Robotic Motion Planning in Uncertain Environments via Active Sensing

by

Ryan MacDonald

A thesis
presented to the University Of Waterloo
in fulfilment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2017

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Perception and control are at the foundation of automation, and in recent years, we have seen growth in feasible applications including self-driving cars and smart homes. As automation moves from regulated, well-monitored locations (e.g., factories) into society, uncertainty in hardware and the environment poses a safety concern. Within this thesis, we focus primarily on uncertainty in the environment and discuss models of the environment known a priori and learned as the robot functions. The robot is tasked with moving from one location or configuration to another while minimizing the expected cost of observation and motion actions. We focus on control that guides the robot to a position/configuration or identifies that it is impossible to reach the position/configuration.

We first focus on a robot creating a plan, prior to deployment, based on a known environment model. This model encodes obstacle configurations into different environmental realizations along with a probability this realization will be encountered. The robot is also provided an observation model it may use to sense the environment during the task. We show that minimizing the expected cost from start to goal within these models is NP-Hard. Therefore, we present an efficient algorithm to create a policy which can react to obstacles in real-time while maintaining safety constraints on motion. A by-product of this algorithm is a lower bound on the expected cost of an optimal policy. We compare the policy and lower bound, generated by our algorithm, against that of an optimal policy and existing research.

Our focus then shifts to remove prior information about environmental obstacles. We ask the robot to complete a finite number of start to goal tasks and show the general version of this problem is PSPACE-Hard. To reduce the complexity, we develop a method that uses an arbitrary reactionary algorithm from prior work to handle unexpected obstacles. For each new environment experienced, we incrementally update the robot's policy and show that the dependence on the reactionary algorithm is not increasing. Tests are performed on a flexible factory to demonstrate the scalability of this method.

Acknowledgements

I would like to thank my supervisor, Stephen Smith, for his guidance and attention to detail throughout my time at the University of Waterloo as a Master's student. Also, I would like to thank my examiners, Dana Kulic and Christopher Nielsen, as well as the members of Professor Smith's research team. Finally, I would like to thank Nicole Bendrich for reviewing my paper submissions.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction and Literature Review

We as humans naturally plan trajectories whether it be to pick up a coffee mug or drive to work. It is natural to brake in order to avoid a car cutting into your lane or to speed up in order to avoid getting rear-ended. The success of this response typically depends on the speed at which we apply the appropriate action. In robotics, we often wish to quickly select actions that maintain safety and minimize costs associated with a task. Within this work, we do not wish to trade-off cost with safety, but rather, we consider a task where the termination conditions allow the robot to identify when a task is unsafe to complete. That is to say, we allow the robot to terminate when it identifies that the environment does not contain a trajectory to reach the goal. We minimize the expected cost of reaching this goal or of showing the goal cannot be reached. For example, if someone locks up your coffee mug, the task of reaching your mug is no longer possible; thus, once you identify it is locked up, you should probably pour another cup.

1.1 Point-to-Point Planning in Uncertainty

In robotics and automation, trajectory planning (i.e., moving from a location or a configuration to another) is a fundamental task that aids in completing a potentially larger task. In order to increase the probability that a task is completed successfully, many practical applications must consider uncertainty. This uncertainty may capture faulty sensors (Liu et al. 2015, Mahoney et al. 2016), faulty actuators (Dydek et al. 2010, Stavrou et al. 2013) and/or environmental uncertainty (Aoude et al. 2013, Du Toit & Burdick 2012) to accurately model the robot's behaviour. Uncertainty in point-to-point planning often uses sub-trajectories computed without uncertainty to simplify the overall problem. For example, Gray et al. (2012) use motion primitives for a semi-autonomous robot in order to avoid obstacles while Majumdar & Tedrake (2013) encode trajectories into a library to ensure safety in uncertain environments. These point-to-point sub-trajectories can be selected via reaction to sensor feedback of the environment (e.g., Arslan & Koditschek (2016)). In particular, we are interested in real-time reaction to sensor readings of the environment and refer to this as active sensing. For example, if someone places a book in front of your coffee cup, you will react by changing your trajectory to avoid hitting the new obstacle and ensure you reach that cup.

Consider environmental uncertainty that models obstacle correlations. Fig. 1 is an example point-to-point (S-to-G) task where obstacle O_1 exists if and only if obstacle O_2 does not. We consider a model where the robot has access to obstacle locations, correlations and probabilities and another model where the robot must learn obstacle locations, correlations and probabilities. The robot also has a model of its sensor that defines the method for observing the state of the environment. Each sensing and motion action is assigned a

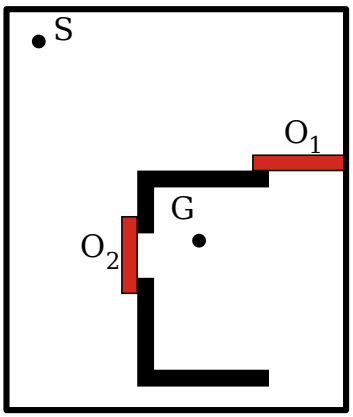

Figure 1: Example point-to-point path planning task with obstacle correlations.

unique cost, which the robot must pay to complete that action. Given a robot with sensor model and environment model (known or hidden), our problem is to minimize the expected cost of a reactive plan that moves the robot from start to goal or identifies the goal cannot be reached. We restrict reactive plans to contain a contingency plan for every environmental observation. Continuing the example in Fig. 1, if the robot knows the correlation between O_1 and O_2, its reactive plan can avoid all obstacles (seen in Fig. 2) or learn the state of the obstacles (seen in Fig. 3).

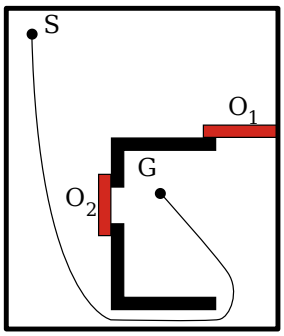

Figure 2: Example reactive plan without sensing.

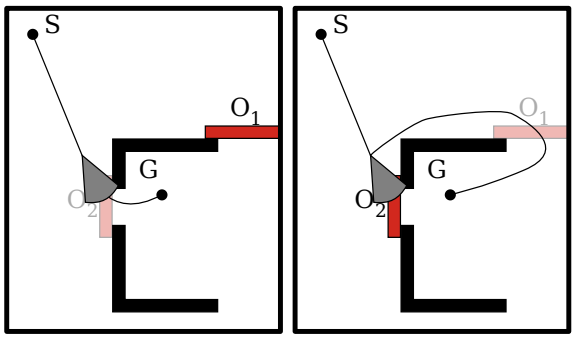

Figure 3: Example reactive plan with sensing.

Planning with Known Environment Model Given an environment with a known set of possible obstacle configurations, a probability mass function (capturing the likelihood that a configuration will be experienced by the robot), a start and a goal, the planning problem is to devise a strategy to reach the goal or determine the goal cannot be reached that minimizes the expected cost of observations and movements made by the robot. We restrict movement to areas where no obstacles exist with probability one. The robot can

make observations to determine the existence of an obstacle for a known cost. We consider observation outcome models that are correct with probability one and provide an extension to handle certain faulty sensor models.

Planning with Hidden Environment Model Given an environment with an unknown set of possible obstacle configurations, a number of tasks T, a start and a goal, the planning problem is to devise a strategy to minimize the expected movement and observation cost of T independent tasks that each require the robot reach to goal or show the goal cannot be reached. The robot has access to memory that can store what it experiences in all prior tasks in order to help in future tasks. Movement is not restricted but if the robot attempts to move through an obstacle, it pays the movement cost even though the move failed (i.e., proximity sensor while moving to avoid collision). The robot can make observations (i.e., long range sensing) to determine the existence of an obstacle for a known cost.

1.2 Literature Review

This section reviews prior research relevant to the area of path planning in uncertainty. We focus mainly on environmental uncertainty in order to reduce the problem complexity, but a large amount of this work also considers sensor and actuator uncertainty. Note that we also discuss the differences between our problem and other similar problems in the same chapter our respective problem is introduced.

1.2.1 Planning with Environment Model

In robotics, there are several effective methods for functioning within environments with known uncertainty models. Point-to-point motion is addressed by Bhattacharya et al. (2015) using persistent paths, which maximize the probability of success. However, if the computed path is obstructed, the robot ends without finishing the task (i.e., failure). To avoid failure, Partially Observable Markov Decision Processes (POMDPs) can be used to compute reactive motion policies (Bai et al. 2014, Chen et al. 2016, Kaelbling & Lozano-Pérez 2013, Van Den Berg et al. 2012). A POMDP selects actions based on partially observed states, but the computation of a POMDP policy is, in general, a PSPACE-Complete problem (Papadimitriou & Tsitsiklis 1987). In our work, we are interested in cases where the environment has a very large state space; for these cases, the POMDP's scalability becomes a barrier to use (LaValle 2006).

To avoid computational complexity, algorithms like lifelong planning A* and D* lite allow the robot to re-plan during execution (Koenig & Likhachev 2005, Koenig et al. 2004) for the case when the robot's location is fully observable. Replanning is also used by Kaelbling & Lozano-Pérez (2013) in their more general problem to form a compact policy, which is followed until the robot transitions to a state outside of the policy and triggers a

re-plan. These replanning phases often provide much needed space complexity savings. In contrast, this work targets complete policies in which all reachable robot states are contained in a compact policy, and thus, the robot does not need to re-plan during execution.

The Informative Path Planning (IPP) problem is studied in several works (Javdani et al. 2014, Lim et al. 2015, Yu, Schwager & Rus 2014), all of which provide methods for real-time reaction to information within the environment. Research in this area focuses on tasks ranging from underwater inspection (Hollinger et al. 2012) to maximizing information from start to goal (Binney & Sukhatme 2012). Similarly, active sensing (Wang et al. 2016) and active perception (Best et al. 2016) allow autonomous robot(s) to intelligently collect data based on prior observations. These works plan policies or paths prior to deployment of the robot and react to new information collected by the robot, where the robot's possible actions are known prior. In contrast, we consider cases where information may not be attainable until the robot has explored parts of the environment, which is not captured in this prior work.

In operations research, a closely related problem is planning with recourse and the Canadian Travellers Problem. Planning with recourse by Andreatta & Romeo (1988) provides possible obstacle locations, but assumes obstacle locations are such that there always exists a path to goal. The Canadian Travellers Problem, in which no prior information on obstacles is given, is a PSPACE-Complete problem (Papadimitriou & Yannakakis 1991). Remote sensing is added to the Canadian Travellers Problem by Bnaya et al. (2009) where the agent pays a sensing cost dependent on its location to determine the absence/presence of a particular obstacle. The authors look to minimize the sum of sensing and traversal costs and decide to use remote sensing based on its potential value. This decision is made for single obstacles where as in our work, the prior information available to the robot allows decisions with correlation between obstacles. A problem similar to the Canadian Travellers Problem was addressed in the area of transportation research, Issac & Campbell (2015). This work presents an integer linear program to solve their route blocked problem, in which they select a primary path and then switch to a secondary path when the primary fails. In contrast, we compute policies that minimize the expected cost for a robot to reach the goal or realize the goal is unreachable.

The work done by Polychronopoulos & Tsitsiklis (1996) introduces the problem R-SSPPR, in which the robot is operating in one of a finite number of differently weighted graphs (realizations), (i.e., each realization is defined over the same set of vertices and edges but the traversal costs differ). The robot observes outgoing edge costs at each vertex it visits and the objective is to minimize the expected cost from start and goal. They present an optimal dynamic program as well as a feedback heuristic. In contrast, our work considers different graph topologies for each realization, and a generalized sensing model along with sensing costs.

1.2.2 Learning the Environment

Consider the case where the environmental uncertainty is hidden from the robot. As the robot travels between positions or configurations, it collects fragments of its surrounding that allow it to estimate what the environment contains. Within the field of robotics, a map of the environment (e.g., occupancy grid map) allows the robot to interpret their surroundings and navigate within them (Elfes 1989). Applications require these maps to accurately encode environments ranging from complex 3D surroundings (Souza & Goncalves 2016) to highly dynamic environments (Mitsou & Tzafestas 2007).

Often, storing maps for complex or dynamic environments requires large amounts of memory, and as such, the research done by Krajnik et al. (2014) harnesses Fourier transforms to convert temporal maps into their spectral representation. Given an environment model or map along with robot dynamics, the task of selecting desirable robot actions or control, prior to task execution, can be a computationally complex task (LaValle 2006). To address this, Kucner et al. (2013) considers obstacle correlations only between neighboring regions dependent on the direction from which the robot enters. The computational burden is often further reduced by allowing the robot to re-plan during execution given its map was incorrect. Algorithms like D* Lite and lifelong planning A* provide fast replanning in order to approach real-time reaction to obstacles (Koenig & Likhachev 2005, Koenig et al. 2004), which are further discussed under the context of online reactive planning. In this work, the mapping objective is to capture regions of the environment critical to task completion. These regions are difficult to identify as their future value depends on the hidden environment model as well as the current policy. We discuss conditions that guide the robot to map regions only if these regions are estimated to benefit future task completion.

The topic of reinforcement learning in robotics, reviewed by Kober et al. (2013), presents a method to iteratively improve performance of difficult tasks. For example, Pastor et al. (2011) uses a reinforcement learning strategy to teach fine motor skills to a robot. Our work focuses on episodic (i.e., repeated robotic tasks) under a finite-time horizon. Within this area, Dann & Brunskill (2015) provides bounds on the number of episodes (tasks) to guarantee performance with high probability on episodic finite-horizon Markov Decision Processes (MDP). Q-Learning has been used to solve similar reinforcement learning problems (Konar et al. 2013, Park et al. 2007). For our cases the space required to store the Q-table is exponentially large with respect to the input, even with the work by Konar et al. (2013) to lower space required to two times the size of the state space.

Our work is focused on minimizing the total expected cost for a given number of episodes. For this problem, there is an explicit reward/cost for an action in the current episode, but there is also implicit reward/cost for an action in the current episode that influences future episodes. This is further complicated as the interaction between the current and future episodes may become less important as the robot approaches the final episode. Several works discuss the inverse reinforcement learning problem (IRL), which builds a model of

the reward function (Kalakrishnan et al. 2013, Kretzschmar et al. 2016, Neu & Szepesvári 2007). Much of this work requires expert examples to learn the underlying reward function (Argall et al. 2009). For our work, this is unavailable to the robot. Instead, we focus on predicting the implicit reward of an action on future tasks.

1.2.3 Online Reactive Algorithms

When the environment is known, point-to-point trajectories can be generated efficiently, with respect to the environment size, via algorithms like Dijkstra's Algorithm (Dijkstra 1959) or A* (Hart et al. 1968). On the other hand, when uncertainty is introduced into the environment, algorithms like Lifelong Planning A* (Koenig et al. 2004) and D* Lite (Koenig & Likhachev 2005) focus on quick replanning when an unexpected obstacle is encountered. In reasonably sized environments and/or configuration spaces, these algorithms may even appear to have real-time reaction to obstacles from the view of human onlookers (e.g. Kuwata et al. (2009)). Within the area of replanning online, applications range from reacting to randomly distributed obstacles while considering movements cost (Yu, Yang, Su & Tu 2014) to autonomous driving amongst many pedestrians (Bai et al. 2015). Often, the complexity of solving the planning problem before the robot attempts the task is intractable, and thus online reactive heuristics simplify the problem while maintaining accurate environmental and robotic models (e.g., Kaelbling & Lozano-Pérez (2013), Ross et al. (2008)).

Within this area, several approaches use learning to better guide the real-time search. For example, Bulitko & Lee (2006) combine path planning and learning in unknown environments to make more intelligent heuristic choices of next actions in real-time. This approach makes a global plan and corrects it as the environment is learned. The agent-centric approach from Koenig (2001) plans within the robot's local vicinity and updates the plan as more of the environment is learned. Koenig et al. (2003) discusses both of these approaches and presents bounds on their sub-optimality.

This work aims to extend the use of an online reactive algorithm from this area to remember what the robot has encountered and how it responded in the past in order to better react next time. Our main goal is to shift the robot's dependency from this reactive online algorithm to a control policy with constant time next action look up. We refer to this as learning to react in constant time as more tasks are executed.

1.3 Thesis Contribution

Our main contributions for planning within a known environment model are four-fold. First, we present the Reactive Planning Problem (RPP) and show it is NP-Hard. Second, we provide properties that allow for a compact representation of a RPP policy. Third, we present an efficient algorithm for a RPP sub-optimal policy that utilizes mutual information to guide exploration and uses an estimation of the cost-to-go for exploitation. Fourth, we

provide a method to bound the gap between the expected cost of our policy and that of the optimal. A prior conference version of this work appeared at WAFR2016, MacDonald & Smith (2016), and a journal version has been submitted to a special issue of the International Journal of Robotics Research.

Our contribution for planning within a hidden environment model are four-fold. First, we introduce the Learned Reactive Planning Problem and show it is PSPACE-Hard. Second, we present an environmental estimator that extracts obstacle correlations and provides a method to integrate an external reactive algorithm into a policy. Third, we present conditions where explicitly mapping regions of the environment is estimated to reduce the expected cost of future tasks. Lastly, we provide an incremental policy update that can be performed between tasks, which is shown to monotonically decrease the dependency on the external reactive algorithm.

1.4 Organization

We review background information in graph theory and related problems used in computational complexity proofs in Chapter 2. The Reactive Planning Problem (RPP) is introduced in Chapter 3. Within this chapter, we review the complexity and limitations of the RPP and provided a heuristic solution. The Learned Reactive Planning Problem is introduced in Chapter 4 as a practical extension and generalization of the RPP through a hidden environment model. This chapter discusses the computational complexity and provides a solution approach that combines a policy with an existing reactive planning algorithm. Finally, we conclude this work in Chapter 5 and discuss possible future extensions.

Chapter 2
Background

This chapter reviews concepts used throughout this thesis. Graph terminology is reviewed in Section 2.1; Dijkstra's shortest path algorithm is discussed in Section 2.2; and Mutual information is reviewed in Section 2.3. We briefly discuss the concept of complexity classes in Section 2.4. The Informative Path Planning (IPP) problem and Canadian Travellers Problem (CTP) are reviewed in Section 2.5 and Section 2.6 respectively. We use these established problems in our computational complexity proofs.

2.1 Graph Terminology

A directed graph G is defined by the pair $G = (\mathcal{V}, \mathcal{E})$ and a cost function $c : \mathcal{E} \to \mathbb{R}$. Fig. 4 shows an example directed graph with costs labelled on the edges. The set $\mathcal{V}$ is the set of vertices that are connected by the set of edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$, and $c(e)$ gives the cost of traversing an 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 $i \in \{1, \ldots, k-1\}$ 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 $v, w \in \mathcal{V}$, $c(v, w)$ refers to the minimum cost of a path from v to w. 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, and the subgraph $G_V = (V, E)$ is induced by $V \subseteq \mathcal{V}$ with $E = (V \times V) \cap \mathcal{E}$ (i.e., every edge between vertices in V).

Note a graph may also be undirected. That is to say, the edges of the graph may be traversed from v to u or u to v with the same cost when $\{v, u\} \in \mathcal{E}$. We use $\{v, u\}$ or $\{u, v\}$ to denote the edge is undirected and use (v, u) if the edge is directed. Every undirected graph can be converted into a directed graph by replacing every undirected edge $\{v, u\}$ with two directed edges of the same cost, (v, u) and (u, v). Within this thesis, when we refer to a graph, we refer to a directed graph unless otherwise indicated.

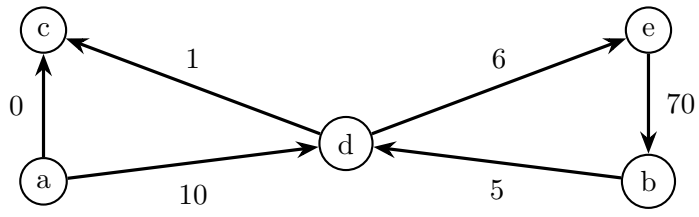

Figure 4: Example graph. Circles show vertices and arrows show directed edges (labels are costs).

An edge $e = (v, u) \in \mathcal{E}$ 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, $I_v \subseteq \mathcal{E}$. Using Fig. 4, vertex d has two incident-out edges to c and e; therefore, $I_{\mathrm{d}} = \{(\mathrm{d}, \mathrm{c}), (\mathrm{d}, \mathrm{e})\}$.

We discuss the connectivity of a directed graph G by the ability to move from one vertex to another. Formally, a directed graph is *strongly connected* if for every vertex there exists a path to every other vertex. We say a set of vertices V forms a *strongly connected component* if its subgraph G_V is strongly connected. Using Fig. 4, the full graph is not strongly connected by inspection since no vertex can reach a, but $V = \{\mathrm{b}, \mathrm{d}, \mathrm{e}\}$ forms a strongly connected component.

2.2 Dijkstra's Algorithm

Consider a graph $G = (\mathcal{V}, \mathcal{E})$ with positive cost function c, start vertex $v_s \in \mathcal{V}$ and goal vertex $v_g \in \mathcal{V}$. The shortest path problem is to find a path originating from v_s that terminates at v_g. The algorithm presented in Dijkstra (1959) computes not only the shortest path from v_s to v_g, but also the shortest path to any $v \in \mathcal{V}$. By switching the direction of all $e \in \mathcal{E}$ (i.e., $e = (v, u)$ changes to $e_{\mathrm{swap}} = (u, v)$), the same algorithm can be used to calculate the shortest path from any vertex to a single destination. We require both properties in this work and include pseudo code for this algorithm in Algorithm 1.

Algorithm 1: Dijkstra's Algorithm

Data: Graph G, cost function c, start vertex v_s
Result: distance array `dist`, previous vertex array `prev`

```
1 put V into unvisited queue Q;
2 set dist[v] = ∞ and prev[v] = N/A for all v ∈ V;
3 set dist[v_s] to 0;
4 while Q not empty do
5     remove v from Q with minimal dist[v];
6     for each neighbor u of v do
7         if dist[v] + c((v, u)) < dist[u] then
8             dist[u] = dist[v] + c((v, u));
9             prev[u] = v;
```

We use a min heap data structure for the unvisited queue Q. The graph is encoded as an adjacency list to minimize the time to collect the neighbors of v. Our problem has edge blockages and as such we add the condition that edge (v, u) is unblocked in Line 7 of Algorithm 1. This condition can be checked in constant time as the edge state is held in an array indexed by the edge itself. The maintenance of this array is formalized in the following chapters.

2.3 Mutual Information

Mutual information quantifies the amount of information one random variable contains about another random variable. Given two random variables X and Y with joint probability mass function $p(x, y)$ and marginal probability mass functions $p(x)$ and $p(y)$, mutual information $I(X;Y)$ is the relative entropy between joint distribution and product distribution $p(x)p(y)$ (Cover & Thomas 2012). Formally,

$$\begin{aligned} I(X;Y) &= \sum_{x \in X} \sum_{y \in Y} p(x,y) \log \left(\frac{p(x,y)}{p(x)p(y)} \right) \\ &= H(X) - H(X|Y) \\ &= H(Y) - H(Y|X) \end{aligned}$$

where $H(X)$ is the entropy of X and $H(X|Y)$ is the conditional entropy of X given Y. We use mutual information to quantify how useful an observation is with respect to understanding the environment.

2.4 Complexity Classes

Problems can be categorized into classes to describe the difficulty (i.e., amount of work) a digital computer has in identifying a solution (Sipser 2006). We show our planning problem within a known environment model falls into the NP-Hard class in Chapter 3, and we show our planning problem within a hidden environment model falls into the PSPACE-Hard class in Chapter 4. Given "efficient" solutions may not exist (long standing open problem), we use these proofs to validate our use of heuristic algorithms as "efficient" methods to generate sub-optimal solutions.

In order to discuss complexity, we first need to formalize the notion of a language. Let an *alphabet* be some non-empty finite set. Each element in the alphabet is called a *symbol*, and we define a *string* over the alphabet as a finite sequence of these symbols. A language is simply a set of these strings (Sipser 2006). Consider the English language as an example. It has acceptable words (strings) as finite sequences of letters (symbols) from alphabet $\{a, b, c, \ldots\}$. Let a decision problem (i.e., a problem with a yes/no answer) be given by $\mathcal{P} = (I, Y)$ where I is the language of all problem instances (as strings) and $Y \subseteq I$ are the yes instances. A verifier is an algorithm A which takes input string $s \in I$ and certificate string c and returns yes if $s \in Y$ and no otherwise.

NP: We say $\mathcal{P}$ is in the class NP (non-deterministic polynomial time) if there exists a polynomial time verifier. For A to satisfy this, it must return the correct answer within polynomial time in the length of input string $s \in I$ (i.e., $\mathbb{O}(p(\text{length}(s)))$ where p is some polynomial). For polynomial time verifiers, the number of symbols in the certificate string c is polynomial in the length of s or A cannot read them all within time limit (Sipser 2006).

PSPACE: We say $\mathcal{P}$ is in the class PSPACE (polynomial space) if there exists a solver that require no more than polynomial space. For a solver to satisfy this, it must return the correct answer by using no more than $\mathbb{O}(p(\text{length(s)}))$ space for input string $s \in I$ and some polynomial p. One can see that PSPACE contains the class NP as the time restriction on NP does not allow its verifier more than a polynomial amount of write operations (Sipser 2006).

Hardness: To establish hardness of a problem $\mathcal{B}$ within a class, we must show that $\mathcal{B}$ is at least as hard as any other problem within the class. To do this, we use polynomial time reduction which first converts any instance of problem $\mathcal{C}$, that is known to be at least as hard as any problem in the class, to an instance for $\mathcal{B}$ using a polynomial amount of work. Then, we show a solution for the instance of $\mathcal{B}$ can be converted back to a solution of the original instance of $\mathcal{C}$ using a polynomial amount of work (Sipser 2006).

2.5 Informative Path Planning

Lim et al. (2015) define the Informative Path Planning (IPP) problem under noiseless observation as a tuple (X, d, H, ρ, O, Z, r). A robot starts at r and can visit the set of sensing locations X. The cost of travel between these locations is $d(x, y)$ for $x, y \in X$. There is a finite set of hypotheses H, which has a probability mass function ρ, and a set of observations O, which are sensed with $Z(x, h, o)$ for $x \in X$, $h \in H$ and $o \in O$. The function Z returns 1 when o agrees with h and 0 otherwise. The problem then asks to minimize the expected cost of identifying the correct hypothesis. An optimal policy can be encoded as a binary tree, where nodes contain sensing information and the outgoing edges are selected via the sensing outcome. From Lim et al. (2015), IPP is NP-hard as it contains the optimal decision tree problem (Chakaravarthy et al. 2007) as a special case. We will use IPP to prove that our start to goal problem within a known environment model is NP-Hard (decision form NP-Complete).

2.6 Canadian Traveller Problem

The Canadian Travellers Problem is similar to our start to goal problem without environmental information. Formally, the robot is given an undirected graph $G = (\mathcal{V}, \mathcal{E})$, a set of uncertain edges $U \subseteq \mathcal{E}$, a start vertex s, and a goal vertex g. A realization of G is the a graph $G_H = (\mathcal{V}, H)$ for set of edges $H \subseteq \mathcal{E}$ that must contain all certain edges $(\mathcal{E} \setminus U) \subseteq H$. The uncertain edges of H, namely U, are learned for zero cost when the robot visits an endpoint of $e = \{v, u\} \in U$. The goal of the robot is to find a strategy that has the best ratio of total distance travelled to the shortest path (Papadimitriou & Yannakakis 1991). Given the robot took path P_H for realization G_H, we call this ratio the competitive ratio

for H,

$$r_{\text{comp}|H} = \frac{\text{cost}(P_H)}{\text{cost}\big(P_{\text{opt}|H}\big)}, \tag{1}$$

for optimal path $P_{\text{opt}|H}$ computed given full knowledge of G_H from the start.

Problem 2.1 (Canadian Travellers Problem (CTP))**.** Given (G, U, s, g, r) for $r \in [1, \infty)$, is there a strategy that for every realization G_H, the competitive ratio is at most r, $r_{\text{comp}|H} \leq r$?

When presenting this problem, Papadimitriou outlines it as a two player game with a searcher (our robot) and adversary, whose goal is to maximize the competitive ratio. Within this model, Papadimitriou shows CTP in PSPACE-HARD (Papadimitriou & Yannakakis 1991). We use this fact to show that our problem of start to goal without prior knowledge of the environment is also PSPACE-HARD.

Chapter 3
Planning with Known Environment Model

Robot motion planning under uncertainty is typically concerned with uncertainty in the robot's state within an environment and/or uncertainty in the outcome of a selected action on the robot's state (Binney & Sukhatme 2012, Dames et al. 2012, Hollinger et al. 2012, Javdani et al. 2014, Kaelbling & Lozano-Pérez 2013, Yu, Schwager & Rus 2014). In this work, we consider motion planning with uncertainty in the set of motion actions that a robot has access to at a given state. This problem arises in scenarios where the robot is given a set of possible locations for obstacles in an environment. The obstacles restrict the set of motions available to the robot at each point in the environment. By taking sensor measurements, the robot can narrow down the set of feasible obstacle locations and thus the motion actions it has available. We refer to this as active sensing, and our goal is to compute motion and sensing policies prior to robot deployment that enable the robot to efficiently navigate in such environments.

3.1 Introduction

In this chapter, we focus on the task of moving from a start location to a goal location or showing the goal location cannot be reached while minimizing the expected action cost. The challenge in this problem is that future costs (for obtaining information and moving between locations) are dependent on the information the robot has obtained thus far. We present conditions where exploration is no longer helpful. When these conditions are met, the robot should exploit the known motion action set to reach the goal. We also develop a policy that provides constant time lookup for the next action given the outcomes of prior observations. This allows for implementation on robots where on-board computational resources are limited at deployment, or in which high-speed motion is required.

Our work leverages the concept of mutual information within discrete environments. Mutual information is widely used to develop efficient, sub-optimal solutions for gaining information in planning (Charrow et al. 2014, Dames et al. 2016, 2012, Lim et al. 2015). Julian et al. (2014) show that by maximizing mutual information within a mapping task, the robot is eventually attracted to unexplored regions. Hollinger & Sukhatme (2014) use mutual information to generate a cost constrained path for an information collection task. Dames et al. (2012) present a mutual information gradient controller, where multiple robots search for targets and avoid hazards. We use mutual information to quantify the robot's value of a sensing action and combine it with the cost of attaining this information in order

to select the next action.

3.2 Problem Definition

We consider a single robot in a discrete environment (i.e., a robotic roadmap). The robot and environment models are defined using a weighted directed graph $G = (\mathcal{V}, \mathcal{E}, c)$ where $\mathcal{V}$ is a set of locations in the robot configuration space and $\mathcal{E}$ is the set of edges between configurations. The function c captures the costs of motion, and for each $e \in \mathcal{E}$ the value $c(e) \in \mathbb{R}_{\geq 0}$ defines the robot's cost for traversing the corresponding edge. The robot knows the vertex it occupies, but does not know which edges leaving that vertex are free to traverse (that is, which edges are obstructed by obstacles). If the robot is unsure an edge is free to traverse, it senses the edge, incurs a sensing cost and traverses it only if the outcome is unblocked. This is formalized in Section 3.2.2.

3.2.1 Environment Model

The unknown environment is one of m subgraphs of G, denoted $G_1, \ldots, G_m$, and we refer to the indices of these subgraphs as *environmental states* with *environmental state space* $\mathbb{N}_m = \{1, \ldots, m\}$. Each subgraph G_i is created by a combination of obstacles in the environment (i.e., it is induced by a subset $E_i \subseteq \mathcal{E}$ for $i \in \mathbb{N}_m$). The robot is given the set of possible obstacle locations as edge subsets $\mathcal{S} = \{E_1, \ldots, E_m\}$ along with a probability mass function (pmf) capturing the likelihood of each subgraph. We encode the probability as a random variable X that takes values from $\mathbb{N}_m$. Given a random draw x from X, the edge subset E_x induces the *realization* $G_x = (V_x, E_x, c_x)$ where $c_x(e) = c(e)$ for all $e \in E_x$; the robot must operate in G_x without knowing x.

Note that if every edge subset is possible, $m = 2^{|\mathcal{E}|}$, then the absence or presence of an edge does not imply the absence or presence of any other edges. In this paper, we focus on cases where $m \ll 2^{|\mathcal{E}|}$, and thus observing one edge allows the robot to infer the state of other edges. This is motivated in Section 3.3.2 by the space complexity required for a control policy.

Example 3.1. To illustrate the problem, consider Fig. 5 as a simplified model of a small office. We discretize the space into cells where edges between cells exist only when the two cells share a border (diagonal cells are not connected). Bars between cells represent an obstacle which implies the robot cannot traverse between these cells directly. Note cells A and B are labelled for future examples. A robot is tasked with delivering a package to cell G, and it starts from cell S. There is a large desk blocking three edges in both Environment 1 and 2, but in Environment 3 the desk now blocks off cell G. Environments 1 and 3 also have a small box next to the desk, further obstructing traversal.

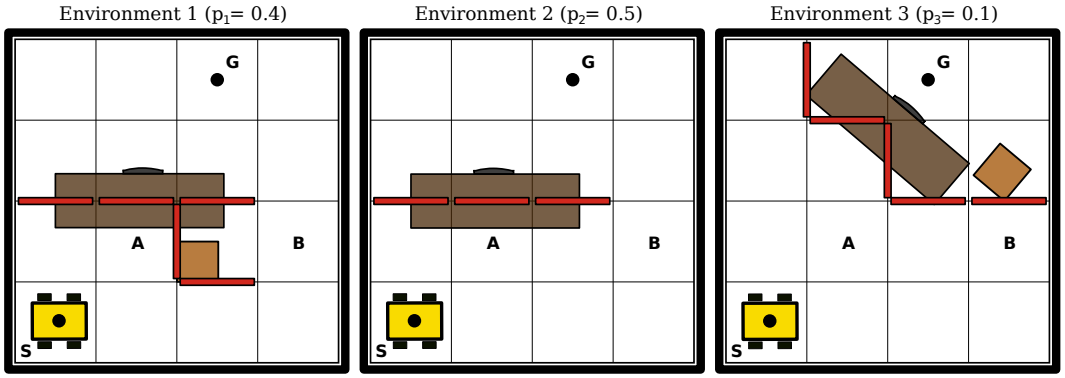

Figure 5: Office environment with three realizations and respective probability of occurrence.

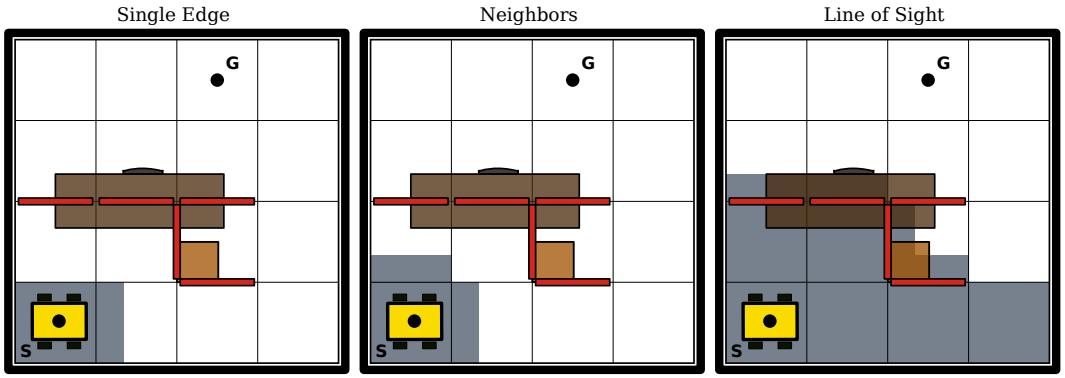

Figure 6: Example observation models: single outgoing edges (left), all outgoing edges (middle) or line of sight (right). Edges between shaded cells are sensed.

3.2.2 Robot Model

When the robot is located at vertex v, it has a finite set of *observations* $\Theta_v = \{O_1, O_2, \ldots\}$ where each $O \in \Theta_v$ is a subset of $\mathcal{E}$ (i.e., $O \subseteq \mathcal{E}$). Given an observation $O \in \Theta_v$, the function μ, with value $\mu(O) \in \mathbb{R}_{\geq 0}$, captures the cost of sensing which edges (i.e., paths between configurations) of O are free to traverse. Both Θ_v, for all $v \in V$, and μ are provided a priori, and we restrict Θ_v to satisfy $I_v \subseteq \cup_{O \in \Theta_v} O$. This ensures the robot may check if an outgoing edge from v is traversable. Some example observation models include limiting each observation to one edge (i.e., $\Theta_v = I_v$ for all $v \in \mathcal{V}$) or creating an omnidirectional sensor with one edge range (i.e., $\Theta_v = \{I_v\}$ for all $v \in \mathcal{V}$). Fig. 6 shows three possible sensor models within Environment 1 from Example 3.1.

If the robot wishes to make an observation $O \in \Theta_v$, it pays $\mu(O)$, and it is returned an *outcome* as the subset of edges in O that are free to traverse in $G_x = (V_x, E_x, c_x)$.

Definition 3.1 (Observation-Outcome)**.** Given a graph $(\mathcal{V}, \mathcal{E})$, a vertex $v \in \mathcal{V}$ and an observation $O \in \Theta_v$ with $O \subseteq \mathcal{E}$, an observation is mapped to an outcome by $O \mapsto O \cap E_x$. The robot must occupy v to attain $O \cap E_x$.

Observations with respective outcomes allow the robot to rule out environmental states. If an observation contains edge e but its outcome does not, the robot knows all edge subsets containing e cannot be correct. Similarly, if the outcome contains e, all edge subsets missing e cannot be correct.

Definition 3.2 (Consistent)**.** Given a set of observation-outcome pairs $\mathcal{O}$, an edge subset E is consistent with $\mathcal{O}$ if and only if $O \cap E = O \cap E_x$ for each $(O, O \cap E_x) \in \mathcal{O}$.

We define $Y \subseteq \mathbb{N}_m$ to be the set of environmental states *consistent* with observation-outcome pairs $\mathcal{O}$ collected by the robot. To avoid collisions with an obstacle in the environment, we impose the restriction that an edge e can be traversed only when the probability it is unblocked equals one, namely

$$\mathbb{P}(e|Y) = \frac{\sum_{i \in Y} \mathbb{P}(X = i \cap e \in E_i)}{\sum_{j \in Y} \mathbb{P}(X = j)} = 1 \, . \tag{2}$$

Note this condition only holds when $e \in E_i$ for all $i \in Y$. If $e \in \mathcal{E}$ may be blocked (i.e., $\mathbb{P}(e|Y) \in (0, 1)$), the robot can take an observation O with $e \in O$, incur observation cost $\mu(O)$ and proceed across the edge when e is in the outcome. Given a belief Y and observation O, the set of all possible outcomes is defined by $\Gamma_{(Y,O)} \equiv \{E \subseteq \mathcal{E} | E = O \cap E_i \text{ for } i \in Y\}$.

3.2.3 Policy Space

The robot state is characterized by the set of environmental states $Y \subseteq \mathbb{N}_m$ that are consistent with its observation-outcome pairs and the vertex v it occupies. Thus, the robot state space is $2^{\mathbb{N}_m} \times \mathcal{V}$. At each state (Y, v), the robot selects an action from $2^{\mathcal{E}} \times \mathcal{C}$ where $\mathcal{C} = \{\text{observe, move, terminate}\}$. The observations available to the robot are $(O, \text{observe})$ for each $O \in \Theta_v$. The move actions available to the robot are (e, move) where $e \in I_v$ and e is obstacle-free (i.e., $\mathbb{P}(e|Y) = 1$). The robot can terminate using the action $(\emptyset, \text{terminate})$. Finally, a policy maps the robot state space to the set of actions, $\pi : 2^{\mathbb{N}_m} \times \mathcal{V} \to 2^{\mathcal{E}} \times \mathcal{C}$.

Given a start and goal $v_s, v_g \in \mathcal{V}$, the environmental state space $\mathbb{N}_m$ is partitioned into $Y_{\text{goal}} = \{i \in \mathbb{N}_m \mid c(v_s, v_g) \text{ calculated on } G_i \text{ is finite}\}$ and $Y_{\text{no goal}}$ otherwise. We restrict policies to satisfy the following definition:

Definition 3.3 (Complete Policy)**.** A policy π is complete if for any realization it produces a finite sequence of actions that reach the goal (i.e., a state (Y, v_g) with $Y \subseteq Y_{\text{goal}}$) or that determine no path exists (i.e., a state (Y, v) with $Y \subseteq Y_{\text{no goal}}$).

Note: There are environments for which no *complete* policy exists. Consider Fig. 7 with a single edge observation model, $\Theta_v = I_v$. Let subgraph G_1 be the left graph with

probability of occurrence $\mathbb{P}(X = 1) = 0.5$, and let subgraph G_2 be the right graph with probability of occurrence $\mathbb{P}(X = 2) = 0.5$. The robot must move to A or B in order to identify the realization in which it resides. If the robot arrives at A and the edge to g is obstructed, then it must terminate, yet there still exists a path to goal (namely s to B to g). The same issue occurs for the other realization if the robot instead initially travels to B. The following is a sufficient condition for a complete policy to exist. Given G_i for any $i \in \mathbb{N}_m$, the connected component containing the start must be strongly connected. We formalize this with the following Lemma.

Lemma 3.1. *Given graphs $G_1, \ldots, G_m$ with start and goal $v_s, v_g \in \mathcal{V}$, if the component containing v_s is strongly connected, there always exists a complete policy.*

Proof. Let the robot assume $x = i$ for $i \in \mathbb{N}_m$ which has a path P from v_s to v_g. Let A be a sequence of sense then move actions to follow P. If any sense action from A returns blocked, the realization is not i (i.e., $x \neq i$). The robot then returns to v_s (always valid as component with v_s is strongly connected). Continue this until the robot reaches the goal or has ruled out all graphs which have a path to goal. All cases end in a terminal state. □

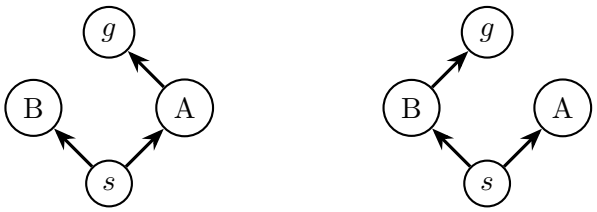

Figure 7: Pathological realizations for which a complete policy may not exist.

A policy π defines a state transition function $f : 2^{\mathbb{N}_m} \times \mathcal{V} \times 2^{\mathcal{E}} \times \mathcal{C} \to 2^{\mathbb{N}_m} \times \mathcal{V}$ where f updates Y after the observe command and updates v after the move command. Given x drawn from $\mathbb{N}_m$ according to the pmf, a policy π emits a sequence of states and actions

$$(\mathbb{N}_m, v_s), a_1, (Y_2, v_2), a_2, \ldots, (Y_z, v_z), (\emptyset, \text{terminate})$$

for some finite positive integer z. The cost of an action a is

$$\text{cost}(a) = \begin{cases} c(e) & \text{if } a = (e, \text{move}) \\ \mu(O) & \text{if } a = (O, \text{sense}) \\ 0 & \textit{otherwise} \end{cases} .$$

Given random draw x, the total cost incurred using π is given by,

$$\text{cost}(\pi | X = x) = \sum_{i=1}^{z} \text{cost}(a_i) . \tag{3}$$

Remark 3.1 (Policy Domain). Note that the domain of the policy has $n2^m$ states. In Section 3.3 we derive properties that enable a more compact representation.

3.2.4 The Reactive Planning Problem

The expected cost of a complete policy π is found by taking the expectation over the environmental states,

$$\mathbb{E}_X(\pi) = \sum_{x \in \mathbb{N}_m} \text{cost}(\pi | X = x) \mathbb{P}(X = x) \ . \tag{4}$$

Problem 3.1 (Reactive Planning Problem, RPP)**.** Given a graph G, start and goal vertices $v_s, v_g \in \mathcal{V}$, a set of edge subsets $\mathcal{S}$ with corresponding random variable X that has a known probability mass function and observations Θ_v for all $v \in \mathcal{V}$, find a complete policy π that minimizes $\mathbb{E}_X(\pi)$ over induced subgraph G_x for random draw x.

3.3 Properties and Complexity of Reactive Planning

In this section, we establish several properties of robot actions that enable us to efficiently represent complete policies along with the complexity of the Reactive Planning Problem.

3.3.1 Action Properties

As the robot moves along a path P in G_x, it takes a set of observations $\mathcal{O}_v \subseteq \Theta_v \cup \emptyset$ at each vertex $v \in P$, where $\emptyset$ is used to denote that no observation is taken at v. We define the sequence of these sets of observations to be an *observed path*.

Definition 3.4 (Observed Path)**.** Given a path $P = v_1, \ldots, v_k$ with observations $\mathcal{O}_v$ for all $v \in P$, the observed path is the sequence $\mathcal{O}_P = \mathcal{O}_{v_1}, \ldots, \mathcal{O}_{v_k}$.

The cost of an observed path can be found as the sum of travel costs and observation costs along the path:

$$\text{cost}(\mathcal{O}_P) = c(P) + \sum_{i=1}^{k} \sum_{O \in \mathcal{O}_{v_i}} \mu(O).$$

The robot's understanding of G_x, namely Y, is based on the observed path beginning at a starting vertex v_s. Two important subgraphs can be formed within this understanding.

Definition 3.5 (Known Subgraph)**.** Given a set of environmental states Y, the graph $\overline{G} = (\overline{V}, \overline{E}, c)$ induced by $\overline{E} = \{e \mid \mathbb{P}(e|Y) = 1\}$ is the known subgraph.

Definition 3.6 (Consistent Subgraph)**.** Given a set of environmental states Y, the graph $\underline{G} = (\underline{V}, \underline{E}, c)$ induced by $\underline{E} = \{e \mid \mathbb{P}(e|Y) > 0\}$ is the consistent subgraph.

The *known subgraph* includes only edges that are sure to exist, while the *consistent subgraph* includes all edges that may still exist. These graphs are updated as the robot collects

constructive observation-outcome pairs of the environment. We say an observation is *constructive* if there are at least two different, possible outcomes. Intuitively, we use the term constructive because the robot already knows the result of an observation with one outcome and thus does not need to incur the respective cost.

Definition 3.7 (Constructive Observation)**.** Given environmental states Y, an observation O is constructive if there exists $i, j \in Y$ such that $O \cap E_i \neq O \cap E_j$.

An observed path can be broken into smaller sections called *legs* that start at one constructive observation and end at the next constructive observation.

Definition 3.8 (Leg)**.** Given an observed path $\mathcal{O}_P$, a leg is a subpath of P, namely $v_i, v_{i+1}, \ldots, v_j$ where $\mathcal{O}_{v_i}$ and $\mathcal{O}_{v_j}$ are constructive observations, and each $\mathcal{O}_{v_{i+1}}, \ldots, \mathcal{O}_{v_{j-1}}$ is an empty set.

A leg can be thought of as a meta-edge between constructive observations. Since the robot can move only on edges that contain no obstacles, a leg is composed only of edges which are understood to be unobstructed after the leg's first observation set $\mathcal{O}_{v_i}$. Therefore, a leg is a sequence of move actions that join constructive observation actions.

The order in which observations can be visited depends on observation-outcome pairs to date. The following definition provides a property of an optimal complete policy that can react to the environment without re-computation of that policy.

Definition 3.9 (Reachable)**.** Given a known subgraph $\overline{G}$ and a vertex v, an observation $O \in \Theta_u$ is reachable from v if there exists a path from v to u in $\overline{G}$.

The following result ties the notion of *reachability* to that of legs between constructive observations.

Lemma 3.2. *Consider two consecutive constructive observations O_1 and O_2 on a path P. Let (Y, v) be the robot state after action $(O_1, observe)$ collects outcome $O_1 \cap E_x$. Then, in the known subgraph $\overline{G}$ defined by Y, observation O_2 is reachable from v.*

Proof. After $O_1 \cap E_x$ the understanding of the environment, namely Y, is fixed until the robot gains new information at O_2. With no loss of generality, let $O_2 \in \Theta_u$ for observation location $u \in \mathcal{V}$. The robot can only select move actions for edges that cannot be blocked given Y. $\overline{G}$, defined by Y, contains only edges that do not need to be observed before traversal; therefore, the robot can only reach O_2 if there exists a path from v to u in $\overline{G}$. □

3.3.2 Control Policy Properties

We now show how a complete policy can be efficiently represented by a tree $\pi = (N, L)$. The nodes N are tuples (Y, O) where Y corresponds to the consistent environmental states

prior to constructive observation O. The edges are defined by legs L between constructive observations. Every non-leaf node $(Y, O) \in N$ must have one leg *incident-in* and $|\Gamma_{(Y,O)}|$ legs *incident-out*. The incident-out legs connect (Y, O) to (Y_E, O') for $E \in \Gamma_{(Y,O)}$ where $Y_E = \{i \in Y | E = O \cap E_i\}$. The robot knows which leg to traverse given outcome $O \cap E_x$ matches $E \in \Gamma_{(Y,O)}$. Informally, the tree stitches together the observed paths, starting from vertex v_s, for each $i \in \mathbb{N}_m$ until observation-outcome pairs disagree at which point the tree branches. This allows real-time reaction in every possible environmental state by Lemma 3.2. We now discuss the space complexity of this encoding.

Lemma 3.3. *Any two nodes $n_1 \neq n_2 \in N$ where n_1 is not an ancestor or descendent of n_2 satisfy $Y_1 \cap Y_2 = \emptyset$ (i.e., the realization at n_1 and at n_2 must be different).*

Proof. Let $(Y, O) \in N$ be the youngest ancestor of both n_1 and n_2. Formally, $Y_1 \cup Y_2 \subseteq Y$ s.t. $Y \subset Y'$ for all other ancestors $(Y', O') \in N$. Consider two outcomes $E^1, E^2 \in \Gamma_{(Y,O)}$ such that $E_i \cap O = E^1$ for all $i \in Y_1$ and $E_j \cap O = E^2$ for all $j \in Y_2$. If $E^1 = E^2$, there exists a node $(Y', O') \in N$ such that $Y' \subset Y$ and $Y_1 \cup Y_2 \subseteq Y'$, but this contradicts the definition of (Y, O). Therefore, $E^1 \neq E^2$ implies $i \neq j$ for any $i \in Y_1$ and any $j \in Y_2$ (i.e., $Y_1 \cap Y_2 = \emptyset$). □

Lemma 3.4. *A complete policy can be represented as a tree using $\mathbb{O}(nm + m^2)$ space where n is the number of G's vertices and m is the number of edge subsets.*

Proof. The worst case encoding requires the robot to always learn random draw x before terminating. Given Lemma 3.3, we can bound the number of constructive observations the robot makes by $m - 1$ with respective node size $\mathbb{O}(m)$. We know the lowest cost leg connecting these observations will visit at most n vertices because non-negative traversal cost allows a path without cycles to always be minimum cost. Therefore, the policy can be stored as a lookup table of size $\mathbb{O}(nm + m^2)$. □

Remark 3.2 (Policy Encodings). The policy size scales with m which motivates $m \ll 2^{|\mathcal{E}|}$. A POMDP with nm states and a MDP with $n2^m$ states can be encoded for the RPP, but for our cases this is still very large.

Continuing Example 3.1, suppose the robot's observations are $\Theta_v = \{I_v\}$ for all $v \in \mathcal{V}$ with zero cost. Let moving between cells cost 1. Consider the two policies presented in Fig. 8 where edge labels are the costs of L and node labels encode Y. Note that node T indicates no goal terminal state was reached. The robot moves to A or B and collects I_A or I_B respectively. Using Eq. 4, the left and right policies in Fig. 8 render expected costs of 6.7 and 7.3 respectively. These policies satisfy both the reachability condition in Lemma 3.2 and the constructive observation property, and note the left policy allows the robot to reach G without fully knowing x.

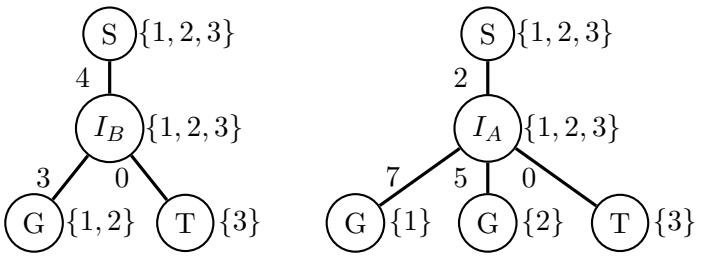

Figure 8: Possible policies for office example with different observation locations.

3.3.3 Computational Complexity

To establish the complexity of the Reactive Planning Problem, we begin by considering the following variant.

Problem 3.2 (Probable World Problem, PWP)**.** Given a graph G, a start vertex $v_s \in \mathcal{V}$ and a set of edge subsets $\mathcal{S}$ with corresponding random variable X that has a known probability mass function and observations Θ_v for all $v \in \mathcal{V}$, find a policy π that minimizes $\mathbb{E}_X(\pi)$ and identifies induced subgraph G_x for a random draw x from X.

Proposition 3.1. *The Probable World Problem is NP-Hard.*

Proof. Consider the tuple (X, d, H, ρ, O, Z, r) that defines an instance of IPP from Section 2.5. We will reduce IPP to PWP. Create a graph of vertices $\mathcal{V} = A \cup B$ where A mirrors X and B mirrors O. Let $v_s = r$. Create an edge subset E_h for every $h \in H$. In every E_h, connect A with edges of cost defined by d. For each $a \in A$ and $b \in B$, add an edge from a to $b \in B$ for subset E_h only if $Z(a, h, b) = 1$. Set observation model $\Theta_v = \{e \in \mathcal{E} | e \in I_v\}$. Let random variable X's pmf be in line with ρ. Set $\mu((a, b)) = 0$ for all observations. Consider a solution S for PWP. Change each visited vertex of A to X and each constructive observation to respective elements of O for a solution S'. The legs of S contain no verticies of B as B has no path to constructive observations. Given S identifies random draw x, S' identifies true hypothesis h. Given IPP (perfect sensing) is NP-Hard (Lim et al. 2015), PWP must be NP-Hard. □

Theorem 3.1. *The Reactive Planning Problem is NP-Hard.*

Proof. We will prove this result by reducing PWP to RPP. Consider an instance of PWP. Given the graph for PWP, add a set of vertices Q with $|Q| = m$, an intermediary vertex h and a goal vertex v_g. Connect every $v \in \mathcal{V}$ to h with 0 cost for all $E \in \mathcal{S}$. Let α be the maximum of all traversal and observation costs. We can upper bound the expected cost of any optimal policy with $\alpha(mn + m^2)$ by Lemma 3.4. Connect h bidirectionally with each

$q \in Q$ with traversal cost of U for all $E \in \mathcal{S}$ such that $(1 - \mathbb{P}(X = y))U \gg \alpha(mn + m^2)$ where $E_y \in \mathcal{S}$ is the most probable edge subset. Add an edge to $E_i \in \mathcal{S}$ from $q_i \in Q$ to v_g with cost of 0. In other words, there will only ever be one edge from Q to v_g, and it is always different for each subset. This new problem is in the form of RPP.

Suppose, by way of contradiction, there exists a solution to this RPP without solving the original PWP. This would imply there were at least two environmental states Y consistent with the observations of an observed path (starting at v_s) of the policy before attempting to reach v_g. This policy would move the robot to $q_i \in Q$ and observe the edge from q_i to v_g for $i \in Y$ (only exists in E_i). The policy must react to $(q_i, v_g) \cap E_x = \emptyset$. The resulting expected cost is at least $p_i U + (1 - p_i)2U$. Given $(1 - p_i)U \gg \alpha(mn + m^2)$, there exists a policy that can do better as $\alpha(mn + m^2)$ is an upper bound on an optimal policy which is a contradiction. This shows RPP solves PWP. Given Proposition 3.1, RPP is NP-Hard. □

Remark 3.3 (NP-Complete). In the decision version of RPP we are given a budget B and asked to find a complete policy with expected cost less than or equal to B. From Lemma 3.4, it is straightforward to see that the decision version is in NP, and thus is NP-Complete. Polychronopoulos & Tsitsiklis (1996) provides a similar result for R-SSPPR, but we were unable to find a gadget from RPP to R-SSPPR or vice versa.

3.4 Policy Generation

The Reactive Planning Problem seeks information to reach the goal. The robot explores until it is beneficial to exploit the observed information and move to the goal. We address this efficiently via a mutual information heuristic and optimally via dynamic programming.

3.4.1 Mutual Information Policies

Our heuristic for policy generation can be broken into its method for exploring the environment, and attempting to reach the goal. Given a state (Y, v) our method for computing a policy requires the set of all reachable constructive observations. We denote this set as R_v, which contains observation-vertex pairs (O, u), and is defined as

$$R_v = \{(O, u) \mid O \in \Theta_u \text{ is constructive and is reachable from } v\}.$$

Recall the definitions of constructive and reachable observations are given in Definitions 3.7 and 3.9, respectively.

Exploration: Consider the RPP. By Lemma 3.2, information can only be collected at the set of reachable observations. To select which constructive observation is beneficial, we maximize mutual information extended from (Charrow et al. 2014, Dames et al. 2016, Lim et al. 2015).

Let X_Y encode the probability distribution over environments given a set of consistent environmental states Y. Its probability mass function is given by

$$\mathbb{P}(X_Y = i) = \begin{cases} \frac{\mathbb{P}(X=i)}{\sum_{j \in Y} \mathbb{P}(X=j)} & \text{for } i \in Y \\ 0 & \text{otherwise} \end{cases}. \tag{5}$$

Mutual information is the difference between entropy of X_Y and conditional entropy of X_Y, given a reachable constructive observation $(O, u) \in R_v$. Formally,

$$MI(X_Y, (O, E_O)) = H(X_Y) - H(X_Y|(O, E_O)) \,. \tag{6}$$

The entropy of X_Y, $H(X_Y)$, does not depend on the observation outcome pair (O, E_O); therefore, this problem can be reduced to minimization of conditional entropy,

$$H(X_Y|(O, E_O)) = - \sum_{E \in \Gamma_{(Y,O)}} \mathbb{P}(E_O = E) \sum_{i \in Y} \mathbb{P}(X_Y = i|E_O = E) \log(\mathbb{P}(X_Y = i|E_O = E)) \,. \tag{7}$$

Exploitation: The robot must be able to decide when it has collected enough information. We begin with the following inequality from the principle of optimality,

$$c_G(v, v_g) \leq c_G(v, u) + \mu(O) + c_G(u, v_g) \quad \text{for each } (O, u) \in R_v, \tag{8}$$

where the subscript on the cost function c indicates the realization of the environment in which the cost is calculated.

Intuitively, making a measurement and going to the goal is at least as expensive as going straight to the goal in G. The cost calculated in G often performs poorly as an under-estimator for G_x. To address this, a new cost-to-go function is calculated as an expectation over the possible environmental states Y. The expected cost-to-go,

$$\mathbb{C}_Y(u, v_g) = \sum_{i \in Y} c_{G_i}(u, v_g) \mathbb{P}(X_Y = i) \,, \tag{9}$$

is found for every vertex $u \in \mathcal{V}$. To calculate $c_{G_i}(u, v_g)$, the edges are flipped in each G_i and a shortest path algorithm is run from v_g to all other $u \in V_i$. If $c_i(u, v_g)$ is infinite, we set such costs to zero as the robot will not travel any further (i.e., no goal terminal state).

Eq. 8 is augmented to include the robot's current environmental understanding and the expected cost-to-go. Given an observation-vertex pair $(O, u) \in R_v$, the pruning inequality can be written as

$$c_{\overline{G}}(v, v_g) \leq c_{\overline{G}}(v, u) + \mu(O) + \mathbb{C}_Y(u, v_g). \tag{10}$$

If this inequality is satisfied, then it is less expensive for the robot to traverse straight to the goal than for it to make observation O at vertex u, and thus this information should not be collected.

Lemma 3.5. *Given a robot state r with constructive observation-vertex pairs R_v, if all $(O, u) \in R_v$ satisfy Eq. 10, then the robot should move to the goal.*

Proof. Consider $c_{\overline{G}}(v, v_g) = \infty$. This implies there is no known path to goal. No observation satisfies Eq. 10, so this trivially holds. Now, consider the case where enough information has been gathered to $r = (Y, v)$ for $c_{\overline{G}}(v, v_g) < \infty$. If all $(O, u) \in R_v$ satisfy Eq. 10, the known cost of making any observation and the expected cost-to-go is more than the known cost to complete the task. Thus, the robot should move to the goal. □

Lemma 3.6. *The expected cost-to-go from the start, $\mathbb{C}_{\mathbb{N}_m}(v_s, v_g)$, forms a lower bound on the expected cost of any policy π.*

Proof. Consider any two environmental states $i, j \in \mathbb{N}_m$. If G_i and/or G_j do not have paths to the goal, the robot must identify the environmental state and return no goal terminal state. To do this, the robot uses an observed path to gain the information. The cost of such a path is at least 0. The expected cost-to-go for these cases is always 0. Suppose G_i and G_j can both reach the goal. There is at least one leg the robot must travel for both G_i and G_j. The expected cost-to-go selects the optimal paths independently. Therefore, the expected cost of the observed paths from π for i and for j can never be less than the expected cost-to-go, even if the robot acts optimally otherwise. □

Combining Exploration and Exploitation: To combine information gain and motion to goal, we can use any function of the exploration metric $H(X_Y|(O, E_O))$ and the exploitation metric $c_{\overline{G}}(v, u) + \mu(O) + \mathbb{C}_Y(u, v_g)$.

Weighted conditional entropy (Suhov et al. 2015) is a well-studied method for combining entropy with a second metric, and within this method, when at state (Y, v), we select observations satisfying

$$O_{\min} = \underset{(O,u)\in R_v}{\operatorname{argmin}} \left(c_{\overline{G}}(v, u) + \mu(O) + \mathbb{C}_Y(u, v_g)\right) H\left(X_Y|(O, E_O)\right) . \tag{11}$$

We also test a weighted sum of the terms in order to select observations,

$$O_{\min} = \underset{(O,u)\in R_v}{\operatorname{argmin}} \left(c_{\overline{G}}(v, u) + \mu(O) + \mathbb{C}_Y(u, v_g)\right) + \rho H\left(X_Y|(O, E_O)\right) , \tag{12}$$

where ρ is tuned based on the importance of information. In Section 3.6, we show the strength of each case.

Detailed Algorithm: Algorithm 2 selects observations that satisfy either Eq. 11 or Eq. 12. It requires the function $Reachable(G, \mathcal{S}, (Y, v))$ which computes the minimum path lengths $d[u]$ from v to all other vertices u in the known subgraph $\overline{G}$. The set R_v is formed from constructive observations which render finite path cost from v, where D_v is an array recording the distance $d[u]$ to each vertex u.

Algorithm 2: RPP Minimization of Conditional Entropy Policy

Data: Graph G, edge subsets $\mathcal{S}$, vertices v_s & v_g, states $\mathbb{N}_m$, probabilities p
Result: Policy π for RPP and expected cost lower bound l_{move}

```
1  Compute c_{G_i}(v, v_g) for all v ∈ V and i ∈ N_m;
2  Let Q contain only (N_m, v_s);
3  while Q not empty do
4      Remove (Y, v) from Q;
5      if c_{G_i}(v, v_g) = ∞ for all i ∈ Y then
6          Mark π, at v for Y, no goal terminal state;
7      else
8          Compute (R_v, D_v) = Reachable(G, S, (Y, v));
9          Remove elements of R_v that satisfy Eq. 10;
10         if |R_v| = 0 then
11             Add leg from v to v_g, marked goal terminal state, to π;
12         else
13             Let (O, u) ∈ R_v be the minimum of Eq. 11 (or Eq. 12);
14             Add leg from v to u and node (Y, O) to π;
15             Add (Y_E, u) to Q for each E ∈ Γ_{(Y,O)};
16 Let l_move = C_{N_m}(v_s, v_g);
17 Return π and l_move;
```

Remark 3.4 (Runtime). The runtime of Algorithm 2 is dominated by the m calls to Dijkstra's Algorithm, which gives complexity $\mathbb{O}(m(|\mathcal{V}|+|\mathcal{E}|)\log|\mathcal{V}|)$ (priority queue implemented as a binary heap).

The biased cost for selecting an observation in Line 13 and for pruning observations in Line 9 complement each other to provide incentive toward the goal. The biased cost encourages observation selection closer to the goal. Once enough information is gained, the pruning condition removes information that is not important for the task. When the pruning condition removes all observations from R_v, the robot moves to the goal.

Remark 3.5 (Parallel Computation). Given Lemma 3.3, the policy π, returned by Algorithm 2, is independent of the order in which states are removed from Q in Line 4 (potential for parallel computation).

Theorem 3.2. *Algorithm 2 returns a complete policy.*

Proof. Suppose by contradiction, Algorithm 2 did not return a complete policy. This would imply either it terminates at (Y, v_g) for $Y \subseteq Y_{\text{no goal}}$ (false positive) or it terminates a (Y, v) for $v \neq v_g$ and there exists $y \in Y$ such that $y \in Y_{\text{goal}}$ (false negative).

False positive: Algorithm 2 must have directed the robot to travel over an obstructed edge because $Y \subseteq Y_{\text{no goal}}$. Since each realization G_i for $i \in Y$ does not have a path to goal, the cost $c_{G_i}(v, v_g)$ will be infinite for all $i \in Y$. Line 5 is satisfied and Line 6 sets this state to terminal no goal.

False negative: Algorithm 2 would not be able to find a path in each realization G_i for $i \in Y$, but since the environmental state y is still possible, $c_{G_y}(v, v_g)$ will have finite cost. This will not satisfy Line 5, and thus, this state cannot be marked no goal terminal state by Line 6. □

Remark 3.6 (Online Policy Generation). Algorithm 2 can be used in an online manner as follows. For a state (Y, v), find the reachable observations and select the minimizer of Eq. 11 or Eq. 12, namely $(O, u) \in R_v$ with $O \in \Theta_u$. Execute shortest path in $\overline{G}$ from v to u and take O. Update (Y, v) given $(O, O \cap E_x)$ and repeat.

3.4.2 Lower Bound on Observation Costs

The lower bound l_{move} from Algorithm 2 does not account for any observation costs. We formulate a lower bound l_{obs} on the expected observation cost such that its sum with l_{move} remains a lower bound on an optimal policy. To accomplish this, a new observation is created for a state (Y, v) from the union of all constructive observations in the reachable set R_v, namely $\theta = \cup_{(O,u)\in R_v} O$. This observation is available at v for the cost $\mu(\theta) = \min_{(O,u)\in R_v} \mu(O)$. We make this new observation if

$$c_{\overline{G}|Y}(v, v_g) \geq \sum_{E \in \Gamma_{(Y,\theta)}} \mathbb{P}(X_Y \in Y_E) c_{\overline{G}|Y_E}(v, v_g) + \mu(\theta)\,, \tag{13}$$

where subscript $\overline{G}|Y$ indicates costs are found for $\overline{G}$ defined by Y.

Lemma 3.7. *Algorithm 3 finds a lower bound on the expected observation cost.*

Proof. First, we must show Algorithm 3 will not make more than the minimum number of observations required. If Line 5 is not met then all $i \in Y$ have no path to goal; thus, the robot may terminate because $Y \subseteq Y_{\text{no goal}}$. Consider Eq. 13. The left-hand-side is the lowest cost path from v_s to v_g without any more observations. The right-hand-side is the expected cost from v_s to v_g, given θ's outcome, for the cost $\mu(\theta)$. When this inequality holds, the added freedom to select a path with the outcome's information saves more than it costs to make the observation. Otherwise, the observation may not improve the known path for future states; thus, the robot will not take the observation θ.

Algorithm 3: RPP Observation Lower Bound

Data: Graph G, edge subsets $\mathcal{S}$, vertices v_s & v_g, states $\mathbb{N}_m$, probabilities p
Result: Lower bound on expected observation cost l_{obs}

```
1  Compute c_{G_i}(v, v_g) for all v ∈ V and i ∈ N_m;
2  Add N_m to Q and set l_obs = 0;
3  while Q not empty do
4      Remove Y from Q;
5      if c_{G_i}(v_s, v_g) ≠ ∞ for any i ∈ Y then
6          Compute (R_{v_s}, D_{v_s}) = Reachable(G, S, (Y, v_s));
7          if Eq. 13 then
8              l_obs = l_obs + P(X ∈ Y)μ(θ);
9              Add Y_E to Q for each E ∈ Γ_{(Y,θ)};
10 Return l_obs;
```

If an observation is required, the optimal observation is in R_v by Lemma 3.2. This information is collected for the minimum cost of any observation in R_v in Line 8. This lower bounds the expected cost of the optimal observation. Therefore, no more than the minimum number of observations are made, each with minimum cost, showing that C lower bounds the expected observation cost. □

3.4.3 A Dynamic Program for the Optimal Policy

The research by Polychronopoulos & Tsitsiklis (1996) presents an exponential-time dynamic program to solve the R-SSPPR problem. Building on their formulation we can create a dynamic programming solution to the reactive planning problem as follows. We define the subproblem $V(Y, v)$ to be the optimal expected cost to go from v in belief Y to the goal v_g. Any belief Y such that $Y \subseteq Y_{\text{no goal}}$, has an optimal cost to go of $V(Y, v) = 0$ for all $v \in \mathcal{V}$. Following Polychronopoulos & Tsitsiklis (1996), the dynamic programming recursion is

$$V(Y, v) = \min_{(O,u)\in R_v} \left[c_{\overline{G}}(v, u) + \mu(O) + \mathbb{E}_{X_{Y_E}}[V(Y_E, u)] \right], \tag{14}$$

where Y_E is the consistent environmental state after observation O gives the outcome $E \in \Gamma_{(Y,O)}$. The recursion looks over all reachable constructive observations and determines the cost to obtain that observation, followed by the optimal expected cost-to-go to the goal, versus the cost to go directly to the goal from the current state (Y, v).

To implement this, we use memoization and propagate the solution forward from initial state $(\mathbb{N}_m, v_s)$. This complements the fact that the next observation must be in the reachable set and often limits the number of states examined. However, note that the number of subproblems $V(Y, v)$ can in general be exponential in the number of realizations m, since $Y \subseteq \mathbb{N}_m$. Our solution uses recursion on each reachable observation to identify the best

observation to make, and we compare this to Algorithm 2's simplified selection method in the simulations.

3.5 Extension to Faulty Sensors

In some applications, the robot sensor may erroneously miss obstacles (false negatives) or detect obstacles that are not actually present (false positives). The reactive planning problem can be extended to certain faulty sensor models. We begin by discussing the difficulties of a general faulty sensor model, and then discuss a special case that can be handled directly.

In a general model, we have a probability $p_O \in [0,1]$ for each observation $O \in \cup_{v\in\mathcal{V}}\Theta_v$, where the robot receives the correct outcome $O \cap E_x$ with probability p_O and an incorrect subset of O with probability $(1-p_O)$, which can consist of both false negatives and false positives. In this case, an observation-outcome pair (O,E) cannot eliminate any environmental state unless it makes an observation O for which $p_O = 1$. Instead of updating Y, observation-outcome pairs update the distribution of X_Y. A natural way to address this is to allow the robot selection of action (e, move) at v even when $\mathbb{P}(e|Y) < 1$. The robot then either successfully traverses the edge, or detects an obstruction using a proximity sensor, or by physical contact, returning to v and incurring cost $c(e)$. This model now bears close resemblance to the form of POMDP presented by Papadimitriou & Tsitsiklis (1987), and we conjecture that as with POMDPs, this general extension is PSPACE-hard. This however, would imply that the optimal policy structure no longer takes the compact form discussed in Lemma 3.4. Thus, we limit our attention to the following class of faulty sensors for which our policy structure still applies.

Structured Sensor Failures: To motivate the idea of structured failures, consider deploying a robot in a building that may contain obstacles with reflective surfaces. Certain sensors have difficulty detecting these surfaces and thus may return incorrect outcomes. We term this as a failure type and quantify it with a faulty edge subset $E \in \mathcal{F}$ where $\mathcal{F} = \{E_{m+1}, E_{m+2}, \ldots, E_{m+f}\}$ is the set of $f \in \mathbb{Z}^+$ possible failure types. The failure experienced by the robot is encoded as a random variable W taking values in $\{0,1,\ldots,f\}$. The outcome $W = 0$ corresponds to no sensor failure (i.e., each outcome is $O \cap E_x$), and occurs with probability $p_{\text{nf}} \in [0,1]$. The failure type $w \in \mathbb{N}_f$, corresponding to edge subset E_{m+w} occurs with probability $\mathbb{P}(W = w)$, where $\sum_{w=1}^{f}\mathbb{P}(W=w) = 1 - p_{\text{nf}}$. The robot knows the pmf of W, but not its random draw w. If the sensor experiences failure type $w \in \mathbb{N}_f$, then the outcome of each observation O will be $O \cap E_{m+w}$, which may not agree with $O \cap E_x$. Thus, this model captures correlated failures: if failure type w occurs when the robot is operating in realization x, then all edges in $E_{m+w} \setminus E_x$ will give false negative, and all edges in $E_x \setminus E_{m+w}$ will give false positives.

The robot state is expanded to capture both the possible realizations of G_x, namely

$Y \subseteq \mathbb{N}_m$, and the indices of edge subsets consistent with the observation-outcome pairs, namely $Z \subseteq \mathbb{N}_{m+f}$. Formally, the robot state is $(2^{\mathbb{N}_m}, 2^{\mathbb{N}_{m+f}}, \mathcal{V})$. The set Y records all environmental realizations that could be consistent with the observations (given that they may be faulty). The set Z records the possible edge subsets in $(E_1, \ldots, E_{m+f})$ that are consistent with the measurements. Under this new sensor model, we provide the actions that update the robot's environmental information.

Definition 3.10 (Constructive Observation)**.** Given state (Y, Z, v) with $O \in \Theta_v$, $(O, \text{observe})$ is constructive if there exist $i, j \in Z$ such that $O \cap E_i \neq O \cap E_j$.

Definition 3.11 (Reactive Move)**.** Given state (Y, Z, v) with $e \in I_v$, (e, move) is reactive if there exist $i, j \in Y$ such that $e \in E_i$ and $e \notin E_j$. The robot remains at v if $e \notin E_x$ and incurs $c(e)$.

Given Z, a constructive observation O has a set, $\Gamma_{(Z,O)} \equiv \{E \subseteq \mathcal{E} | E = O \cap E_i, i \in Z\}$, of possible outcomes where outcome $E \in \Gamma_{(Z,O)}$ updates Z to $Z_E = \{i \in Z | E = O \cap E_i\}$. A reactive move, (e, move), partitions Y and Z into edge subsets that contain e and those that do not. If the robot state (Y, Z, v) satisfies $Z \subseteq \mathbb{N}_m$ (i.e., the robot knows no fault types have been encountered), the problem returns to the RPP under perfect sensing with start state (Z, v). Note the robot may reach a terminal state without satisfying this special case.

The tree policy can be extended by changing the nodes to (Y, Z, a) where action a is a constructive observation or reactive move. To calculate its expected cost, we extract each root to leaf observed path. Given realization $x \in \mathbb{N}_m$, there are $f + 1$ observed paths corresponding to sensing $x \in \mathbb{N}_m$ (observed paths $A_{x|x}$) and sensing $w \in \mathbb{N}_f$ (observed paths $A_{m+w|x}$). The expected cost of policy π is,

$$\mathbb{E}(\pi) = \sum_{x \in \mathbb{N}_m} \left(p_{\text{nf}} \mathbb{P}(X = x) \text{cost}(A_{x|x}) + \sum_{w \in \mathbb{N}_f} \mathbb{P}(W = w) \text{cost}(A_{m+w|x}) \right).$$

Lemma 3.8. *A complete policy tree has at most $2m + f - 2$ nodes where m is the number of realizations and f is the number of faulty edge subsets.*

Proof. The largest policy tree requires the robot to learn if it is getting incorrect outcomes and then find the true realization x via reactive moves. Every constructive observation partitions Z and it requires at most $m + f - 1$ constructive observations to identify z or x. If it identifies z, only reactive moves can partition Y and it takes $m - 1$ reactive moves to learn every possible x. This can be encoded into $2m + f - 2$ nodes. □

Algorithm 2 can be altered to handle this faulty sensor model as follows. The queue now holds the new state, and Y is still used to calculate $\overline{G}$. The function $\mathit{Reachable}(G, \mathcal{S} \cup \mathcal{F}, (Y, Z, v))$ must also find the reactive moves that can be reached with finite cost. Finally, Eq. 7 must be altered to capture the conditional entropy of the new state given a reactive move or a constructive observation.

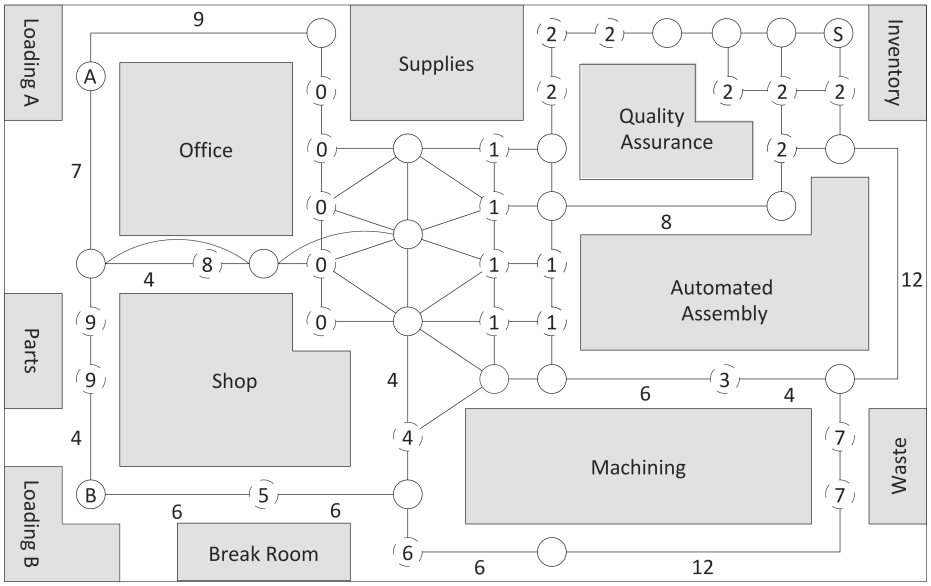

Figure 9: Flexible factory model with potential obstacles.

3.6 Simulation Results

In this section we provide simulation results on a large scale practical example and on randomly generated environments. We compare against online algorithms and the optimal solution presented in Section 3.4.3. Tests where run on a single Intel Core i7-6700 at 3.4GHz.

3.6.1 Flexible Factory

Flexible factories often spend considerable downtime between contracts due to changes in infrastructure and machinery. Consider Fig. 9 as a simple flexible factory where dashed vertices may be obstructed and edges with traversal costs of 2 or 3 are not labelled for simplicity. Note the curved edges have a traversal cost of 2 more than the shortest path between their end points, which corresponds to the robot waiting for the obstacle to move. The factory is interested in knowing the expected cost (e.g. expected time and/or fuel) to move items from inventory to the loading areas.

The dashed vertices indicate areas that require heavy use. For clarity, in Table 1 the column labelled "Vertex Obstructions" indicates the properties of the environment obstruction. For instance, in region 0 (vertices labelled 0) up to two vertices may be missing from the graph. Regions 1 and 2 each contain one forklift obstruction (which corresponds to removing the two adjacent vertices it occupies). When regions 5 and 6 are obstructed, all other vertices exist.

We cast this as a Reactive Planning Problem (RPP) by enumerating all combinations

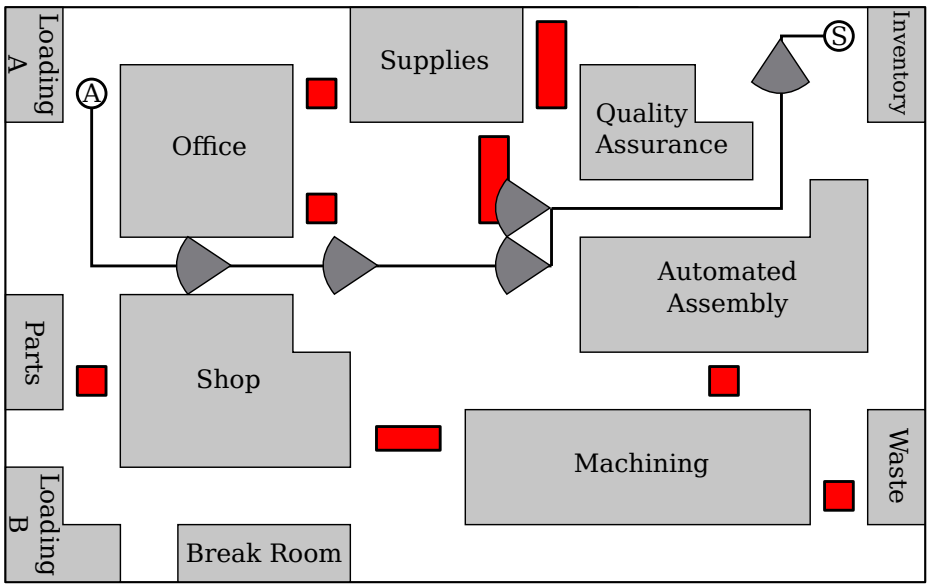

Figure 10: Example policy run from S→A for flexible factory example.

of the obstructed vertices and removing their incident edges. This generates 34561 edge subsets each with a corresponding probability. We compute policies from S to A, from S to B, from A to S, and from B to S. The robot is faster when not loaded, so the movement costs of A to S and B to S are decreased by a factor of 2.

Table 1: Flexible factory model parameters used in simulations.

Regions	Vertex Obstructions	Obstruction Probability	Observation Cost
0	≤ 2 independent	uniform over combinations	0.5
1,2	2 adjacent vertices	uniform over combinations	0.25
3,4	1	0.3	0.25
5,6	both or none	0.02	0.5
7,9	≤ 1 independent	0.1	0.5
8	1	0.4	0.5

Due to the size of the environment, we will allow replanning during the online phase for comparison purposes (note these results do not have constant action lookup time). We compare against A* and maximum probability of success ($\mathbb{P}_s$). Both approaches generate a path, which we follow until it is obstructed. Then we take the edges of the path as observations and use this new information to replan. This is completed for every realization $x \in \mathbb{N}_m$. The cost of the corrected paths from v_s to a terminal state and X's pmf are used to calculate the expected cost found in Table 2. We also compare against the feedback heuristic from Polychronopoulos & Tsitsiklis (1996). In contrast to the two prior algorithms,

if any new information is collected the robot replans with this new information. Given the difference between the R-SSPPR and the RPP, the optimal expected cost path to goal may not be well-defined. To remedy this, we let the robot find the shortest path in $\underline{G}$ and make the lowest cost constructive observations required to use this path. If an observation is made, the robot replans with the acquired information applied to $\underline{G}$.

Table 2: Flexible factory simulation results.

	Algorithm 2		Eq. 11	Eq. 12	A*	Max $\mathbb{P}_s$	Feedback
Task	$\lvert N\rvert$	$l_{\text{move}} + l_{\text{obs}}$	$\mathbb{E}_X(\pi)$	$\mathbb{E}_X(\pi)$	$\mathbb{E}_X$	$\mathbb{E}_X$	$\mathbb{E}_X$
S→A	114	40.0	**42.5**	42.6	47.7	85.3	47.3
S→B	6869	42.3	**50.0**	50.0	50.1	61.7	53.7
A→S	84	20.6	**23.3**	23.3	30.2	43.6	29.9
B→S	175	22.0	25.7	**25.4**	28.3	33.9	27.6

To show how a policy emits an observed path, we display the actions taken for an environmental realization in Fig. 10. Although the obstacles are shown, the robot does not know the exact location of several of them unless these obstacles have been sensed. If any sensing action returned a different outcome, we could expect a different path. This facilitates our desired real-time reaction.

In Table 2, the number of observation nodes in the policy is given by $|N|$; the upper bound on this number is given by $m - 1$, namely 34560, for reference. Note the column for Eq. 12 shows expected cost for optimized ρ. The proposed algorithm provides lower expected cost than the three online solutions. In addition, for each task the expected cost of Algorithm 2 is within 20% of the lower bound ($l_{\text{move}} + l_{\text{obs}}$) on the optimal expected cost. In general, the feedback heuristic should always perform at least as well as A*. Notice this is not the case for task S→B, but this occurs due to the different tie-breaking methods for shortest path in each solution. When comparing Eq. 11 and Eq. 12, the solution for weighted conditional entropy closely compares to the additive method for optimized ρ and is computed only once verses multiple runs for different values of ρ. We contribute this to the correlated environment structure, and in the next section we show, when the environments are randomized, this is not always the case.

3.6.2 Performance versus Optimal

In order to test the performance of the lower bound as well as the proposed algorithm, we compare against the optimal solver discussed in Section 3.4.3 as well as the feedback heuristic discussed in Section 3.6.1. The test environments are cell worlds where edges connect left, right, up and down cells. Traversal and observation costs are set uniformly at random between $[5, 6]$ and $[1, 2]$ respectively. The edge subsets are generated by adding obstacles randomly. To do this, we incrementally relax the addition of obstacles into a

realization until a path to goal exists and vice versa for no path to goal. Finally, X's pmf is formed iteratively by selecting a realization uniformly at random from the remaining unselected realizations and assigning it one fourth of the remaining probability (the final realization takes the remaining mass). Due to the exponential runtime nature of the optimal solver, we test on grids of fewer than 40 vertices and up to 100 possible realizations.

The results in Fig. 11 highlight a shortcoming in the lower bound, $l_{\text{move}} + l_{\text{obs}}$. The lower bound does not perform well for cases where the probability a random draw does not have a path to goal is high. This is expected because the lower bound l_{move} assumes the robot does not move when a realization does not contain a path to the goal. In practice, the robot must traverse and observe to identify if it can make its way to the goal.

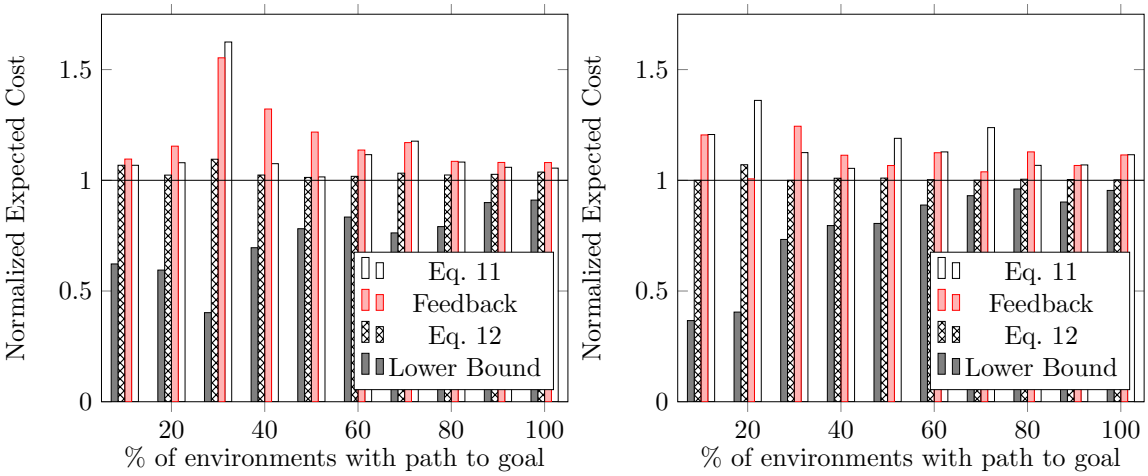

Figure 11: Comparison of proposed approach versus the optimal. Left: G is a 5x5 grid with single edge observations and 40 realizations. Right: G is a 6x6 grid with single hop observations and 100 realizations.

Fig. 11 also displays a strength in Eq. 12 over Eq. 11 and the feedback heuristic. First, Eq. 12 with an optimized ρ is always within 10% of the optimal while both Eq. 11 and the feedback heuristic are within 63% of the optimal. These environments are randomly generated, and thus their structure does not foster a strong relationship between obstacles. When this is the case, the weighted additive method of Eq. 12 allows the robot to put less importance on information gathering, $H(X_Y, (O, E_O))$, and more importance on the cost of traversal, $(c_{\overline{G}}(v, u) + \mu(O) + \mathbb{C}_Y(u, v_g))$.

We also test the runtime of Algorithm 2 by fixing the number of vertices in the grid while varying the number of realizations and fixing the number of realizations while varying the number of vertices. Ten percent of the realizations have no path to goal. The timing results in Fig. 12 are generated based on a C-programming implementation. For a low number of realizations, a single observation greatly impacts where the robot can traverse; in contrast, as more realizations are introduced, the robot requires more observations to

traverse a similar area (more while loop iterations) as seen in the left plot. The right plot shows computation time dominated by Dijkstra's algorithm as expected given the number of vertices are growing while the number of realizations remain the same.

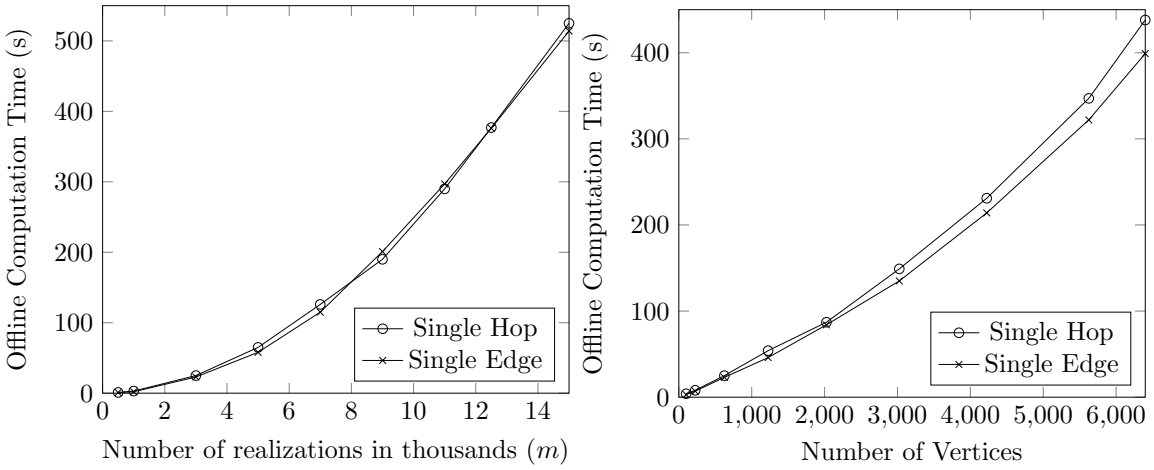

Figure 12: Empirical runtime analysis of the proposed approach. Left: G has 625 vertices and 2600 edges for varied number of realizations. Right: G has 3000 realizations for varied number of vertices and edges.

3.6.3 Performance with Inaccurate Prior Data

Often prior data fails in that it may not accurately represent the physical environment the robot functions in (i.e., the realized edge subset is not an element of $\mathcal{S}$). We test Algorithm 2's policy by randomly selecting k edges of the realized environment and complementing their obstruction (i.e., adding an obstacle if one does not exist and vice versa). The robot follows the policy until it reaches a terminal state or finds an inconsistency, namely state $(\emptyset, v)$. If $Y = \emptyset$, the robot switches to the A* algorithm presented in Section 3.6.1 where all uncertain edges are assumed to exist but still must be sensed. The A* algorithm always uses a single hop observation model.

The environment is a 12x12 grid with 400 realizations where 10% have no path to goal. For a fixed percentage of inaccurate edges, we perform 20 independent tests using both single edge and single hop observation models and normalize the expected cost by the expected cost of Algorithm 2's policy found assuming no errors. Prior tests show Eq. 11 strongly values information; thus, we test Algorithm 2, using Eq. 11, to show how corrupted information can adversely affect the expected cost.

The results in Fig. 13 show the single hop observation model has higher expected cost than the single edge observation model. This occurs because the single hop model is more likely to identify errors in the environment given it often collects more information; thus as

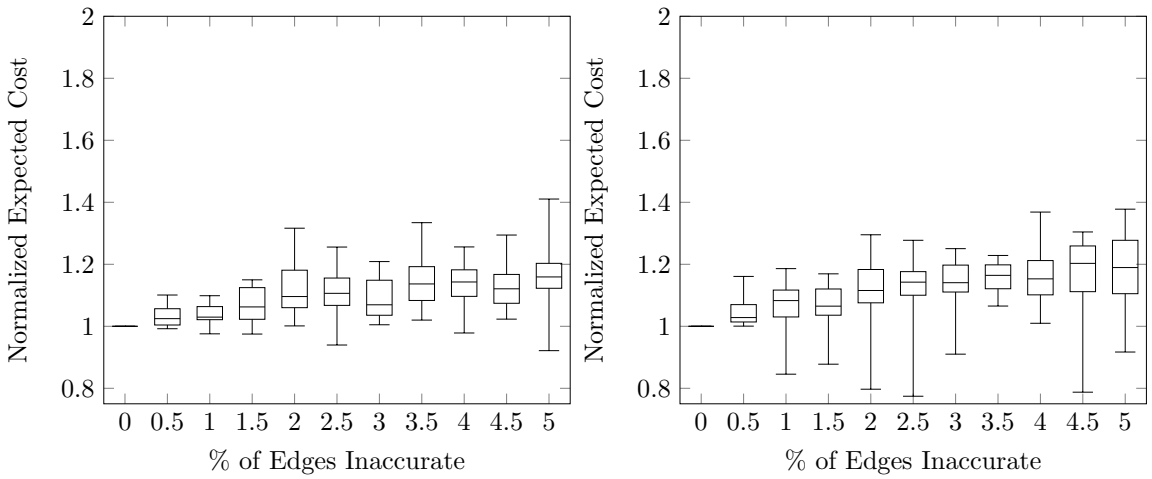

Figure 13: Imperfect prior data under single edge observation model (left) and single hop observation model (right).

error grows, this model abandons its prior knowledge sooner causing a tendency to increase the expected cost (seen in the right plot). Also, the single edge observation model only senses edges that may be blocked so an error will only be encountered on a leg. The robot belief may not satisfy $x \in Y$, yet it often continues travelling toward the goal before abandoning the prior knowledge.

Notice values less than one in Fig. 13 correspond to particular errors in the environment. Edges may become unblocked causing a lower cost to goal. The robot may become stuck before reaching a terminal state as the strongly connect component containing v_s may be broken (see Fig. 7 for an example). Also, a terminal no goal state may be reached when originally there was a path to goal due to added obstructions.

Chapter 4
Planning with Hidden Environment Model

In settings ranging from factories to restaurants, many tasks are highly repetitive, and these tasks may appear ideal for a robot to execute and/or assist in streamlining. In practice, this may not always be true and typically is not when the environment is highly uncertain. In these environments, robots can map (e.g., using occupancy grid map) and complete their task separately or in unison. The environmental uncertainty makes it difficult to harness a prior map to complete a future task; thus, for simplicity, a designer may ask the robot to re-map or use only the most recent map to guide the robot's direction.

In contrast, this work focuses on identifying fundamental structure in the environment and planning within this structure to complete the task or identify that the task is impossible to complete. We provide two methods to use what the robot maps during a task to incrementally build (i.e., update after every task) the policy which guides the robot. The first method does not require storage of prior environmental maps, but we show the robot's performance is dependent on the order in which these maps are acquired. In other words, if the robot poorly completes an initial task for a given environment, we show the performance for the remaining tasks may be adversely affected. Our second method uses a filter to store only the most relevant maps, and we show that this method removes the dependence on the order in which the robot experiences environments.

4.1 Introduction

Chapter 3 discussed the Reactive Planning Problem where the robot has access to the environment model. In practice, the model parameters may be inaccurate as tested in Section 3.6.3 or even unavailable. The goal of this chapter is to learn obstacle correlations and use these correlations to improve the robot's performance as more tasks are completed. Given the environment model is hidden, this problem has added complexity as the number of possible obstacles is exponential in the number of edges in the graph. We divide the problem into two sub-problems. The first sub-problem handles unexpected obstacles by using an online reactive algorithm, and we show the second sub-problem is closely related to the Reactive Planning Problem. As the robot completes tasks, we show the probability the online reactive algorithm is used monotonically decreases. This allows the robot to learn real-time reaction to obstacles and decreases time spent replanning during the task.

4.2 Problem Models

Throughout this chapter, the robot functions within a directed graph $G = (\mathcal{V}, \mathcal{E})$ with the cost $c : \mathcal{E} \to \mathbb{R}_{\geq 0}$ for traversing an edge and the cost $\mu : \mathcal{E} \to \mathbb{R}_{\geq 0}$ for sensing if an edge is blocked. Note this is a simplification of the observation model in Chapter 3 in order to reduce the required notation. The robot is given G without the prior knowledge of edge blockages within G, yet we require the robot to complete T tasks. In each task, the robot is required to move from a start vertex $v_s \in \mathcal{V}$ to a goal vertex $v_g \in \mathcal{V}$ or identify it is impossible to move to v_g; however, the set of blocked edges may change after a task is completed. We allow the robot to store what it has learned in prior tasks to aid future tasks.

4.2.1 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 a set of graphs 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, the robot does not know the pmf over $\mathcal{G}$. The RPP is in fact a special case of this environment model where $\mathcal{S} = \{E_1, \ldots, E_m\}$ from Section 3.2.1 contains only edge subsets which have non-zero probability of occurrence. The robot experiences a sequence of T random graphs $G_{X_1}, \ldots, G_{X_T}$ for random variables $X_1, \ldots, X_T$ independent and identically distributed (i.i.d) according to the pmf over $\mathbb{N}_r$ (i.e., $\mathbb{P}(X_t = j) = p_j$ for $j \in \mathbb{N}_r$ 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 $\mathcal{G}$ dominate the pmf. That is to say, the probability a graph is drawn from this set is much greater than the probability it is not. 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 does not provide the intended increase in functionality over pre-existing online reactive algorithms. From a practical point of view, we are interested in well structured environments (even though that structure is unknown), which on occasion has unexpected structure. We believe this captures many environments where certain areas are often blocked or unblocked (e.g., a doorway) but others are expected to be a given state (e.g., empty room is expected to be unblocked). Our work still reacts to the unexpected case, but we wish to speed up reaction time for the most probable cases. Section 4.6 continues with the flexible factory presented in Chapter 3 to provide an example of this case.

4.2.2 Robotic Model

Suppose the robot functions within the realization $G_{x_t} = (\mathcal{V}, E_{x_t})$ of $G_{X_t} = (\mathcal{V}, E_{X_t})$ for some $x_t \in \mathbb{N}_r$. 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. When we refer to an edge's state, we are referring to its blocked or unblocked status within E_{x_t}. If the robot, positioned at $v \in V$, wishes to traverse $e = (v, u) \in \mathcal{E}$, it arrives at u only if $e \in E_{x_t}$; otherwise, we say the move fails and the robot remains at v. The robot pays $c(e)$ independent of success or failure. This model assumes the robot is capable of long range sensing (sense action) and proximity sensing (identifies failed moves). Therefore, there are two methods for the robot to determine the state of an edge: 1) sense and pay $\mu(e)$, or 2) try to move across the edge and pay $c(e)$. Note the position of the robot is altered for a move over e when $e \in E_{x_t}$; thus, when comparing the cost difference between a move and an observation, one must also consider the final position. In this chapter, we focus on $\mu(e) < c(e)$ for simplicity.

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

4.3 Problem and Approach

Given a graph G with unknown pmf over all subgraphs $\mathcal{G}$, a start and goal $v_s, v_g \in \mathcal{V}$, the objective of this work is to minimize the expected cost of completing T tasks that each traverse from v_s to reach v_g or show v_g cannot be reached. Within this section, we show this problem in PSPACE-Hard, and we discuss an approach that combines a policy, similar to the policy for the RPP from Chapter 3, with an existing online reactive algorithm. The main difficulty in this work is that actions during a given task have implicit dependency on future tasks. For example, it may be difficult to decide if the cost of sensing an edge has value to reduce cost in future tasks.

4.3.1 Complete Policy

Let the current task be $t \in \mathbb{N}_T$, but we remove the label t to reduce notation complexity. 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, $E^{\text{b}} \in 2^{\mathcal{E}}$ is the set of known blocked edges and $E^{\text{u}} \in 2^{\mathcal{E}}$ is the set of known unblocked edges. At a given state $(v, E^{\text{b}}, E^{\text{u}})$, the robot selects an action defined by an outgoing edge $e \in I_v$ and a command from $\mathcal{C} = \{\text{move}, \text{sense}, \text{call } \lambda, \text{terminate}\}$ where we use λ to denote the online reactive algorithm. Formally, a policy maps the robot state space to the set of actions, $\pi : \mathcal{V} \times 2^{\mathcal{E}} \times 2^{\mathcal{E}} \to I_{\mathcal{V}} \times \mathcal{C}$. A move action (e, move) in realization G_x updates the position only if it is successful (i.e., $e \in E_x$). Both move and sense actions, for edge e, update the E^{b} if $e \notin E_x$ and E^{u} if $e \in E_x$. The terminate command is only used for a *terminal state*.

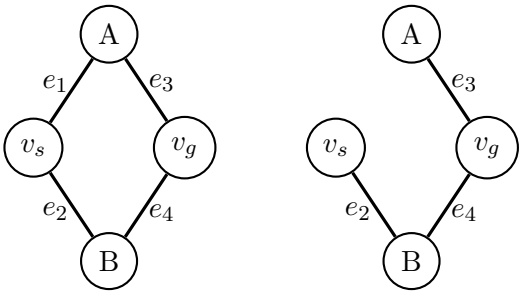

Figure 14: Example realizations G_1 (left) and G_2 (right) with undirected edges.

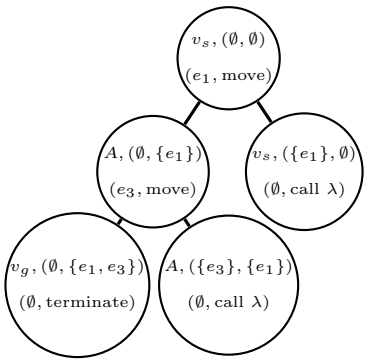

Figure 15: Example policy where nodes contain state action pairs.

Given a start and a goal $v_s, v_g \in \mathcal{V}$, a state $(v, E^{\text{b}}, E^{\text{u}})$ is said to be terminal if $v = v_g$ or the graph $\underline{\text{G}} = (\mathcal{V}, \mathcal{E} \setminus E^{\text{b}})$ has no path from v_s to v_g. The call λ command allows the robot to use another algorithm λ.

Definition 4.1 (Reactive Algorithm λ)**.** Reactive algorithm λ is an algorithm which computes next actions during a task (online) that are executed by the robot until it reaches a terminal state.

The reactive algorithm λ allows the robot to handle unexpected environments as they are encountered. That is to say, the robot will always enter a terminal state (in some finite number of actions) after it calls λ. An example reactive algorithm is D* Lite reviewed along with several other existing algorithms in Section 1.2.3. Finally, we define a complete policy.

Definition 4.2 (Complete Policy)**.** A policy π is complete if for any graph in $G_1, \ldots, G_r$ it produces a finite sequence of actions that ends in a *terminal state*.

Fig. 15 is an example complete policy π for graphs shown in Fig. 14. Notice the policy will not call λ if $G_x = G_1$ but will call λ if $G_x = G_2$.

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 (and λ if called) for some $z \in \mathbb{N}$. The cost of element a from A_{G_j} is

$$\text{cost}(a) = \begin{cases} c(e) & \text{if } a = (e, \text{move}) \\ \mu(e) & \text{if } a = (e, \text{sense}) \\ 0 & \textit{otherwise} \end{cases},$$

with total cost of A_{G_j} given by $\text{cost}(A_{G_j}) = \sum_{i=1}^{z} \text{cost}(a_i)$. Given π, the expected cost to complete a task is,

$$\mathbb{E}_X(\text{cost}(\pi)) = \sum_{j \in \mathbb{N}_r} \mathbb{P}(X = j) \text{cost}(A_{G_j}) \,. \tag{15}$$

From Lemma 3.1 in Section 3.2.3, for a complete policy to exist it is sufficient that the component containing v_s is strongly connected for each graph in $\mathcal{G}$ with non-zero probability of occurrence. In many application this holds as the robot can exit regions it can enter. An environment with a crater, that the robot cannot exit, is an example where this condition is not satisfied. We have not identify necessary conditions for the existence of a complete policy.

4.3.2 Formal Problem and Complexity

Let the current task be $t \in \mathbb{N}_T$. The robot experiences a graph from $G^{t-1} = \{G_{x_1}, \ldots, G_{x_{t-1}}\}$ or $G_{\text{new}} \in \mathcal{G} \setminus G^{t-1}$. Consider the special case where the robot is given G^{t-1} and occurrence probabilities $p_1, \ldots, p_{t-1}, p_{\text{new}}$ (that sum to one) but does not know G_{new}. We capture the probabilities of occurrence with random variable Z that takes values from $\mathbb{N}_t$ where t indicates G_{new} was drawn.

Problem 4.1 (Stage t)**.** Given directed graph G with a set of known realizations G^{t-1} and unknown realization G_{new} that have occurrence probabilities $p_1, \ldots, p_{t-1}, p_{\text{new}}$, start and goal $v_s, v_g \in \mathcal{V}$, find a travel strategy π (that travels to v_g or shows v_g is unreachable) that minimizes $\mathbb{E}_Z(\text{cost}(\pi))$.

Lemma 4.1. *If $p_{new} > 0$, Stage t is PSPACE-Hard; otherwise, Stage t is NP-Hard.*

Proof. Consider an instance of the Canadian Travellers Problem (CTP) on graph G_{CTP} discussed in Section 2.6. We can cast this as the Stage t problem by letting $p_{\text{new}} = 1$, $G = G_{\text{CTP}}$, $v_s = s$, $v_g = g$ and $\mu(e) = 0$ for all $e \in \mathcal{E}$ from G. Note $p_{\text{new}} > 0$ contains this sub problem after the robot rules out all graphs in G^{t-1} through a sequence of n actions (i.e., $p_{\text{new}} = 1$ after n actions). Consider an adversary that wishes to maximize the competative ratio from Papadimitriou & Yannakakis (1991). Let this adversary set the edge states of G_m. The competitive ratio is $\frac{\text{cost}(A_{G_{\text{new}}})}{c_{G_{\text{new}}}(v_s, v_g)}$ where $c_{G_{\text{new}}}(v_s, v_g)$ is the shortest path cost in G_{new}. For any given G_{new}, minimizing the competitive ratio reduces to minimizing the associated action cost, $\text{cost}(A_{G_{\text{new}}})$. Since $p_{\text{new}} = 1$, $\mathbb{E}_Z(\text{cost}(A_{G_{\text{new}}})) = \text{cost}(A_{G_{\text{new}}})$. At each vertex the optimal strategy will make all observations given the information has zero cost just as the original CTP. If the robot can find a strategy that generates $A_{G_{\text{new}}}$ for Stage t, it finds a strategy to generate a path $P_{G_{\text{new}}}$ for the original CTP by removing all explicit observations and replacing them with implict observations. Given the Canadian Travellers Problem is PSPACE-Hard Papadimitriou & Yannakakis (1991), Stage t with $p_{\text{new}} > 0$ is PSPACE-Hard.

Consider an instance of the Reactive Planning Problem $(G, \mathcal{S}, X, v_s, v_g)$ where $|\mathcal{S}| = t-1$ and $\Theta_v = I_v$ for all $v \in \mathcal{V}$. Let G^{t-1} be the set of edge-induced subgraph for all $E \in \mathcal{S}$, $p_i = \mathbb{P}(X = i)$ for $i \in \mathbb{N}_{t-1}$ and $p_m = 0$. Chapter 3 shows the optimal travel strategy is in form of a policy tree π, but reactive moves are not permitted. From Theorem 3.1, PWP is NP-Hard even with zero cost observations, and this implies RPP is NP-Hard for zero cost

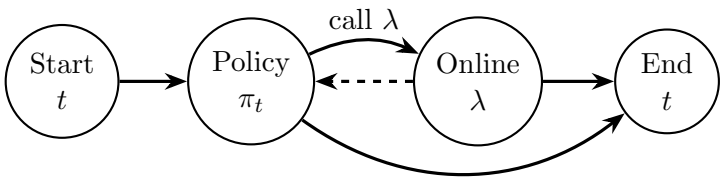

Figure 16: Solution approach that integrates an external algorithm with a policy.

observations from Theorem 3.1. Consider the case where observations cost 0. Any reactive move can be replaced with an observation by making the observation first and moving if the edge is unblocked. This is performed for no more cost than a reactive move. Therefore, given RPP is NP-Hard, Stage t with $p_m = 0$ is NP-Hard. □

Note Stage t assumes we know G^{t-1} and $p_1, \ldots, p_{t-1}, p_m$, but in practice we only know the maps collected by the robot, $\mathcal{M}_{t-1}$. We wish to consider a sequence of T stages where the robot wishes to minimize the summed cost. When considering a sequence of T tasks, note that all prior stages affect the way in which the robot completes the current task. This fact makes selecting actions in the current task dependent on the actions selected in prior tasks. In other words, 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 4.2 (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 $\sum_{i=1}^{T} \mathbb{E}_{X_i}(\text{cost}(\pi_i))$ where π_i may depend on the observations made in tasks $1, \ldots, i-1$.

We know stage 1 is PSPACE-Hard as $p_{\text{new}} = 1$; therefore, it follows naturally that the Learned Reactive Planning Problem is PSPACE-Hard.

4.3.3 Solution Approach

Consider the composite approach displayed in Fig. 16. The number of possible realizations for G_{X_t} may be exponential in the number of edges; therefore, our approach is to use π_t for a subset of environments and λ for the remaining. The robot starts by following π_t until an obstacle (or the lack there of) is encountered that π_t does not react to directly. In this case, π_t calls λ to finish the task. This satisfies the complete policy requirement, and we incrementally build such a policy as more tasks are completed. The policy, π_t, reacts to obstacles in real-time, and as such we wish to maximize the probability the robot can react to a realization G_{x_t} using only π_t. Consider Table 3 as an example of what the robot experienced during prior tasks as well as the uncertainty in the current task. Notice that the realizations and probabilities of occurrence for each task are hidden from the robot. Our

Table 3: Robot's experiences for $t-1$ tasks and uncertainty in the t^{th} task.

task	1	2	3	$\cdots$	8	9	$\cdots$	t-1	t
maps	M_1	M_2	M_3	$\cdots$	M_8	M_9	$\cdots$	M_{t-1}	M_t
realizations	G_1	G_7	G_{11}	$\cdots$	G_3	G_7	$\cdots$	G_1	$G_?$
probability	p_1	p_7	p_{11}	$\cdots$	p_3	p_7	$\cdots$	p_1	$p_?$

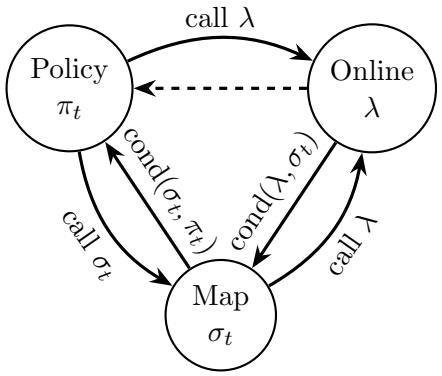

Figure 17: Solution approach with a mapping policy. Start and end nodes are unaltered but removed for simplicity.

approach to stage t is to find a policy π_t that can react to all realizations $G_{x_1}, \ldots, G_{x_{t-1}}$, and we discuss our implementation of such a policy in Section 4.4.

We show our approach to stage t works well to ensure the probability π_t calls λ is not increasing, but we also show its expected cost performance suffers based on the order in which maps are collected. To address this, we add the policy $\sigma_t : \mathcal{V} \times 2^{\mathcal{E}} \times 2^{\mathcal{E}} \to I_{\mathcal{V}} \times \mathcal{C}$ in Fig. 17 to map regions that the robot identifies may help reducing the total expected cost. We say σ_t is our mapping policy similar to π_t, but its goal is to learn the state of certain edges in G_{x_t}. We discuss building this mapping policy as well as the conditions that increase performance in Section 4.5. The policy σ_t is kept separate from π_t to ensure the robot has a complete policy independent of the results found while mapping.

Remark 4.1. The dashed edge in Fig. 16 and Fig. 17 is not considered within this chapter as returning from λ may result in a large number of states that the policy must map to actions.

4.4 Environmental Estimator

The use of command call λ is not restricted, but we wish to have some guarantees about the performance of π_t as $t \to T$. In other words, we wish to reduce the robot's dependency on reactive algorithm λ as more tasks are completed. This objective reduces time spent

Table 4: Comparison between the RPP and our approach to the LRPP.

RPP	LRPP	Description
–	t	The task number to be executed by the robot with $t \in \mathbb{N}_T$.
$\mathcal{S}$	$\mathcal{M}_{t-1}$	Maps $\mathcal{M}_{t-1}$ encode what the robot observed of $\mathcal{S}$ during $t-1$ tasks.
–	n	Action number n within a task t.
v	$v_{t,n}$	Robot's position known at all times for both the RPP and the LRPP.
Y	$\mathcal{A}_{t,n}$	Set $\mathcal{A}_{t,n}$ contains tasks whose maps agree with observations (Y is hidden).
$\mathbb{P}(e\|Y)$	$\hat{p}_e(t,n)$	Estimate $\mathbb{P}(e\|Y)$ with $\hat{p}_e(t,n)$ using $\mathcal{A}_{t,n}$ instead of Y (Y is hidden).
–	λ	Reactive algorithm λ computes actions during the task.
π	π_t	Policy π_t uses λ to react to a subset of environments.
–	σ_t	Mapping policy σ_t observes edges to optimize cost of future tasks.

planning during the robot's task as the policy can be executed in constant time.

4.4.1 Similarity to the RPP

Table 4 compares the main differences between the RPP and our approach to the LRPP. The notation is formally introduced throughout this chapter, but we include it here for reference and intuition into our approach. These similarities allows us to form properties related to legs and constructive observations presented in Chapter 3.

4.4.2 Estimator Model

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 $\mathcal{R}_{t,n} = (v_{t,n}, E^{\mathrm{b}}_{t,n}, E^{\mathrm{u}}_{t,n})$ is the robot state after the nth action. The set of prior tasks whose maps *agree* with the map, $M_{t,n}$, of G_{x_t} is $\mathcal{A}_{t,n} \equiv \{\bar{t} < t | E^{\mathrm{b}}_{t,n} \cap E^{\mathrm{u}}_{\bar{t}} = \emptyset \ \& \ E^{\mathrm{u}}_{t,n} \cap E^{\mathrm{b}}_{\bar{t}} = \emptyset\}$. Formally, this is known as map agreement.

Definition 4.3 (Map Agreement)**.** Given maps M_1 and M_2, we say M_2 agrees with M_1 if $E^{\mathrm{b}}_2 \cap E^{\mathrm{u}}_1 = \emptyset$ and $E^{\mathrm{u}}_2 \cap E^{\mathrm{b}}_1 = \emptyset$.

Consider the sequence of maps in Table 3. From inspection both M_2 and M_9 must agree as they both map G_7. In general, if two maps agree this does not imply they are maps of the same realization. This occurs as the robot does not need to know all of the environment to accomplish its task and can leave regions unmapped.

For a fixed set of edges E_{x_t} the sensing action $\gamma_v(e)$ has fixed output, yet the robot must reach v and pay $\mu(e)$ before it learns if $e \in E_{x_t}$. Consider estimating the probability an edge is blocked or unblocked. To do so, we consider all maps that agree with the robot's current knowledge. Let I be the indicator function which returns one if the input event is

true. We use the following estimator,

$$\hat{p}_e(t,n) = \begin{cases} \frac{\sum_{\bar{t}\in\mathcal{A}_{t,n}} I(e\in E_{\bar{t}}^{\mathrm{u}})}{\sum_{\bar{t}\in\mathcal{A}_{t,n}} I(e\in E_{\bar{t}})} & \exists\, j\in\mathcal{A}_{t,n} \text{ such that } e\in E_j \\ 0 & \text{otherwise} \end{cases},$$

where the condition, that at least one of the tasks have mapped e, ensures the estimator is well defined. In words, this estimator considers only the maps that know the state of e and agree with the current map $M_{t,n}$. This estimates the true probability that an edge exists given the current mapped edges, namely

$$\mathbb{P}(e\in E_{X_t}|M_{t,n}) = \sum_{i=1}^{r} \mathbb{P}(X_t = i | E_i\cap E_{t,n}^{\mathrm{b}} = \emptyset \text{ AND } E_{t,n}^{\mathrm{u}}\cup e\subseteq E_i). \tag{16}$$

This is analogous to the probability an edge exists given the set of possible environmental states, namely $\mathbb{P}(e\in E_X|Y)$ from Eq. 2 in Chapter 3.

Notice our approach is a frequentist or classical approach. The classical approach can be seen as over fitting the data and often performs poorly when there is little data as compared to the Bayesian approach Murphy (2012). This being said, we use the over fitting nature of the frequentist approach to allow aggressive behaviour at the start as well as simplified action selection.

Lemma 4.2. *Given the current map $M_{t,n}$, prior maps $\mathcal{M}_{t-1}$ and edge $e\in\mathcal{E}$, the estimate $\hat{p}_e(t,n)$ is asymptotically consistent with the number of tasks in $\mathcal{A}_{t,n}$ that have mapped e.*

Proof. Let the set of tasks that map e and agree with $M_{t,n}$ be H (i.e., $H\subseteq\mathcal{A}_{t,n}$ and $e\in E_{\bar{t}}$ for all $\bar{t}\in H$). Note H contains all tasks that contribute to our estimate. We now show the estimate is unbiased if $|H|>0$.

$$\begin{aligned}
\mathbb{E}(\hat{p}_e(t,n)) &= \frac{\sum_{\bar{t}\in H}\mathbb{E}(I(e\in E_{\bar{t}}^{\mathrm{u}}))}{|H|} \\
&= \frac{\sum_{\bar{t}\in H}\left(\mathbb{P}(e\in E_{X_{\bar{t}}}|M_{t,n}) + 0\cdot\mathbb{P}(e\notin E_{X_{\bar{t}}}|M_{t,n})\right)}{|H|} \\
&= \frac{|H|\mathbb{P}(e\in E_X|M_{t,n})}{|H|} \text{ (using fact } X_{\bar{t}} \text{ is i.i.d.)} \\
\mathbb{E}(\hat{p}_e(t,n)) &= \mathbb{P}(e\in E_X|M_{t,n}).
\end{aligned}$$

Next, we show the variance of $\hat{p}_e(t,n)$ tends to zero as $|H| \to \infty$.

$$\begin{aligned}
\mathbb{V}(\hat{p}_e(t,n)) &= \frac{\mathbb{V}\big(\sum_{\bar{t}\in H} I(e \in E^{\mathrm{u}}_{\bar{t}})\big)}{|H|^2} \\
&= \frac{\sum_{\bar{t}\in H} \mathbb{V}\big(I(e \in E^{\mathrm{u}}_{\bar{t}})\big)}{|H|^2} \text{ (using independence of each } X_{\bar{t}}) \\
&= \frac{1^2\mathbb{P}(e \in E_X) + 0^2\mathbb{P}(e \notin E_X) - p_e^2}{|H|} \\
\mathbb{V}(\hat{p}_e(t,n)) &= \frac{p_e - p_e^2}{|H|} \to 0 \text{ as } |H| \to \infty.
\end{aligned}$$

□

We use the estimated probability an edge e exists in a realization E_{x_t} of E_{X_t} to form a model of the sensing function,

$$\hat{\gamma}_{t,n}(e) = \begin{cases} \text{blocked} & \hat{p}_e(t,n) = 0 \\ \text{unblocked} & \hat{p}_e(t,n) = 1 \\ \text{unknown} & \text{otherwise} \end{cases}.$$

This model is being updated during the current task. In other words, after an action is performed the robot updates $\mathcal{A}_{t,n}$; thus, it updates the model of the sensing function $\hat{\gamma}_{t,n}(e)$. Given our goal is to react in constant time, the following sections provide methods to update $\hat{\gamma}_{t,n}(e)$ only after the task is completed.

4.4.3 Estimator Based Policies Π

Consider the class of complete policies, Π, that obey the following procedure.

Definition 4.4 (Complete Policy Class Π)**.** For a complete policy π to have membership to the class Π, π must obey the following restrictions:

1. Action (e, sense) only used if $\hat{\gamma}_{t,n}(e) = \text{unknown}$.
2. Action (e, move) only used if $\hat{\gamma}_{t,n}(e) = \text{unblocked}$.
3. Action $(\emptyset, \text{call } \lambda)$ only used if $|\mathcal{A}_{t,n}| = 0$.
4. Action $(\emptyset, \text{terminate})$ only used if robot state is terminal.

Remark 4.2 (Relation to the RPP). A policy in the class Π produces properties similar to policies for the RPP from Chapter 3. Consider a move action (e, move) as the n^{th} action taken by the robot. The model of the sensing function must return unblocked if the robot wishes to use this action. If this action is successful, the set of tasks which agree with $M_{t,n}$,

namely $\mathcal{A}_{t,n}$, remain unaltered (i.e., $\mathcal{A}_{t,n-1} = \mathcal{A}_{t,n}$). This occurs because the agreeing maps that contain e must all be unblocked or $\hat{p}_e(t,n) < 1$. This is the fundamental requirement for all move actions of a leg from Chapter 3. In the case where the move fails, the robot calls λ. Similarly, the restriction on a sense action, namely $\hat{\gamma}_{t,n}(e) =$ unknown, only allows observations if there are two possible outcomes. This is the requirement of a constructive observation from Chapter 3.

Given a fixed λ, the robot cannot identify the sequence of actions the reactive algorithm will output without knowing G_{x_t}. In order to evaluate the cost of π from Π, we estimate Eq. 15 as it is not computable without X or the actions generated by λ. Formally,

$$\text{cost}(T \text{ tasks}) = \sum_{t=1}^{T} \text{cost}_{\pi_t}(M_t).$$

Notice that π_t is formed with all $\mathcal{M}_{t-1}$ and as such the actions during prior tasks affect the ability to minimize cost of future tasks. This creates implicit dependencies of all prior maps and policies on the current task.

The restrictions on Π allow the robot to function assuming G_{x_t} has been mapped in $\mathcal{M}_{t-1}$, but, if the robot learns this assumption is not true, it can rely on λ to finish the task. We will use this framework to show the dependence on λ is not increasing as more maps are collected.

4.4.4 Properties of Π

We show the probability, that a policy from the class Π calls λ, does not increase as more tasks are completed, but also show it tends to zero as the number of tasks tends to infinity.

Property 1: The class Π guarantees that the probability of any π from Π calling reactive algorithm λ is not increasing as tasks are completed. In effect, the robot has learned how it should react to the environmental realizations it has experienced.

Lemma 4.3. *Given any policy π in the class Π, the probability that λ is called decreases monotonically with the number of completed tasks.*

Proof. To show this, we first prove that command (call λ) does not occur if G_{x_t} has been mapped in $\mathcal{M}_{t-1}$. In other words, the policy is complete in $\mathcal{M}_{t-1}$. Let G_{x_t}'s prior map be $(E_i^{\text{b}}, E_i^{\text{u}})$ for some $i \in \mathcal{A}_{t,n}$. By construction of a map, we know the recorded edge states cannot disagree with the current realized edge states, namely $E_i^{\text{b}} \subseteq \mathcal{E} \setminus E_{x_t}$ and $E_i^{\text{u}} \subseteq E_{x_t}$. Call λ only occurs when $|\mathcal{A}_{t,n}| = 0$. We know any final map, M_t, of G_{x_t} must agree with M_i because they both observed the same realization. This implies no sense or move action can remove i from $\mathcal{A}_{t,n}$. Therefore, the set of maps that agree will never be empty and call λ will not be used. Given independent draws from the same pmf, the probability any realization G_x has been mapped in $\mathcal{M}_a$ can never be less than $\mathcal{M}_b$ for tasks $a \geq b$ because all maps in $\mathcal{M}_b$ exist in $\mathcal{M}_a$. □

Figure 18: Overview of the incremental update.

Property 2: Given G_{x_t} has not been mapped in $\mathcal{M}_{t-1}$, a policy in Π may not require the reactive algorithm λ. To show this, consider two graphs in $\mathcal{G}$ which have all the same edges except for some $e \in \mathcal{E}$ (e.g., say G_1 and G_2). Let G_1 be mapped in $\mathcal{M}_{t-1}$ as M_i but let G_2 not be mapped. If M_i does not contain the state of e, the set of tasks which agree, $\mathcal{A}_{t,n}$, will always include task i for $G_{x_t} = G_2$; implying, $|\mathcal{A}_{t,n}| \neq 0$ (i.e., the procedure for Π will not allow the call λ command).

Property 3: Given a policy π_t from class Π, the robot can estimate the probability it will call λ during task t. Given the number of calls to λ during the prior $t-1$ tasks is $n_{\lambda|t-1}$, we can estimate the probability λ will be called next task by,

$$\hat{p}_{\lambda|t} = \frac{n_{\lambda|t-1}}{t-1}. \tag{17}$$

Lemma 4.4. *Given π_t from Π, the estimate, $\hat{p}_{\lambda|t}$ is asymptotically unbiased and tends to zero as $t \to \infty$.*

Proof. Consider a task $\bar{t}$ such that each graph with non-zero probability of occurrence has been mapped in $\mathcal{M}_{\bar{t}}$. Given $\pi_{\bar{t}}$ is in the class Π, no sense or move action can remove all maps that agree (i.e., $|\mathcal{A}_{\bar{t}+1,n}| \geq 1$ for all n). Therefore, all future tasks $t > \bar{t}$ will not call λ implying $n_{\lambda|\bar{t}} = n_{\lambda|t}$ and $\lim_{t\to\infty} \frac{n_{\lambda|t-1}}{t-1} = 0$. We know after $\bar{t}$ the true probability a new graph will occur is 0 (i.e., $p_{\lambda|t} = 0$); thus, the estimator is asymptotically unbiased and tends to 0 as $t \to \infty$. □

4.5 Policy Generation

Our approach to the Learned Reactive Planning Problem is to update the current policy based on the realizations of the environment it has encountered. We focus on the update shown in Fig 18 that occurs between tasks and builds a policy as more tasks are complete. A more rigorous notion of our approach can be seen in Algorithm 4. This algorithm shows how the robot's knowledge interacts with task completion. Within this section, we discuss implementation options for Line 9 and Line 10 of Algorithm 4 and extend the tree policy for $\theta_v = I_v$ from Chapter 3 by adding the call λ command.

Algorithm 4: Sequential Task Completion

1 let π_1 return call λ command;
2 **for** $t = 1, \ldots, T$ **do**
3 initialize state $\mathcal{R}_{t,n} = (v_s, \emptyset, \emptyset)$ for $n = 0$;
4 **do**
5 execute $\pi_t(\mathcal{R}_{t,n})$; `// if λ is called wait until it terminates`
6 update $\mathcal{R}_{t,n}$ increment n;
7 **while** $\mathcal{R}_{t,n}$ not terminal;
8 collect $E^{\mathrm{b}}_{t,n}$ and $E^{\mathrm{u}}_{t,n}$ from $\mathcal{R}_{t,n}$;
9 update $\mathcal{M}_t$ to include $(E^{\mathrm{b}}_t, E^{\mathrm{u}}_t)$; `// the n is dropped as the task is over`
10 build π_{t+1} given $\mathcal{M}_t$ and π_t; `// incrementally building a policy`

4.5.1 Policy Structure

A policy can be efficiently encoded into a binary tree $\pi = (N, O)$. The nodes N of the tree are given by tuples (M, a) for map $M = (E^{\mathrm{b}}, E^{\mathrm{u}})$ before action $a = (e, c)$, and the edges are outcomes O connecting from node n' to node n'' if and only if $E^{\mathrm{b}''} = E^{\mathrm{b}'} \cup e$ or $E^{\mathrm{u}''} = E^{\mathrm{u}'} \cup e$. Therefore, nodes with call λ or terminate commands are leaf nodes given they do not directly update the robot state. The nodes with move or sense commands have two outgoing edges corresponding to $e \in E_{x_t}$ and $e \notin E_{x_t}$.

4.5.2 Incremental Update

Consider a policy π_t, for task t, in the class Π from Section 4.4. The robot uses π_t while it traverses its unknown environmental realization G_{x_t}. The call λ action occurs if $t = 1$, a move node n_{move} for action (e, move) fails or the robot senses an edge unblocked when it is trying to show there does not exist a path to v_g. If λ was called during task t, the policy π_t would not be in the class Π after task t because π_t calls λ when the new map M_t agrees with the robot state (i.e., it does not satisfy requirement 3 of Π). We ensure π_{t+1} remains in the class Π by implementing Line 10 of Algorithm 4 with the following incremental update.

1. Call *simplePolicy* to create π_{new}
2. If $t = 1$ let $\pi_2 = \pi_{\mathrm{new}}$
3. Else If n_{move} failed, add (e, sense) before n_{move} attaching π_{new} to outcome $\gamma_v(e) =$ blocked
4. Else attach π_{new} to outcome $\gamma_v(e) =$ unblocked

In words, the incremental update extends π_t into π_{t+1} to react to graph G_{x_t} if it is encountered again in a future task. Note the solution to Line 8 of *simplePolicy* is NP-Hard. If the time between tasks is short, sub-optimal solutions may be required.

Algorithm 5: simplePolicy

Input: v, v_g, M_t
Output: π_{new}

```
1  G = (V_t, E_t^u) with V_t endpoints of E_t^u;
2  find shortest path P from v to g in G;
3  if P exists then
4  |  let π_new encode P;
5  else
6  |  let E^b ⊆ E_t^b s.t. (V, E \ E^b) does not contain a path from v to v_g;
7  |  let V^b = {v ∈ V | (v, u) ∈ E};
8  |  find the shortest path P^b from v that visits all v' ∈ V^b;
9  |  let π_new encode P^b;
10 |  add (e, sense) ∀e ∈ E^b;
11 add required call λ nodes to π_new;
```

To give an example of the incremental update, consider the very simple undirected graphs displayed in Fig. 14. Suppose the robot's policy, π_t, is displayed in Fig 15, and let $G_{x_t} = G_2$ (i.e., the right graph from Fig. 14). Action (e_1, move) fails because edge $e_1 \notin E_{x_t}$. The robot calls λ which terminates with map $M_t = (\{e_1\}, \{e_2, e_4\})$. Following *simplePolicy*, the shortest path is v_s, B, v_g. To encode this as a policy the robot must perform two move actions, namely (e_2, move) and (e_4, move). Finally, we add call λ nodes to the failed result of these moves as defined in Line 11. The resulting policy π_{t+1} can be seen in Fig. 19.

Remark 4.3 (Faulty Sensors). If the sensor is faulty (incorrect observation outcome), the policy should always call λ in the failed move case. For example, node (e_1, move) from Fig. 19 could call λ when $e \in E^{\text{b}}_{t,n}$ even though the sensor indicated otherwise.

Lemma 4.5. *A policy π_T built via the incremental update can be encoded into $\mathcal{O}(T|\mathcal{E}|)$ nodes.*

Proof. Policy π_T requires the most space if λ was called after every task $t \in \mathbb{N}_T$. In the case v_g can be reached, π_{new} will contain at most $2|\mathcal{V}| - 2$ move and call λ actions given a path of lowest cost between any two vertices in G can always be found without visiting the same vertex twice (positive traversal cost). If v_g is not reachable, the robot must sense all edges between the connected component containing v_s and the component containing v_g. π_{new} will need to sense at most $|\mathcal{E}|$ edges and move $|\mathcal{V}| - 2$ times. If a sense action returns unblocked or a move action fails, the robot will call λ. We can assume G is connected or a smaller G exists for the same problem; therefore $|\mathcal{E}| + 1 \geq |\mathcal{V}|$. π_{new} is added T times and has $\mathcal{O}(|\mathcal{E}|)$ nodes. □

The main benefit of this algorithm is its ability to update the policy using only the latest map. In other words, this incremental update removes Line 9 and the space required

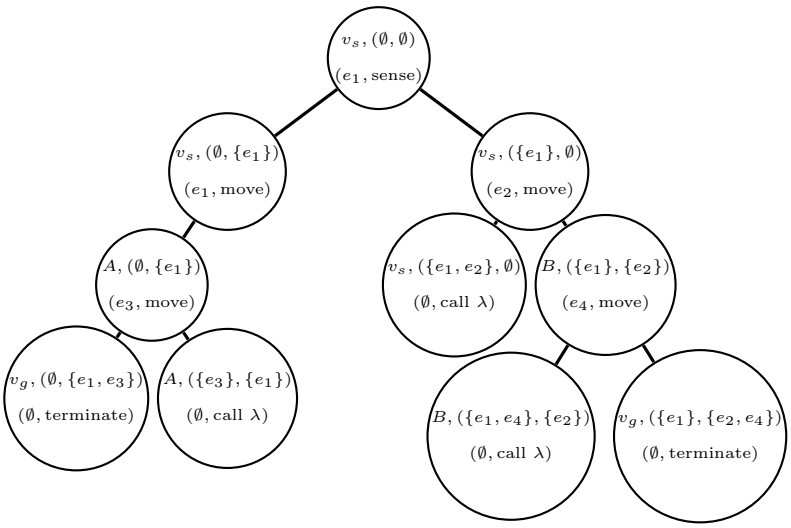

Figure 19: Example policy after incremental update. Nodes show state action pairs.

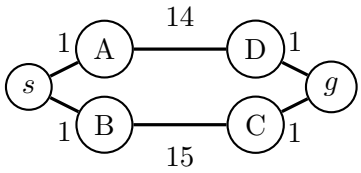

Figure 20: Example G with travel costs shown on edges.

for map storage from Algorithm 4 as the update uses map M_t and π_t to form π_{t+1}. On the other hand, the lack of map memory creates an issue with optimization across maps. That is to say, the solution is strongly tied to the order of $G_{x_1}, \ldots, G_{x_t}$. We use the following example to illustrate this weakness.

Example 4.1. Consider Fig. 20 where all sensing actions cost 0.5. Let $\mathbb{P}(X = 1) = \mathbb{P}(X = 2) = 0.5$ with G_1 only missing edge (D, g) and G_2 only missing edge (s, B). Let $G_{x_1} = G_2$ and $G_{x_2} = G_1$. Suppose π_1 (i.e., call λ) ends at state $R = (g, \emptyset, \{(s, A), (A, D), (D, g)\})$. The incremental update selects π_2 to traverse path s, A, D, g. Given $G_{x_2} = G_1$, π_2 will call λ at vertex D resulting in π_3 which must back track through S when $G_{x_t} = G_1$. Suppose the order is changed to $G_{x_1} = G_1$ and $G_{x_2} = G_2$. Using the same λ, the robot completes task 1, and the incremental update set π_2 to traverse s, B, C, g. In task 2, (s, B) is blocked, but the incremental update now creates π_3 that does not back track.

4.5.3 Map Memory Filter

To address the map order dependency displayed in Ex. 4.1, consider a map $M_{\bar{t}}$ of $G_{x_{\bar{t}}}$. After task $\bar{t}$, any edges not mapped in $M_{\bar{t}}$ restricts optimization for a future task t. If $G_{x_t} = G_{x_{\bar{t}}}$, the robot cannot show this unless all edges were mapped in $M_{\bar{t}}$ and M_t. In essence, the robot is learning experiences of the environment instead of learning the environment. Suppose the robot could selectively store or overwrite maps to facilitate learning more of the environment. With this in mind, consider the function *mapFilter* as an implementation option for Line 9 in Algorithm 4 with the set of maps, $\mathcal{M}_t^{\text{filtered}}$, stored and $\mathcal{M}_0^{\text{filtered}} = \emptyset$. We say $\mathcal{M}_t^{\text{filtered}}$ is the set of *super maps* from $\mathcal{M}_t$ with smallest cardinality (i.e., no repeated super maps).

Definition 4.5 (Super Maps)**.** Given $\mathcal{M}_t$, 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 $E_i^{\text{b}} \subseteq E_j^{\text{b}}$ and $E_i^{\text{u}} \subseteq E_j^{\text{u}}$.

Algorithm 6: mapFilter

Input: M_t, $\mathcal{M}_{t-1}^{\text{filtered}}$
Output: $\mathcal{M}_t^{\text{filtered}}$
for each $(E^{\text{b}}, E^{\text{u}}) \in \mathcal{M}_{t-1}^{\text{filtered}}$ **do**
  if $E_t^{\text{u}} \subseteq E^{\text{u}}$ AND $E_t^{\text{b}} \subseteq E^{\text{b}}$ **then**
    return $\mathcal{M}_{t-1}^{\text{filtered}}$;
  if $E^{\text{u}} \subset E_t^{\text{u}}$ AND $E^{\text{b}} \subset E_t^{\text{b}}$ **then**
    return $(\mathcal{M}_{t-1}^{\text{filtered}} \cup M_t) \setminus (E^{\text{b}}, E^{\text{u}})$;
return $\mathcal{M}_{t-1}^{\text{filtered}} \cup M_t$;

Note, we call M_i the sub map. Using this method of storage, we can simplify the expected cost estimate Eq. 15 to,

$$\text{cost}(\pi_t) = \sum_{M_j \in \mathcal{M}_t^{\text{filtered}}} \left(\frac{n_j}{t}\right) \text{cost}_{\pi_t}(M_j)\,, \tag{18}$$

where n_j is the number of maps in $\mathcal{M}_t$ that agree with super map M_j. The filtered map storage can be thought of as overwriting all sub maps with their super map. We show that this filter requires no more than q maps where q is the total number of realizations with non-zero probability of occurrence.

Lemma 4.6. *$|\mathcal{M}_t^{filtered}| \leq \min(T, q)$ for all $t \leq T$ where q is the number of graphs in $\mathcal{G}$ with non-zero probability of occurrence.*

Proof. Given q possible environments with $T > q$, we must show there are at most q unique super maps. Suppose by contradiction, $|\mathcal{M}_t^{\text{filtered}}| > q$ for any $t \in T$. This implies there exists at least one super map (say $M_{\text{extra}} \in \mathcal{M}_t^{\text{filtered}}$) more than the number of environments.

Given $\mathcal{M}_t^{\text{filtered}}$ is minimal we know that each pair $M \neq M' \in \mathcal{M}_t^{\text{filtered}}$ satisfies $E^{\text{b}} \cap E^{\text{u}'} \neq \emptyset$ or $E^{\text{u}} \cap E^{\text{b}'} \neq \emptyset$. In words, no pair of maps can agree or they would not all be unique super maps. Therefore, M_{extra} must satisfy $E^{\text{b}}_{\text{extra}} \cap E_i \neq \emptyset$ and $E^{\text{u}}_{\text{extra}} \cap (\mathcal{E} \setminus E_i) \neq \emptyset$ for all $(V_i, E_i) \in \mathcal{G}$ with non-zero probability. This is a contradiction as $\mathcal{M}_{\text{extra}}$ does not satisfy the definition of a map; thus, $|\mathcal{M}_t^{\text{filtered}}| \leq q$.

Consider the case where $T \leq q$. Given the robot only completes T tasks, it experiences T maps. Therefore, the robot cannot collect more unique super maps than tasks, (i.e., $|\mathcal{M}_t^{\text{filtered}}| \leq T$). □

If the robot executes a policy built with the incremental update from the prior section, it will only create new super maps rather than update existing super maps. To see this, consider the case where λ is not called, which implies $|\mathcal{A}_{t,n}| > 0$. The edges mapped in M_t must agree with some map in $\mathcal{M}_{t-1}$; and by the creation of π_{new}, the policy only uses actions on edges with known state. Therefore, M_t cannot be a super map. In the case $|\mathcal{A}_{t,n}| = 0$, all maps disagree with M_t, and a new super map must be created.

4.5.4 Observation Swapping using Mapping Policy σ_t

Swapping Observations: The goal of this work is to minimize the expected cost of the sequence of policies. Consider an observation inserted by the incremental update. The observation is selected to avoid calling λ. In this section, we discuss replacing this observation with another observation as an *observation swap* in order to lower the expected cost of the policy.

Let the task $t-1$ be completed by the robot where the incremental update creates π_t with expected cost estimated by Eq. 18. The robot wishes to reduce the expected cost of its actions for the remaining tasks. After each task, we review every observation made by the robot during the task including any just added by the incremental update. The n^{th} action, B = (e_{old}, sense), can only be swapped with another observation, B' = (e_{new}, sense), if B' partitions $\mathcal{A}_{t,n-1}$ the same way B does. This is analogous to the way a constructive observation partitions Y from Chapter 3. Fig. 21 displays the swapping process. The sequence of move actions from A to B' uses the environmental estimator defined by $\mathcal{A}_{t,n-1}$ where as B' to C and D use the respective partition $\mathcal{A}_{t,n}$. The robot will swap B' for B if the expected cost, using Eq. 18, of the actions removed is greater than the expected cost of the actions added.

Mapping in Order to Swap: The robot may wish to swap a sense action with another of an edge e that is currently unknown in $M_j \in \mathcal{M}_t^{\text{filtered}}$ (i.e., $e \notin E_j$ for known edges of map M_j). That is to say, if this edge was in a certain state, it would qualify to be swapped and reduce the expected cost. We estimate the probability e partitions $\mathcal{A}_{t,n-1}$ correctly, $\hat{p}^e_{\text{part}}$, to be $\hat{p}_e(t,n)$ if e must be unblocked or $1 - \hat{p}_e(t,n)$ if e must be blocked. Note we

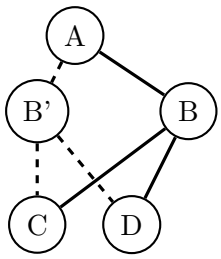

Figure 21: Example node swap. Edges are sequences of move actions connecting observations. Dashed edges represent possible swap.

calculate $\hat{p}_e(t, n)$ assuming M_j mapped e to be in the desired state. We set $\hat{p}^e_{\text{part}} = 0$ for an edge that cannot partition $\mathcal{A}_{t,n-1}$ correctly. If the robot senses this edge, it updates the existing super map M_j.

To update an existing super map, we use the mapping policy $\sigma_t : \mathcal{V} \times 2^{\mathcal{E}} \times 2^{\mathcal{E}} \rightarrow I_{\mathcal{V}} \times \mathcal{C}$ discussed in Section 4.3.3. To use σ_t, π_t contains a node with the call σ command. The condition, $\text{cond}(\sigma_t, \pi_t)$ from Fig. 17, is met when the robot has returned to the vertex v where σ_t was first called. From this point π_t continues to function as if σ_t had not been called. When the robot completes the task, the observations made with σ_t will update the super map that agrees with map M_t.

Remark 4.4 (Online mapping). Given Lemma 3.1, the robot can always return to v and may use the call λ command if required. Note the condition, $\text{cond}(\lambda, \sigma_t)$ from Fig. 17, is met when the robot has returned to v.

This work considers mapping a single edge e by adding σ^e to σ_t in order to make σ_{t+1} for the super map $M_j \in \mathcal{M}_t^{\text{filtered}}$ of task t. Algorithm 7 defines the selection process of σ^e. Line 11 is used to estimate the value of mapping edge e in order to swap it with another observation, where t_{remain} is the number of tasks remaining after σ^e updates M_j. To estimate t_{remain} at time t, we model the event of M_j agreeing with a future $M_{t'}$ for $t' > t$ as a geometric distribution with event probability $\frac{n_j}{t}$. Thus, we expect this event to occur $\frac{t}{n_j}$ tasks from now and render,

$$t_{\text{remain}} = \max\left(T - t - \frac{t}{n_j}, 0\right).$$

Consider Algorithm 7 which selects an edge to be mapped that is expected to reduce the cost of future tasks. This algorithm evaluates every edge which has not been mapped in M_j and estimates the expected savings if it was mapped in M_j. Algorithm 7 assess each vertex v_i from the action sequence A (i.e., v_i is the robot's position before action a_i) and finds the lowest expected cost σ^e to sense e and return to v_i in Line 7. Then it selects the overall lowest expected cost σ^e in Line 8. Line 10 evaluates the expected savings of swapping any current observation with an observation of this new edge. Finally, Line 11

Algorithm 7: Mapping Selection

Input: π_{t+1}, M_j, $\mathcal{M}_t^{\text{filtered}}$, T
Output: σ^e

Let A be the sequence of action from π_{t+1} for M_j;
Let e_{new} be no edge with savings $\texttt{save} = 0$;
for each $e \notin E_j$ **do**
    Set $\mathcal{A}_{K_{t|0}} = \mathbb{N}_{|\mathcal{M}_t^{\text{filtered}}|}$;
    Set $\texttt{opt}_\sigma = \infty$;
    for i from 1 to $|A|$ **do**
        Find $\hat{\gamma}_{t|i-1}$ and use it to create σ^e from v_i;
        Let $\texttt{opt}_\sigma = \min\left(\texttt{opt}_\sigma, \frac{|\mathcal{A}_{t,i-1}|\text{cost}(\sigma^e)}{t}\right)$;
        if $c = \text{sense}$ **then**
            Let $opt_\pi = \text{cost}(\pi_{t+1}) - \text{cost}(\pi^e_{t+1})$;
            if $\frac{\texttt{opt}_\pi t_{\text{remain}}}{\hat{p}^e_{\text{part}}} - \frac{opt_\sigma t}{n_j} < \texttt{save}$ **then**
                Update $\texttt{save}$ and set $e_{\text{new}} = e$;
return $\sigma^{e_{\text{new}}}$ if e_{new} was set;

identifies that the expected savings is greater than the expected cost of mapping the edge, and Algorithm 7 maps the edge with the highest expected savings a swap can produce.

4.6 Simulations

Consider the flexible factory presented in Section 3.6.1. This environment has 48 vertices, 146 edges and 34561 realizations. In order to test the LRPP, we use this factory model but hide the prior data from the robot. The robot is provided the graph but no information on obstacles or their correlations. We consider four separate start and goal configurations, namely $S \to A$, $S \to B$, $A \to S$ and $B \to S$. Tasks returning to S have half the traversal cost as the robot is unloaded. Given the robot does not know if any edge is obstructed, the cost of any observation is set to 0.3 (i.e., $\mu(e) = 0.3$ for all $e \in \mathcal{E}$).

In order to test the performance of this work, we compare the incremental update (with and without swapping observations) against using only the reactive algorithm for the same sequence of random draws. The reactive algorithm, λ, used for our testing is as follows:

1. Find the shortest path from v to v_g and follow it.
2. If edge on path unknown, observe it and go to 1.

The results are outlined in Table 5. The savings column is found by taking the percent difference between the cost of only using λ and the cost of the incremental update (with

Table 5: Simulation results for the flexible factory with hidden obstacle correlations.

		Inc. Update			Inc. Update and Swap Obs.		
Task	T	% saved	$\hat{p}_{\pi_T}$	p_{π_T}	% saved	$\hat{p}_{\pi_T}$	p_{π_T}
S→A	500	3.9	0.88	0.97	**4.2**	0.87	0.98
	1000	**3.0**	0.93	0.99	2.8	0.93	0.99
	2000	5.1	0.92	0.99	**6.0**	0.91	1
	4000	**3.5**	0.98	1	3.4	0.98	1
S→B	500	**-4.3**	0.86	0.93	-4.5	0.86	0.93
	1000	0.1	0.92	0.95	**5.2**	0.90	0.95
	2000	3.1	0.94	0.97	**3.4**	0.94	0.97
	4000	3.3	0.96	0.98	**3.9**	0.95	0.98
A→S	500	**9.8**	0.86	0.94	9.1	0.86	0.93
	1000	**21.8**	0.96	1	20.4	0.95	1
	2000	25.6	0.98	1	**30.4**	0.97	1
	4000	20.5	0.99	1	**27.9**	0.99	1
B→S	500	9.5	0.88	0.99	**9.9**	0.87	0.98
	1000	5.7	0.93	0.99	**11.4**	0.91	0.99
	2000	6.4	0.96	1	**10.6**	0.96	1
	4000	-1.8	0.98	1	**7.0**	0.97	1

and without swapping observations). For the shorter tasks (i.e., $T = 500$ and $T = 1000$), observation swapping has similar performance to not swapping observations. This occurs because the condition to map an edge considers how many tasks are left and is more cautious as $t \to T$. For these cases, we found that, after the robot had identified new maps, mapping was expected to cost more than the savings from the few remaining tasks. For the longer tasks (i.e., $T = 2000$ and $T = 4000$) with observation swapping, the robot frequently identified regions to be mapped with possible savings for the remaining tasks. Note without observation swapping, the robot's performance varied greatly (e.g., B→S with $T = 2000$ versus with $T = 4000$). This is linked to the ordering issue discussed in Ex. 4.1.

Using the hidden data, we calculated the probability the robot will not call λ with policy π_T, $p_{\pi_T} = 1 - p_{\lambda|T}$. This is compared against the estimated probability the robot would not call λ with policy π_T, $\hat{p}_{\pi_T} = 1 - \hat{p}_{\lambda|T}$, in Table 5. In several cases, the policy found will not call λ after T tasks. Note this occurs due to Property 2 of the environmental estimator as the robot cannot map all 34561 realizations that have non-zero probability of occurrence (i.e., $T \leq 4000 < 34561$). The estimated probability was always higher than the true probability after T tasks for both algorithms. This cannot be proven true in general, but we found empirically that it often occurs.

Chapter 5
Conclusions and Future Work

This work discusses reaction to uncertainty in the environment. We show the importance of allowing the robot to terminate if there does not exist a path to the desired goal. When the environment model is known, we present a policy that generates trajectories in real-time response to environmental observations made by the robot, which we describe as active sensing. When the environment model is hidden, we present a system where the probability this system will react in real-time is monotonically increasing as more tasks are completed.

5.1 Closing Thoughts

This thesis presents two methods for reactive planning based on prior knowledge of environmental uncertainty and learning this uncertainty as the robot functions. Given prior knowledge, we present a sub-optimal, efficient algorithm that produces a policy which guides the robot to the goal or shows the goal cannot be reached. This algorithm also produces a lower bound on the optimal expected cost of any policy. We compare the policy generated by this algorithm to existing research and an optimal policy.

Given the robot has no prior environmental knowledge, we present an incremental update that builds a policy as tasks are completed. This policy calls an external reactive algorithm to handle unexpected environments, and each incremental update incorporates the experience into the policy. The probability this policy calls the reactive algorithm is shown to be monotonically decreasing as more tasks are completed. Finally, we test the incremental update against the reactive algorithm alone.

5.2 Future Work in Known Environment Model

The primary task within this thesis is to find a strategy to reach the goal or show it cannot be reached. Consider the task for a robot to dispose some waste within an office building. There are often several waste baskets which the robot can visit. In other words, there are environments where a task can be completed at different locations or configurations. A useful extension of our work would consider multiple goal vertices; thus, the robot can judge the best goal given its current environmental understanding. This differs from multiple policies as it may not be clear when to select a different goal until the robot has reached a level of environmental awareness.

The second extension would provide a set of possible start vertices. Consider a robot deployed at some location. If the deployment is not similar each time, there may be discrepancies in performance based on localization within the environment. Let a probability

mass function be defined over the start vertices that quantifies the probability the robot will start at a given vertex. The robot is now tasked to reach the goal or show it cannot be reached. We have not considered the complexity of this problem or any policy changes required.

5.3 Future Work in Hidden Environment Model

Within our hidden environment model, the realizations, experienced by the robot, are independently and identically distributed based on an unknown probability mass function. This models some environments, but it is often desirable to consider environments with temporal correlations. For example, an office may be more crowded and dynamic at noon than at midnight. The map memory filter presented in this work can be extended to consider this correlation. Another alternative is to develop several policies; the robot then selects the policy it wishes to execute for a given task based on the maps collected to date.

In future work, we wish to extend the observation model from single edges to consider edge subsets. As presented in Chapter 3, many sensor models can be captured by this extension, but observing more than one edge no longer guarantees a failed move will provide the same information as an observation action. The incremental update handles this difference, but we speculate finding a swap observation that partitions the maps similarly is a computationally hard problem dependent on the number of edges in the observation.

On a practical note, we are interested in creating different map memory filters with multiple objectives. Consider a fix memory constraint (i.e., a maximum number of maps much smaller than T). Within this constraint, minimize the probability $p_{\lambda|T}$ as well as the cost incurred by the robot. This added constraint makes selecting which maps to store more complicated as both how many maps occurred and when may be critical.

Consider the hidden probability mass function. The estimated probability an edge exists sums the probability that a drawn graph contains that edge. An interesting sub problem is to estimate the original hidden probability mass function. The exact probability of a given graph occurring may not be possible due to Property 2 from Section 4.4.4, but we consider the probability a subset of these graphs occur.

References

Andreatta, G. & Romeo, L. (1988), 'Stochastic shortest paths with recourse', *Networks* **18**(3), 193–204.

Aoude, G. S., Luders, B. D., Joseph, J. M., Roy, N. & How, J. P. (2013), 'Probabilistically safe motion planning to avoid dynamic obstacles with uncertain motion patterns', *Autonomous Robots* **35**(1), 51–76.

Argall, B. D., Chernova, S., Veloso, M. & Browning, B. (2009), 'A survey of robot learning from demonstration', *Robotics and autonomous systems* **57**(5), 469–483.

Arslan, O. & Koditschek, D. E. (2016), Sensor-based reactive navigation in unknown convex sphere worlds, *in* 'submitted to) the 12th International Workshop on the Algoritmic Foundations of Robotics (WAFR)'.

Bai, H., Cai, S., Ye, N., Hsu, D. & Lee, W. S. (2015), Intention-aware online pomdp planning for autonomous driving in a crowd, *in* 'Robotics and Automation (ICRA), 2015 IEEE International Conference on', IEEE, pp. 454–460.

Bai, H., Hsu, D. & Lee, W. S. (2014), 'Integrated perception and planning in the continuous space: A POMDP approach', *The International Journal of Robotics Research* **33**(9), 1288–1302.

Best, G., Faigl, J. & Fitch, R. (2016), Multi-robot path planning for budgeted active perception with self-organising maps, *in* 'IEEE/RSJ International Conference on Intelligent Robots and Systems', pp. 3164–3171.

Bhattacharya, S., Ghrist, R. & Kumar, V. (2015), 'Persistent homology for path planning in uncertain environments', *IEEE Transactions on Robotics* **31**(3), 578–590.

Binney, J. & Sukhatme, G. S. (2012), Branch and bound for informative path planning., *in* 'IEEE International Conference on Robotics and Automation', pp. 2147–2154.

Bnaya, Z., Felner, A. & Shimony, S. E. (2009), Canadian traveler problem with remote sensing, *in* 'International Joint Conference on Artificial Intelligence', pp. 437–442.

Bulitko, V. & Lee, G. (2006), 'Learning in real-time search: A unifying framework', *Journal of Artificial Intelligence Research* **25**, 119–157.

Chakaravarthy, V. T., Pandit, V., Roy, S., Awasthi, P. & Mohania, M. (2007), Decision trees for entity identification: approximation algorithms and hardness results, *in* 'Proceedings ACM Symp. on Principles of Database Systems', ACM, pp. 53–62.

Charrow, B., Kumar, V. & Michael, N. (2014), 'Approximate representations for multi-robot control policies that maximize mutual information', *Autonomous Robots* **37**(4), 383–400.

Chen, M., Frazzoli, E., Hsu, D. & Lee, W. S. (2016), POMDP-lite for robust robot planning under uncertainty, *in* 'IEEE International Conference on Robotics and Automation', pp. 5427–5433.

Cover, T. M. & Thomas, J. A. (2012), *Elements of information theory*, John Wiley & Sons.

Dames, P. M., Schwager, M., Rus, D. & Kumar, V. (2016), 'Active magnetic anomaly detection using multiple micro aerial vehicles', *IEEE Robotics and Automation Letters* **1**(1), 153–160.

Dames, P., Schwager, M., Kumar, V. & Rus, D. (2012), A decentralized control policy for adaptive information gathering in hazardous environments, *in* 'IEEE Conference on Decision and Control', pp. 2807–2813.

Dann, C. & Brunskill, E. (2015), Sample complexity of episodic fixed-horizon reinforcement learning, *in* 'Advances in Neural Information Processing Systems', pp. 2818–2826.

Dijkstra, E. W. (1959), 'A note on two problems in connexion with graphs', *Numerische mathematik* **1**(1), 269–271.

Du Toit, N. E. & Burdick, J. W. (2012), 'Robot motion planning in dynamic, uncertain environments', *IEEE Transactions on Robotics* **28**(1), 101–115.

Dydek, Z., Annaswamy, A. & Lavretsky, E. (2010), Combined/composite adaptive control of a quadrotor uav in the presence of actuator uncertainty, *in* 'AIAA Guidance, Navigation, and Control Conference', p. 7575.

Elfes, A. (1989), 'Using occupancy grids for mobile robot perception and navigation', *Computer* **22**(6), 46–57.

Gray, A., Gao, Y., Lin, T., Hedrick, J. K., Tseng, H. E. & Borrelli, F. (2012), Predictive control for agile semi-autonomous ground vehicles using motion primitives, *in* 'American Control Conference (ACC), 2012', IEEE, pp. 4239–4244.

Hart, P. E., Nilsson, N. J. & Raphael, B. (1968), 'A formal basis for the heuristic determination of minimum cost paths', *IEEE transactions on Systems Science and Cybernetics* **4**(2), 100–107.

Hollinger, G. A., Englot, B., Hover, F. S., Mitra, U. & Sukhatme, G. S. (2012), 'Active planning for underwater inspection and the benefit of adaptivity', *The International Journal of Robotics Research* pp. 3–18.

Hollinger, G. A. & Sukhatme, G. S. (2014), 'Sampling-based robotic information gathering algorithms', *The International Journal of Robotics Research* **33**(9), 1271–1287.

Issac, P. & Campbell, A. M. (2015), 'Shortest path problem with arc failure scenarios', *EURO Journal on Transportation and Logistics* pp. 1–25.

Javdani, S., Chen, Y., Karbasi, A., Krause, A., Bagnell, D. & Srinivasa, S. S. (2014), Near optimal bayesian active learning for decision making., *in* 'International Conference on Artificial Intelligence and Statistics', pp. 430–438.

Julian, B. J., Karaman, S. & Rus, D. (2014), 'On mutual information-based control of range sensing robots for mapping applications', *The International Journal of Robotics Research* **33**(10), 1375–1392.

Kaelbling, L. P. & Lozano-Pérez, T. (2013), 'Integrated task and motion planning in belief space', *The International Journal of Robotics Research* **32**(9-10), 1194–1227.

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

Kober, J., Bagnell, J. A. & Peters, J. (2013), 'Reinforcement learning in robotics: A survey', *The International Journal of Robotics Research* **32**(11), 1238–1274.

Koenig, S. (2001), 'Agent-centered search', *AI Magazine* **22**(4), 109.

Koenig, S. & Likhachev, M. (2005), 'Fast replanning for navigation in unknown terrain', *IEEE Transactions on Robotics* **21**(3), 354–363.

Koenig, S., Likhachev, M. & Furcy, D. (2004), 'Lifelong planning a', *Artificial Intelligence* **155**(1-2), 93–146.

Koenig, S., Tovey, C. & Smirnov, Y. (2003), 'Performance bounds for planning in unknown terrain', *Artificial Intelligence* **147**(1-2), 253–279.

Konar, A., Chakraborty, I. G., Singh, S. J., Jain, L. C. & Nagar, A. K. (2013), 'A deterministic improved q-learning for path planning of a mobile robot', *IEEE Transactions on Systems, Man, and Cybernetics: Systems* **43**(5), 1141–1153.

Krajnik, T., Fentanes, J. P., Cielniak, G., Dondrup, C. & Duckett, T. (2014), Spectral analysis for long-term robotic mapping, *in* 'Robotics and Automation (ICRA), 2014 IEEE International Conference on', IEEE, pp. 3706–3711.

Kretzschmar, H., Spies, M., Sprunk, C. & Burgard, W. (2016), 'Socially compliant mobile robot navigation via inverse reinforcement learning', *The International Journal of Robotics Research* **35**(11), 1289–1307.

Kucner, T., Saarinen, J., Magnusson, M. & Lilienthal, A. J. (2013), Conditional transition maps: Learning motion patterns in dynamic environments, *in* 'Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on', IEEE, pp. 1196–1201.

Kuwata, Y., Teo, J., Fiore, G., Karaman, S., Frazzoli, E. & How, J. P. (2009), 'Real-time motion planning with applications to autonomous urban driving', *IEEE Transactions on Control Systems Technology* **17**(5), 1105–1118.

LaValle, S. M. (2006), *Planning algorithms*, Cambridge university press.

Lim, Z. W., Hsu, D. & Lee, W. S. (2015), 'Adaptive informative path planning in metric spaces', *The International Journal of Robotics Research* **35**(5), 585–598.

Liu, B., Zhang, F. & Qu, X. (2015), 'A method for improving the pose accuracy of a robot manipulator based on multi-sensor combined measurement and data fusion', *Sensors* **15**(4), 7933–7952.

MacDonald, R. A. & Smith, S. L. (2016), Reactive motion planning in uncertain environments via mutual information policies, *in* 'International Workshop on the Algorithmic Foundations of Robotics (WAFR)'.

Mahoney, A. W., Bruns, T. L., Swaney, P. J. & Webster, R. J. (2016), On the inseparable nature of sensor selection, sensor placement, and state estimation for continuum robots or where to put your sensors and how to use them, *in* 'Robotics and Automation (ICRA), 2016 IEEE International Conference on', IEEE, pp. 4472–4478.

Majumdar, A. & Tedrake, R. (2013), Robust online motion planning with regions of finite time invariance, *in* 'Algorithmic Foundations of Robotics X', Springer, pp. 543–558.

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

Murphy, K. P. (2012), *Machine learning: a probabilistic perspective*, MIT press.

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

Papadimitriou, C. H. & Tsitsiklis, J. N. (1987), 'The complexity of markov decision processes', *Mathematics of operations research* **12**(3), 441–450.

Papadimitriou, C. H. & Yannakakis, M. (1991), 'Shortest paths without a map', *Theoretical Computer Science* **84**(1), 127–150.

Park, J.-J., Kim, J.-H. & Song, J.-B. (2007), 'Path planning for a robot manipulator based on probabilistic roadmap and reinforcement learning', *International Journal of Control, Automation, and Systems* **5**(6), 674–680.

Pastor, P., Kalakrishnan, M., Chitta, S., Theodorou, E. & Schaal, S. (2011), Skill learning and task outcome prediction for manipulation, *in* 'Robotics and Automation (ICRA), 2011 IEEE International Conference on', IEEE, pp. 3828–3834.

Polychronopoulos, G. H. & Tsitsiklis, J. N. (1996), 'Stochastic shortest path problems with recourse', *Networks* **27**(2), 133–143.

Ross, S., Pineau, J., Paquet, S. & Chaib-Draa, B. (2008), 'Online planning algorithms for pomdps', *Journal of Artificial Intelligence Research* **32**, 663–704.

Sipser, M. (2006), *Introduction to the Theory of Computation*, Vol. 2, Thomson Course Technology Boston.

Souza, A. & Goncalves, L. M. G. (2016), 'Occupancy-elevation grid: an alternative approach for robotic mapping and navigation', *Robotica* **34**, 2592–2609.

Stavrou, D., Eliades, D. G., Panayiotou, C. G. & Polycarpou, M. (2013), A path correction module for two-wheeled service robots under actuator faults, *in* 'Control & Automation (MED), 2013 21st Mediterranean Conference on', IEEE, pp. 1119–1126.

Suhov, Y., Stuhl, I., Sekeh, S. Y. & Kelbert, M. (2015), 'Basic inequalities for weighted entropies', *Aequationes mathematicae* pp. 1–32.

Van Den Berg, J., Patil, S. & Alterovitz, R. (2012), 'Motion planning under uncertainty using iterative local optimization in belief space', *The International Journal of Robotics Research* **31**(11), 1263–1278.

Wang, R., Veloso, M. & Seshan, S. (2016), Active sensing data collection with autonomous mobile robots, *in* 'IEEE International Conference on Robotics and Automation', pp. 2583–2588.

Yu, J., Schwager, M. & Rus, D. (2014), Correlated orienteering problem and its application to informative path planning for persistent monitoring tasks, *in* 'IEEE International Conference on Intelligent Robots and Systems', pp. 342–349.

Yu, W.-C., Yang, C.-Y., Su, K.-H. & Tu, Y.-H. (2014), Dynamic path planning under randomly distributed obstacle environment, *in* 'Automatic Control Conference (CACS), 2014 CACS International', IEEE, pp. 138–143.