a previous task. This is known as map agreement.

Definition IV.1 (Map Agreement)**.** *Given maps M_1 and M_2, we say M_2 agrees with M_1 if $E^b_2 \cap E^u_1 = \emptyset$ and $E^u_2 \cap E^b_1 = \emptyset$.*

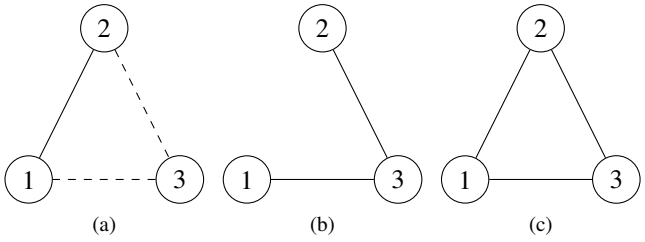


Fig. 1: Map examples

Consider the graph examples in Fig. 1 where the solid lines are unblocked edges, dashed lines are unknown edges, and the lack of a line indicates a blocked edge. Map (a) agrees with map (c), but not map (b) because edge (1,2) is missing. Note that the robot does not need to know all of the environment to accomplish its task and can leave regions unmapped, which may result in different realizations that seem identical to the robot.

Thus, the robot only needs to keep track of one map which we call a *super map*, formally defined below.

Definition IV.2 (Super Maps)**.** *A map M_j with $j \in \mathbb{N}_t$ is a super map if all M_i for $j \neq i \in \mathbb{N}_t$ that agree with M_j satisfy $E_i^b \subseteq E_j^b$ and $E_i^u \subseteq E_j^u$.*

Then, to reduce storage and search space, we redefine $\mathcal{M}_t$ as the set of *super maps* at the end of task t. The problem of computing a minimal set of supermaps can be formalized as follows.

Problem 2 (Map Merging Problem)**.** *Given a set of collected maps from each task, $\mathcal{M}_T = [M_1, M_2, \ldots, M_T]$, find a minimum partition of $\mathcal{M}_T$ such that every map in each subset agree with each other.*

Note that merging the maps in a subset forms a super map, and thus the solution to the Map Merging Problem provides a compressed representation of the robot's past experiences. We can show that the Map Merging Problem is NP-Hard through a reduction from the minimum clique cover (MCC) problem to the Map Merging Problem. Consider an instance of the MCC problem: Given a graph G, find the minimum number of cliques to cover every vertex. Given an instance of map merging, let each map M_t be a vertex, and let there be an edge between every two maps that agree with each other. This graph is called an *agreement* graph. Set G to be the agreement graph, and then the MCC of G provides a minimal partition of the maps in $\mathcal{M}_T$.

The agreement graph is built over time as the robot completes each task, so map merging is a form of online MCC. The MCC of a graph is equivalent to the minimum graph coloring of the complement of the graph [23]. The map merging method proposed in Algorithm 1 is a greedy approach that immediately adds a map to a subset via merging (lines 3 and 4) if it agrees with an existing super map. It is analogous to the First Fit approach to the online graph coloring problem, which, while not an approximation algorithm [24], provides good performance in practice.

Algorithm 1: mapFilter

Input: M_t, $\mathcal{M}_{t-1}$
Output: $\mathcal{M}_t$
1 **for** each $(E_j^{\mathrm{b}}, E_j^{\mathrm{u}}) \in \mathcal{M}_{t-1}$ **do**
2 **if** $E_t^{\mathrm{u}} \subseteq E_j^{\mathrm{u}}$ AND $E_t^{\mathrm{b}} \subseteq E_j^{\mathrm{b}}$ **then**
3 return $\mathcal{M}_{t-1}$;
4 **if** $E_t^{\mathrm{u}} \cap E_j^{\mathrm{b}} = \emptyset$ AND $E_t^{\mathrm{b}} \cap E_j^{\mathrm{u}} = \emptyset$ **then**
5 Update $M_j = (E_j^{\mathrm{u}} \cup E_t^{\mathrm{u}}, E_j^{\mathrm{b}} \cup E_t^{\mathrm{b}})$;
6 return $\mathcal{M}_{t-1}$;
7 return $\mathcal{M}_{t-1} \cup M_t$;

Using this method of storage, we can simplify the expected cost estimate (1) to,

$$\mathbb{E}_X[\mathrm{cost}(\pi_t)] = \sum_{M_j \in \mathcal{M}_t} \left(\frac{n_j}{t}\right) \mathrm{cost}_{\pi_t}(M_j)\,, \qquad (2)$$

where n_j is the number of maps the robot has experienced by task t that agree with super map M_j. Thus the estimated probability of encountering M_j in the next task is $\hat{p}_{M_j} = n_j/t$. Let $\hat{P} = [\hat{p}_{M_0}, \hat{p}_{M_1}, \ldots, \hat{p}_{M_t}]$, and together with the set $\mathcal{M}_t$, forms our estimate of the pmf of G. Note that $\mathcal{M}_0 = \{(\emptyset, \mathcal{E})\}$ and initialize $n_0 = 1$, leading to the robot's initial assumption of $\hat{p}_e = 1 \quad \forall e \in \mathcal{E}$, i.e., all edges in G are unblocked.

B. Calling A Reactive Algorithm

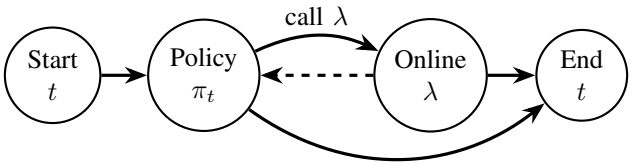


Fig. 2: Integrating a reactive algorithm λ with a policy.

Consider the composite approach displayed in Fig. 2 for executing task t. The number of possible realizations for G_{X_t} may be exponential in the number of edges; therefore, our approach is to plan paths for only a subset of environments and use a *reactive algorithm* λ for the remaining, adding a call λ command to C.

Definition IV.3 (Reactive Algorithm λ)**.** *A reactive algorithm λ computes online a sequence of move commands to lead the robot to a terminal state. Such an algorithm must guarantee that it can find a path from v_s to v_g if one exists.*

The reactive algorithm λ allows the robot to handle unexpected environments as they are encountered. Meaning the robot will always enter a terminal state (in some finite number of moves) after it calls λ. An example reactive algorithm is D* Lite [8].

For a given task t, the robot starts by following the preplanned paths in the policy π_t until either 1) an obstacle prevents the robot from continuing (in which case the robot is in a new map) or 2) all super maps that are consistent with the robots observations have no path to the goal v_g. In either case, π_t calls λ to finish the task. This satisfies the complete policy requirement, and the policy is updated each time a new task is completed. The preplanned paths are expected to be more efficient at reaching v_g than λ, and as such we wish to minimize the probability of the robot calling λ.

Remark 2. *The dashed edge in Fig. 2 is not considered within this work as returning from λ may result in a large number of states that the policy must map to actions.*

C. Policy Structure

A policy for task t can be efficiently encoded into a binary tree $\pi = (N, L)$. The nodes N of the tree are given by tuples (Y, v, e) for belief $Y = \{i \in \mathcal{M}_{t-1} | M_i \text{ agrees with } M_{t,n}\}$ at vertex $v \in \mathcal{V}$. The edge e is an *observation* at vertex v. For each node, the set L contains two paths, one to each child node, corresponding to each outcome of e. Thus,

Fig. 3: Overview of the policy update.

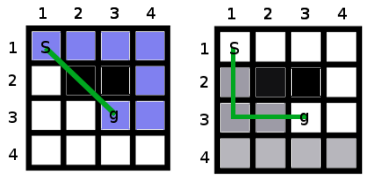
there are two outcomes, one corresponding to $e \in E_{x_t}$ and the other $e \notin E_{x_t}$. If $e = \emptyset$, then either $v = v_g$ or there is no path to goal in any of the agreeing super maps and λ must be called. To match our robot model and to facilitate understanding, we will limit $e \in I_v$. Then in this work, we can now define the full command set $C = \{move, observe, call\,\lambda, terminate\}$.

Remark 3. *From the RPP [3], the policy definition can be extended to more general sensor models. Nodes become (Y, v, O) where O is a set of edges, but the policy tree will no longer be binary. Due to space constraints we leave this extension for future work.*

D. Build Policy

The key step in our approach is the update shown in Fig. 3 that occurs between tasks and builds a policy as more tasks are completed. After completing task t, we have our estimate of the set of subgraphs $\mathcal{M}_t$ and the pmf $\hat{P}$ based on all prior experience. With some modifications, we can solve the RPP problem from [3] using these estimates as parameters, resulting in a policy π which has the structure from Section IV-C. Note that the call λ command must be explicitly added to every node that has no path to goal.

Since each super map in $\mathcal{M}_t$ is only a partial representation of a realization, it is necessary to make some assumptions to fill in missing information. If the state of an edge in super map M_j is unknown (i.e., $e \notin E^{\text{b}}$ and $e \notin E^{\text{u}}$), we assume it to be unblocked. This choice encourages the robot to explore, as it will attempt to traverse an unknown edge if it is on a shortest path.



(a) (b)

Fig. 4: (a) shows a realization for task t, and (b) is the collected M_t. The green line in (a) is the path determined by the policy π_t, in (b) by the policy π_{t+1}. The blue squares are the path that the robot actually took, the grey squares are unknown.

Consider the simple example in Fig. 4, assuming the robot only has an empty grid for M_0 in $\mathcal{M}_t$, it attempts to execute the task in (a) using the green path. However, it must call the reactive algorithm, and at the end of the task, the robot stores the map in (a) as M_1, the state of the grey squares are unknown. When building the policy, an observation for the edge ((1,1),(2,2)) will be selected, and in the case this edge is blocked, a path will be calculated from s to g in M_1 since no other maps exist. If only the partial map M_1 was available, the blue shaded path would be used in the policy. However, since we are assuming the grey squares are unblocked, the algorithm will select the green path in (b). Even if that assumption was proven wrong during task execution, it will result in more knowledge of the realization, and the next time the policy is built, the algorithm will not repeat the same path for that particular super map.

E. Policy Update

Finally, we present our entire solution in Algorithm 2, which covers task execution and policy building. In Line 1, we initialize the set of super maps $\mathcal{M}$ with $\mathcal{E}$ as a set of unblocked edges. In other words, the robot is aware of all edges that it could potentially move across. Such information could come from a floor plan of the environment, containing all permanent obstacles. The robot initially assumes that $\hat{p}_e = 1\,\forall e \in \mathcal{E}$. This assumption ensures that the reactive algorithm λ will always initially attempt the shortest possible path to v_g. In Lines 4-6, the robot executes the task by following the policy until it reaches a terminal state, updating its set of super maps and policy in Line 11 and 3 respectively, before executing the task again.

Algorithm 2: Sequential Task Completion

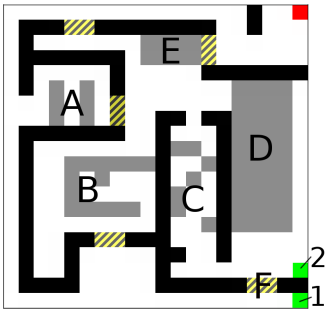
```
   Input: E,v_s,v_g
1  M_0 = [(E, ∅)];
2  for t = 1, ..., T do
3      π_t =buildPolicy(M_{t-1},v_s,v_g);
4      initialize state R_{t,n} = (v_s, ∅, ∅) for n = 0;
5      do
6          execute π_t(R_{t,n});     // if λ is called
             wait until it terminates
7          update R_{t,n};
8          increment n;
9      while R_{t,n} not terminal;
10     M_t = (E^b_{t,n}, E^u_{t,n}) from R_{t,n};
11     M_t = mapFilter(M_t,M_{t-1});
```

V. Simulation Results

In this section, we describe the simulations we conducted with Algorithm 2 and compare the results with only using a reactive algorithm. Of particular interest is the average cost of the path taken across all tasks given T. The reactive algorithm used in our simulation calls A* to replan when it encounters an unexpected obstacle.

A. Test Environment

Our tests were conducted on the environment in Fig. 5. A floor plan with the black obstacles is given to the robot. The red square is the goal and the green squares are possible starting locations. The grey and striped obstacles are unknown to the robot, and the probability of them being present in a given task t is as shown in the table in Fig. 5.



Obstacle	Probability
A	0.5
B	0.4
C	0.6
D	0.8
E	0.5
F	0.2

Fig. 5: Base map and obstacle distribution of the environment. The green and red squares are starting and ending points respectively.

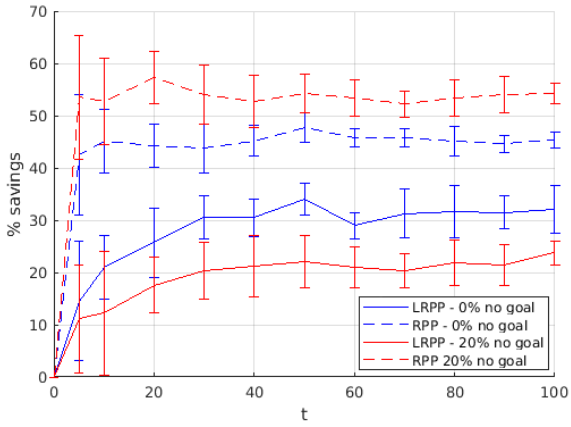
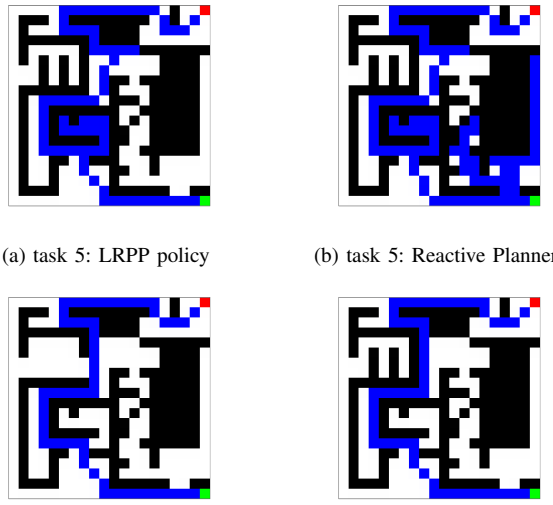


Fig. 6: Average cost savings compared to using only the reactive algorithm.

Each letter corresponds to the grey obstacles in that area. The exceptions are: A includes the closest striped obstacle to the right, and F is every striped obstacle in the environment. This map is given as a 20×20 8-direction grid, resulting in a graph with 400 vertices, 1654 edges, and 64 realizations.

B. Simulation Results

Fig. 6 shows the average cost savings over t task executions using the online policy update (LRPP) and the policy generated by using the hidden environment data (RPP) compared to running only the reactive algorithm. The data points are an average over 10 trials. The savings are generally greater than 20%, and none of the averages were greater than A*. Simulations for the blue lines were run on the map in Fig. 5, with location 1 as the start. Simulations for the red lines were run on the same map, with location 2 as the start where there is a probability of 20% for the robot to be in a realization with no possible path to the goal. The larger gap in performance between LRPP and RPP is because in LRPP, the policy must call the reactive algorithm when it thinks there is no path to goal, since it is always possible that the robot is in a new environment. On the other hand, RPP knows all possible maps, and depending on the pmf, it may be able to determine no path to goal exists without exhaustively searching the environment. Notice that in both cases, there is an overall logarithmic increase in savings as T increases. This was a surprising result for the LRPP-20% no goal trials due to how expensive the exhaustive search can be.



(a) task 5: LRPP policy (b) task 5: Reactive Planner

(c) task 8: LRPP policy (d) task 5 and 8: RPP policy

Fig. 7: Paths taken in the same realization at $t_1 = 5$ and $t_2 = 8$

Fig. 7 shows the robot navigating the same map at task t_1 and t_2, where $t_1 < t_2$, and you can see the updated policy is able to avoid unnecessary backtracking and dead ends after just a few task executions. Fig. 6 shows that these savings can be quite significant.

Since the order of the realizations encountered affects the LRPP policy, it is possible for the robot to not take a shorter route if the estimated likelihood of backtracking and its cost is too high, which is a reason why the percent savings of LRPP does not converge to RPP, even in the 0% no goal case. This is a trade-off of not having a priori knowledge of the subgraphs and their pmf.

The runtime between tasks to update the policy increases as the number of super maps stored by the algorithm increases. In the experiments, the runtime increased by a factor of five with 15 super maps collected. However, for $T = 100$, the average number of super maps was 8.3 and 16.9 for the 0% no goal and 20% no goal trials respectively, while there are 64 different realizations of the environment.

VI. CONCLUSIONS

We defined the LRPP and proved it is a PSPACE-hard problem. We then proposed a solution that combines a constant time motion policy with a reactive algorithm which is able to consistently complete a set of tasks with a lower average cost than using just the reactive algorithm. Future work includes modifying the algorithm to update the policy incrementally rather than rebuilding it at the beginning of each task, improving the current solution to the map merging problem, and to integrate different types of observations such as landmarks rather than limiting observations to only edges.

REFERENCES

[1] F. Blochliger, M. Fehr, M. Dymczyk, T. Schneider, and R. Siegwart, "Topomap: Topological mapping and navigation based on visual slam maps," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, May 2018, pp. 1–9.

[2] R. Valencia and J. Andrade-Cetto, "Active pose SLAM," in *Mapping, Planning and Exploration with Pose SLAM*. Springer, 2018, pp. 89–108.

[3] R. A. MacDonald and S. L. Smith, "Active sensing for motion planning in uncertain environments via mutual information policies," *The International Journal of Robotics Research*, pp. 1–16, 2018.

[4] A. Souza and L. M. G. Goncalves, "Occupancy-elevation grid: an alternative approach for robotic mapping and navigation," *Robotica*, vol. 34, pp. 2592–2609, 2016.

[5] N. C. Mitsou and C. S. Tzafestas, "Temporal occupancy grid for mobile robot dynamic environment mapping," in *Control & Automation, 2007. MED'07. Mediterranean Conference on*. IEEE, 2007, pp. 1–8.

[6] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[7] T. Kucner, J. Saarinen, M. Magnusson, and A. J. Lilienthal, "Conditional transition maps: Learning motion patterns in dynamic environments," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013, pp. 1196–1201.

[8] S. Koenig and M. Likhachev, "Fast replanning for navigation in unknown terrain," *IEEE Transactions on Robotics*, vol. 21, no. 3, pp. 354–363, 2005.

[9] S. Koenig, M. Likhachev, and D. Furcy, "Lifelong planning A*," *Artificial Intelligence*, vol. 155, no. 1-2, pp. 93–146, 2004.

[10] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[11] A. Konar, I. G. Chakraborty, S. J. Singh, L. C. Jain, and A. K. Nagar, "A deterministic improved q-learning for path planning of a mobile robot," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 43, no. 5, pp. 1141–1153, 2013.

[12] J.-J. Park, J.-H. Kim, and J.-B. Song, "Path planning for a robot manipulator based on probabilistic roadmap and reinforcement learning," *International Journal of Control, Automation, and Systems*, vol. 5, no. 6, pp. 674–680, 2007.

[13] T. Lei and L. Ming, "A robot exploration strategy based on Q-learning network," in *2016 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, June 2016, pp. 57–62.

[14] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 3357–3364.

[15] G. Brunner, O. Richter, Y. Wang, and R. Wattenhofer, "Teaching a Machine to Read Maps with Deep Reinforcement Learning," *32nd AAAI Conference on Artificial Intelligence*, vol. abs/1711.07479, 2017.

[16] D. Zhu, T. Li, D. Ho, C. Wang, and M. Q. Meng, "Deep Reinforcement Learning Supervised Autonomous Exploration in Office Environments," *Proceedings of the 2018 IEEE Conference on Robotics and Automation (ICRA)*, pp. 7548–7555, 2018.

[17] H. Kretzschmar, M. Spies, C. Sprunk, and W. Burgard, "Socially compliant mobile robot navigation via inverse reinforcement learning," *The International Journal of Robotics Research*, vol. 35, no. 11, pp. 1289–1307, 2016.

[18] M. Kalakrishnan, P. Pastor, L. Righetti, and S. Schaal, "Learning objective functions for manipulation," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 1331–1336.

[19] G. Neu and C. Szepesvári, "Apprenticeship learning using inverse reinforcement learning and gradient methods," in *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, 2007, pp. 295–302.

[20] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and autonomous systems*, vol. 57, no. 5, pp. 469–483, 2009.

[21] D. Fried, S. E. Shimony, A. Benbassat, and C. Wenner, "Complexity of Canadian traveler problem variants," *Theoretical Computer Science*, vol. 487, pp. 1–16, 2013.

[22] C. H. Papadimitriou and M. Yannakakis, "Shortest paths without a map," *Theoretical Computer Science*, vol. 84, no. 1, pp. 127–150, 1991.

[23] J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier, "Data reduction and exact algorithms for clique cover," *Journal of Experimental Algorithmics*, vol. 13, p. 2.2, 2009. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1412228.1412236

[24] S. Vishwanathan, "Randomized Online Graph Coloring," *Journal of Algorithms*, vol. 669, pp. 464–469, 1992.