Towards Efficient Planning for Navigation using Global Information in Large and Uncertain Environments

by

Martina Stadler Kurtz

S.B., Massachusetts Institute of Technology, 2018
S.M., Massachusetts Institute of Technology, 2020

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2024

Authored by:	Martina Stadler Kurtz Department of Aeronautics and Astronautics August 8, 2024
Certified by:	Nicholas Roy Professor of Aeronautics and Astronautics, Thesis Supervisor
Certified by:	Leslie Pack Kaelbling Panasonic Professor of Computer Science and Engineering
Certified by:	Seth Hutchinson Professor and KUKA Chair for Robotics, Georgia Institute of Technology
Accepted by:	Jonathan P. How Richard Cockburn Maclaurin Professor in Aeronautics and Astronautics Chair, Graduate Program Committee

Towards Efficient Planning for Navigation using Global Information in Large and Uncertain Environments

by

Martina Stadler Kurtz

Submitted to the Department of Aeronautics and Astronautics
on August 8, 2024 in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

ABSTRACT

We would like to enable a team of robots to navigate quickly and efficiently in large and uncertain outdoor environments. We hypothesize that in such environments, global, uncertainty-aware information is necessary to enable high-quality planning. However, most existing systems do not model or plan using global, uncertainty-aware information. For example, many planners assume access to complete global information in the form of full environment maps, or they assume that locally good planning decisions under uncertainty will result in globally good planning outcomes. To enable the use of global information for planning in large and uncertain environments, we must develop models that concisely represent key navigation features of the environment, and build planners that are capable of reasoning efficiently about global information. In this thesis, we design models and planners that use global information in large and uncertain environments to increase the efficiency and quality of planning for navigation.

We present four contributions towards using global information for efficient navigation. First, we propose a high-level planning representation that can be learned from previous plans considered in the environment and used online during hierarchical, multi-query robot navigation. Second, we propose a planner for collaborative multiagent navigation in an uncertain environment; the approach uses macro-actions and value function approximations to maintain computational tractability. Third, we develop a robust hierarchical planning system to enable the deployment of the collaborative multiagent planner on a real-world team navigating in a structured, uncertain outdoor environment. Finally, we develop a method for learning uncertainty-aware, single agent value function-based approximations from graph data to increase the efficiency of the collaborative multiagent planner.

Thesis supervisor: Nicholas Roy
Title: Professor of Aeronautics and Astronautics

To my parents,

for guiding me to the gates of this journey,

and to Andrew,

for seeing me through it, every step of the way.

Acknowledgments

I would like to thank Nick Roy, who has advised me in some capacity for the past nine years. In that time, Nick showed me how to navigate MIT, welcomed me into the robotics community, and shaped me into the researcher that I am today. I am especially grateful to Nick for teaching me how to convey a research story.

I would like to thank the other members of my committee, Seth Hutchinson and Leslie Pack Kaelbling. Seth's contributions throughout our collaborations changed the way I think about multiagent planning, and his practical approach to robotics drove me to complete and publish the experiments reported in Chapter 5. Leslie's sharp and insightful questions were integral in helping me solidify the overarching message of this thesis. I would also like to thank my thesis readers, Harish Ravichandar and Andrew Messing, for their engagement during our collaborations, and for their support of my work.

I would like to thank all the members of the RRG for their insights and guidance. I am especially grateful to Katherine Liu, Jacopo Banfi, Sam Prentice, and Yasmin Veys, who directly contributed to the ideas and results in this thesis. Apart from being my first research mentor, Katherine's support during COVID-19 was a life raft in more ways than I can write. Jacopo introduced me to the Canadian Traveller's Problem (CTP). Sam fixed every unfixable real robot problem that I encountered (and on more than one occasion, caused), and he paved the way for the experiments in Chapter 5. Yasmin's thoughtful conversations about the CTP changed the way that I view the model, its benefits, and its limitations. I would also like to thank all of my DCIST collaborators for providing high-level research guidance and practical support during field deployments.

I would like to thank all of my family, both Stadler and Kurtz, for their endless support throughout this journey. Thank you Dad, for introducing me to this transformative community. Thank you Mom, for being a fantastic scientist and role model, and for listening to me on my toughest days. Thank you Berna, for being available to talk, regardless of your time zone. Thank you JJ, for all of the dinners and hikes, and for helping me keep things in perspective.

Finally, thank you Andrew, for joining me on this adventure, and for believing in me when I did not believe in myself. Your unconditional love and support is the greatest privilege of my life.

This thesis was supported by the Army Research Laboratory under Cooperative Agreement Number W911NF-17-2-0181 and by the Singapore Defence Science and Technology Agency (DSTA)-MIT Master Research Agreement. The MIT SuperCloud and Lincoln Laboratory Supercomputing Center provided HPC resources that contributed to the results reported in this thesis. Their support is gratefully acknowledged.

Contents

Title page **1**

Abstract **3**

Acknowledgments **7**

List of Figures **13**

List of Tables **15**

1 Introduction **17**
1.1 Online Hierarchical Model Generation for Multi-Query Planning 22
1.2 Models and Approximations for Collaborative Multiagent Planning under Uncertainty 23
1.3 Real-World Collaborative Multiagent Planning under Uncertainty 24
1.4 Learned Value Functions for Planning under Uncertainty 24
1.5 Statement of Contributions 25

2 Background **27**
2.1 Overview 27
2.2 The Continuous Single-agent Optimal Planning Problem 27
2.3 The Discrete Single-agent Optimal Planning Problem 28
2.3.1 Modeling the Problem 29
2.3.2 Solving the Problem 32
2.3.3 Multi-Query Planning 33
2.4 The Hierarchical Discrete Single-agent Optimal Planning Problem 34
2.4.1 Multi-Resolution Planning 35
2.4.2 Task-Based Planning 37
2.4.3 Correct Abstractions 39
2.4.4 Learned Abstractions 39
2.5 The Discrete Single-agent Optimal Planning Problem under Uncertainty . . 40
2.5.1 Learning Properties of Unknown Spaces 42
2.5.2 Modeling Cost Uncertainty on Environment Graphs/Roadmaps . . . 43
2.6 Modeling Traversability Uncertainty on Graphs: The Canadian Traveller's Problem 44

2.7 Partially Observable Markov Decision Processes (POMDPs) . . . 47
2.7.1 Types of POMDPs . . . 48
2.8 The CTP as a POMDP . . . 48
2.9 The Discrete Optimal Multiagent Planning Problem . . . 50
2.10 The Discrete Optimal Multiagent Planning Problem Under Uncertainty . . . 53

3 Online High-Level Model Estimation for Efficient Hierarchical Robot Navigation 57
3.1 Motivation . . . 57
3.2 Approach . . . 60
3.2.1 Models for Hierarchical Planning . . . 60
3.2.2 The High-Level Action Space . . . 61
3.2.3 Primitive Subplans as Instances of High-Level Actions . . . 63
3.2.4 The High-Level Functions . . . 64
3.2.5 Online High-Level Function Updating as Recursive Estimation . . . 65
3.2.6 Hierarchical Planning using High-Level Functions Estimated Online . . . 67
3.3 Experiments . . . 71
3.3.1 Experimental Setup . . . 71
3.3.2 Simulation Evaluation Results . . . 72
3.4 Conclusion . . . 81

4 Approximating the Value of Collaborative Team Actions for Efficient Multiagent Navigation in Uncertain Graphs 83
4.1 Motivation . . . 84
4.2 Approach . . . 87
4.2.1 The Canadian Traveller's Problem . . . 87
4.2.2 Multiagent Planning on Unknown Graphs . . . 88
4.2.3 Macro-actions for Information-gathering Teams . . . 89
4.2.4 Policy Class Augmentation for Planning . . . 91
4.2.5 Approximations for Policy Selection with One Stochastic Agent . . . 92
4.2.6 Using Problem Structure to Reduce the Number of Candidate Policy Classes . . . 98
4.2.7 Planning using Δ and Multiagent Macro-Actions on CTPs . . . 98
4.3 Experiments . . . 99
4.3.1 Toy Environment . . . 101
4.3.2 Islands Environment . . . 103
4.4 Conclusion . . . 106

5 Real-World Deployment of a Hierarchical Uncertainty-Aware Collaborative Multiagent Planning System 107
5.1 Motivation . . . 108
5.2 Approach . . . 111
5.2.1 Real-World Planning System Overview . . . 111
5.2.2 Level 1: Collaborative Multiagent Planning on Abstract Graphs . . . 112

5.2.3 Level 2: Executing Macro-actions for Collaborative Planning 114
5.2.4 Level 3: Primitive Actions 118
5.3 Deployments and Results 120
5.3.1 Deployment Results: Camp Buckner 121
5.3.2 Deployment Results: Magazine Beach 124
5.4 Discussion 134
5.5 Conclusion 135

6 Learned Value Function Approximations for Efficient Collaborative Multiagent Planning on Uncertain Graphs 137
6.1 Motivation 138
6.2 Problem Statement 139
6.3 Learned Approximations for Multiagent Planning Under Uncertainty 140
6.3.1 Graph Neural Networks for Stochastic Motion Planning 141
6.3.2 Neural Network Structure and Optimization 142
6.3.3 Using Value Function Approximations for Online Planning 144
6.4 Experiments 144
6.4.1 Clusters Environment Setup 145
6.4.2 Dataset Generation 145
6.4.3 Clusters Environment Results 145
6.4.4 Islands Experiment Setup 148
6.4.5 Islands Environment Results 148
6.5 Conclusion 150

7 Conclusion 151
7.1 Future Work: Hierarchical Model Estimation for Multi-Query Planning . . . 153
7.2 Future Work: Collaborative Multiagent Planning under Uncertainty 154
7.3 Future Work: Real-World Deployments of Collaborative Multiagent Planning under Uncertainty 155
7.4 Future Work: Learned Value Function Approximations on Uncertain Graphs 156

List of Figures

1.1 Example deployment environments. 18
1.2 Motivating example of outdoor team navigation. 19

3.1 Online high-level model estimation for efficient hierarchical robot navigation. 58
3.2 Overview of the online high-level model estimation approach. 61
3.3 Example regions, actions, primitive subplans, and functions in a simulated outdoor domain. 62
3.4 Planning in graphs with known and unknown edge costs. 70
3.5 Example overhead images and occupancy maps from the simulated outdoor environment. 71
3.6 Planning metrics for the hierarchical planners over time. 73
3.7 High-level cost functions for the 'right' action and planning solutions for the hierarchical planners over time. 74
3.8 Scatter plots of steady state planning results. 76
3.9 Example *Medium-Risk* solutions during late planning. 77
3.10 Plan similarity between high-level and primitive plans over time. 78
3.11 Visualizations of the *Average* high-level functions for the 'right' action over multiple planning queries. 79
3.12 Visualizations of the *Medium-Risk* high-level functions for the 'right' action over multiple planning queries. 79
3.13 Visualizations of the *High-Risk* high-level functions for the 'right' action over multiple planning queries. 80

4.1 Motivating example of collaborative team planning under uncertainty. 85
4.2 Visual plan comparison in the toy graph. 101
4.3 Visual plan comparison in the islands domain. 105

5.1 Real-world deployment of a collaborative multiagent planner. 109
5.2 Overview of the hierarchical planning system. 111
5.3 An example real-world collaborative team plan. 113
5.4 An example real-world macro-action. 114
5.5 Examples of macro-action interrupts and their impacts on planning. 117
5.6 Graph-based planning results, Camp Buckner. 122
5.7 Real-world executed trajectories, Camp Buckner. 123
5.8 The Magazine Beach environment. 125

5.9 Non-collaborative graph-based planning results, Magazine Beach, Scenario 1 126
5.10 Collaborative graph-based planning results, Magazine Beach, Scenario 1. . . 127
5.11 Executed trajectories, Magazine Beach, Scenario 1. 128
5.12 Scatter plots of timing results, Magazine Beach, Scenario 1. 129
5.13 Non-collaborative graph-based planning results, Magazine Beach, Scenario 2 130
5.14 Collaborative graph-based planning results, Magazine Beach, Scenario 2. . . 131
5.15 Executed trajectories, Magazine Beach, Scenario 2. 132
5.16 Scatter plots of timing results, Magazine Beach, Scenario 2. 133

6.1 GNN structure. 143
6.2 Learning results in the cluster dataset. 146
6.3 Learning results in the islands dataset. 149

List of Tables

3.1 Planning performance for the hierarchical planners during different stages of planning. . . . 75

4.1 Toy graph results. . . . 102
4.2 Islands domain results. . . . 104

5.1 Comparison of collaborative and baseline plans, Camp Buckner. . . . 124
5.2 Summary of failed trials, Camp Buckner. . . . 124
5.3 Aggregate timing results, Magazine Beach, Scenario 1. . . . 128
5.4 Aggregate timing results, Scenario 2. . . . 133

6.1 Planning results in the cluster domain. . . . 147
6.2 Planning results in the islands domain. . . . 149

Chapter 1

Introduction

We would like to enable a team of robots to navigate quickly and efficiently in large, uncertain outdoor environments, such as those in Fig. 1.1. Efficient outdoor navigation is a core capability for many robotic teams, including those completing search and rescue and scouting missions. However, many traditional planning approaches were not designed for use in long length scale environments, where reasoning over all global structure can be computationally infeasible, or for use in unknown environments, where global structure is not available. For robots planning long length scale navigation tasks, or for teams of robots navigating in uncertain environments, traditional planning approaches that do not take into account the global structure of the environment can lead to low-quality plans.

For example, consider the air vehicle and ground vehicle team navigating in the unmapped outdoor environment in Fig. 1.2-a, given only goal coordinates and an overhead image. The ground vehicle is tasked with navigating from start SG to goal GG and the air vehicle is tasked with navigating from start SA to goal GA while minimizing team makespan. If the vehicles were to adopt the common robotics assumption that all unmapped regions of the environment are unoccupied, the ground vehicle would likely attempt to navigate to the goal through the untraversable trees and buildings in the center of the image, resulting in poor team planning performance or even team planning failure.

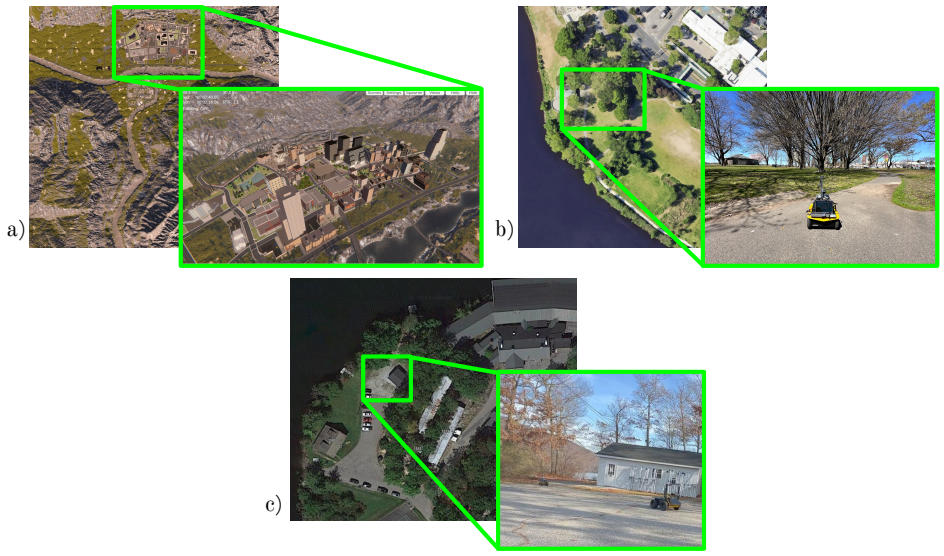


Figure 1.1: **Example deployment environments.** (a) A kilometer-scale simulated city and outpost. (b) An outdoor park. (c) A semi-structured outdoor environment.

One method for improving team planning performance in this example is to generate models of global environment information and use the models for robot planning. For example, given the overhead image, a human can generate a high-level idea of the navigation properties of the environment, such as difficult navigation in the dense forest surrounding the buildings in Fig. 1.2-a, and efficient navigation on the roads. The human can then use the global information to identify low expected cost candidate paths for the agents. For example, the air vehicle can likely navigate directly to the goal using roads, and there are two possible low cost paths for the ground vehicle — one path through the sparse trees at the top of the image (Fig. 1.2-b), and one path that avoids all areas with trees and uses only roads to navigate to the goal (Fig. 1.2-d).

However, enabling robots to *autonomously* generate and plan in global environment models is non-trivial. Good global models for planning must be both *representative* and *sparse* — they must capture key navigation features of environments, such as traversable roads and impassable dense forests, but they also must be efficient for global planning. Unfortunately, these two objectives are at odds with one another; capturing navigation features of environments requires an agent to model many specific environment details, yet planning is most

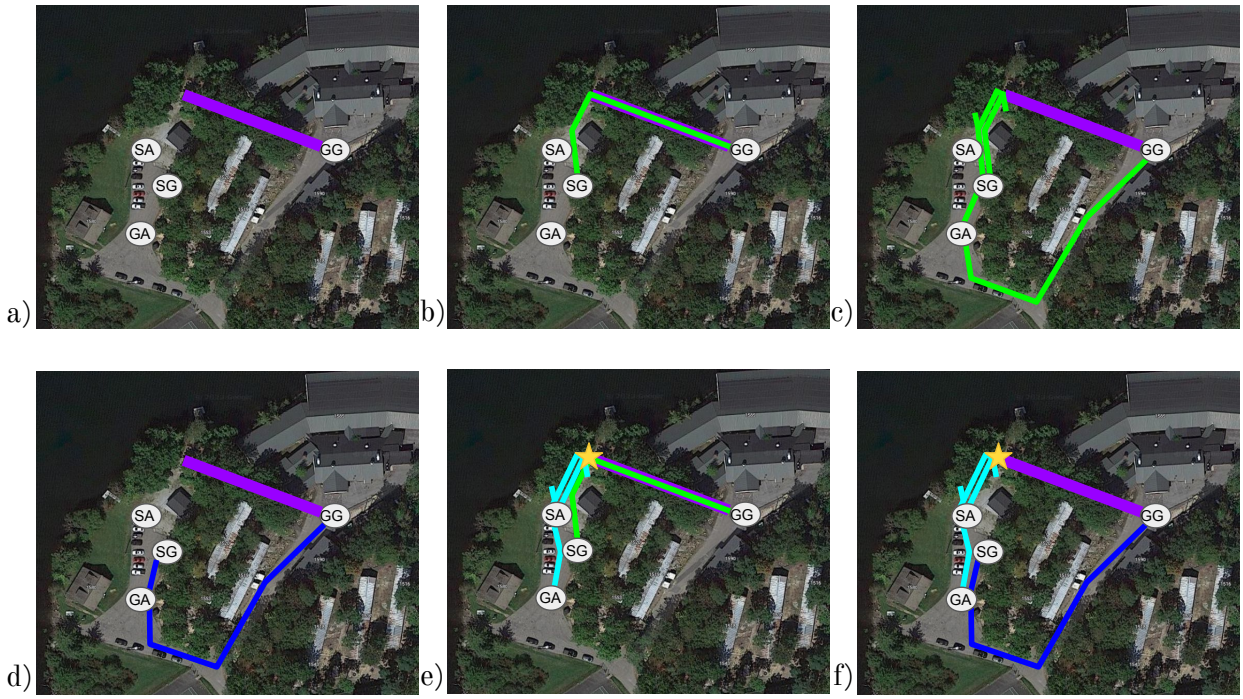


Figure 1.2: **Motivating example.** Consider an air vehicle/ground vehicle team navigating in a real-world structured outdoor environment. (a) From only an overhead image, it is not obvious if the forested region is traversable (purple). A ground vehicle starting at SG and navigating to GG with no additional information has to select between an unknown trajectory that could lead to either a short path (b, green) or a long detour (c, green) and a medium-length, known trajectory (d, dark blue), either of which can be suboptimal given the unknown traversability of the forest. By taking a small diversion to the gold star, the air vehicle (with start SA and goal GA) can sense and share the traversability of the forest with the ground vehicle (e, light blue), allowing the ground vehicle to select the best available path to GG for every forest state (e, green and f, dark blue), reducing the total expected team navigation cost.

efficient when the environment models are small and sparse. The challenge of generating sparse, representative models is even greater in uncertain environments, as uncertainty often increases model complexity.

Hierarchical and abstract environment models are a promising method for incorporating long length scale and uncertainty-aware planning structure into robot planning decisions. High-level models have the potential to represent key navigation information in a compact way that is compatible with efficient robot planning. For example, high-level models can be used to summarize key environment information when it is computationally intractable to represent all environment information at robot execution time, or they can be used to model environment information that is beyond the horizon of traditional local planning models. For example, in Fig. 1.2, an abstract model based on environment semantics may represent the untraversable water at the top left of the environment coarsely, as it is not important to maintain detailed information about the untraversable lake. At the same time, the abstract model may represent the roads in finer detail, as detailed road information is likely to be important for generating good plans in the environment. Similarly, an abstract model based on metric task information may include information about roads near the ground vehicle's goal that are not otherwise captured by the vehicle's local map or planner, enabling better decision-making about paths that avoid the central cluster of buildings and trees.

Unfortunately, it is not obvious how to automatically generate the sparse, representative models that are most useful for efficient navigation planning in large, possibly unknown outdoor environments, especially prior to planning. While it is intuitive that outdoor environmental structure, like the presence or absence of a path or a forest, or the existence of a traversable bridge in an impacted city, can change an agent's ability to navigate efficiently, the usefulness of environmental information for planning is dependent on the agent or team task, environmental uncertainty, and the history of prior (team) actions, among others. However, most models must decide the trade off between the completeness of information they retain about the environment and the efficiency of the planning process prior to

having any task- or timestep-relevant context, which can significantly impact the usefulness of specific environmental information. Therefore, it is fundamentally challenging to generate sparse, representative models of large and uncertain environments that capture important environment features for general navigation tasks.

Additionally, the usefulness of an abstract representation is dependent on the planner that uses the model. In this work, we define *high-quality* planners as planners that are able to efficiently find low cost, risk-aware solutions to planning problems using abstract models. High-quality planners must be capable of using the abstract models to quickly reduce the number of plans or policies considered during online planning, either by directly identifying good plans and policies, or by identifying uncertain information that must be resolved to enable low-cost, risk-aware planning. In uncertain environments, the planners must be able to use the abstract models to balance between exploring regions of the environment that may lead to low cost plans, and exploiting known low cost plans in the environment. For example, in Fig. 1.2, a planner must be able to decide between navigating to sense the traversability of the sparse forest in the top right of the image, which results in either navigating directly to the goal (Fig. 1.2-b) or a long detour (Fig. 1.2-c), or navigating to the goal via a conservative path on roads that is longer but known to be traversable (Fig. 1.2-d). Multiagent planners under uncertainty also have the flexibility to use different agents to reduce team environmental uncertainty. For example, in Fig. 1.2-e-f, a multiagent planner that sends the air vehicle to the gold star to sense the traversability of the forest and share the information with the ground vehicle will reduce expected team makespan.

Unfortunately, generating informed planners that are capable of taking advantage of abstract environment models is also challenging. Planners that use global environmental information for decision-making must be capable of reasoning about planning at long time horizons, as global navigation information that is important for planning often exists at long length scales and time horizons. Additionally, these planners must be capable of trading off between exploration and exploitation in uncertain environments, which is known to be a

challenging problem. Finally, multiagent planners must be able to plan with delayed rewards, as the benefit of a teammate's action may not be realized until later in an agent's plan.

In this thesis, we develop models and planners that are updated online based on past and current planning information to more efficiently represent and reason about long length scale, uncertain global information for navigation in large, outdoor environments. We use a combination of prior domain knowledge and online observations to generate sparse, representative abstractions for planning. We explicitly model and reason about environmental uncertainty in our planning algorithms, enabling agents to make risk-aware planning decisions, and enabling multiagent teams to collaborate in the face of uncertainty, all while maintaining computational tractability. Finally, we develop methods to increase the efficiency of our planners, enabling their deployment for complex teams navigating in large and uncertain outdoor environments.

1.1 Online Hierarchical Model Generation for Multi-Query Planning

In our first contribution, we propose an abstract model for planning that can be learned from previous plans considered in the environment and used online during hierarchical, multi-query robot navigation. While existing hierarchical planners can plan efficiently using static, coarse, often hand-selected abstractions that are sparse and representative in select environments, these planners often fail to capture the lossy nature of plan abstraction for planning in complex environments. Rather than generating an abstraction tailored to a specific environment, we select a simple environmental decomposition to generate an abstract model of the planning problem, and explicitly represent uncertainty caused by imperfect abstraction in the abstract model. Then, we recognize that we can treat previous planning results in the environment as noisy measurements of the navigation properties of the abstract model. We use the measurements to update the navigation properties of the abstract model on-

line during multi-query planning using recursive estimation. Finally, we use a hierarchical, uncertainty-aware planner that is capable of using the high-level model to generate plans over the course of a multi-query planning trial.

We tested our approach in a simulated long length scale outdoor environment using standard and risk-aware planning schemes, and demonstrated up to an 86% decrease in the number of nodes expanded and a 66% decrease in wallclock time as compared to a baseline A* planner while finding plans that were only 2-10% more expensive.

1.2 Models and Approximations for Collaborative Multiagent Planning under Uncertainty

In our second contribution, we develop a planner for collaborative multiagent navigation in an uncertain environment, represented as a stochastic graph, where some edges may be untraversable. We model collaborative team policies on stochastic graphs using macro-actions, where each macro-action for a given agent can consist of a sequence of movements, sensing actions, and actions of waiting to receive information from other agents. To reduce the number of macro-actions considered during planning, we generate optimistic approximations of the value functions of candidate future team states, and then restrict the planning domain to a small policy class that consists of only macro-actions that are likely to lead to high-reward future team states. Then, we optimize team plans over the small policy class.

We demonstrated our approach in simulated toy graph and island road network domains, and showed that our approach found up to 21.0% and 18.1% better plans, respectively, than other planning methods that did not reason about collaborative multiagent plans. Additionally, in the toy environment, we showed that our approach found plans up to 43.0% more quickly than a collaborative planner that did not use approximations to generate a small policy class for collaborative team planning.

1.3 Real-World Collaborative Multiagent Planning under Uncertainty

In our third contribution, we deployed the collaborative multiagent planner for a real-world, two-agents team navigating in one structured, uncertain outdoor environment and in one semi-structured, uncertain outdoor environment. While enabling tractable planning requires the development of abstract models that can represent complex, high-quality plans, such models often abstract away information needed to generate directly executable plans for real-world agents in real-world environments, as planning in such detail, especially in the presence of real-world uncertainty, would be computationally intractable. We describe the deployment of a planning system that used a hierarchy of planners to execute collaborative multiagent navigation tasks in real-world, unknown environments. By developing a planning system that was robust to failures at every level of the planning hierarchy, we enabled the team to complete collaborative navigation tasks, even in the presence of imperfect planning abstractions and real-world uncertainty.

1.4 Learned Value Functions for Planning under Uncertainty

In our fourth contribution, we introduce a method for learning uncertainty-aware, single-agent value functions from graph data to enable more efficient collaborative multiagent planning under uncertainty. While information-gathering actions, such as sensing the traversability of a route, can have a large impact on aggregate team performance for collaborative agents navigating through an unknown environment, planning over the full space of joint team actions is generally computationally intractable. The approach described in Section 1.2 developed multi-agent macro-actions for navigation on stochastic graphs and a value func-

tion approximation-based planner that enabled us to find high-quality collaborative plans for small environments and teams, but the approach was computationally limited, especially as the size of the environment and team increased. We propose learning a value function approximator to increase the efficiency of the uncertainty-aware collaborative multiagent planner. Specifically, we develop a method for learning belief-dependent value function approximations on uncertain graphs.

We demonstrate the approximations in the collaborative planner in toy graph and island road network domains, and demonstrate a 36.6% improvement in planning time as compared to a Monte Carlo (MC) rollout-based approximation approach for team planning in the toy domain, while finding plans of similar quality.

1.5 Statement of Contributions

The contributions of this thesis are as follows:

1. A high-level planning representation that can be learned from previous plans considered in the environment and used online during hierarchical, multi-query robot navigation (Chapter 3).

2. A planner for collaborative multiagent navigation in an uncertain environment that is represented as a stochastic graph with edges with uncertain traversabilities (Chapter 4).

3. A robust hierarchical planning system for deploying the planner from Chapter 4 on a real-world team navigating in a structured, uncertain outdoor environment, and the results of two real-world deployments of the system (Chapter 5).

4. A method for learning uncertainty-aware, single-agent value functions from graph data to enable more efficient collaborative multiagent planning under uncertainty (Chapter 6).

A subset of the ideas in Chapter 2 were previously presented in [134]. The works described in Chapter 3, Chapter 4, and Chapter 5 were originally presented in Stadler et al. [135], Stadler et al. [136], and Kurtz et al. [75], respectively.

Chapter 2

Background

2.1 Overview

In this section, we review the point-to-point single-agent and collaborative multiagent navigation problems. We review models for planning in large and unknown environments, and discuss the challenges associated with generating high-quality (e.g., low-cost and risk-aware) plans and policies in such domains.

2.2 The Continuous Single-agent Optimal Planning Problem

Consider a robot with state x defined in some continuous configuration space $\mathcal{X} \in \mathbb{R}^d$, where d is the dimensionality of the state. Let ζ be a continuous mapping through $\mathcal{X}$, i.e., $\zeta : [0, 1] \rightarrow \mathbb{R}^d$. The goal of the motion planning problem is to find a valid (i.e., non-colliding and dynamically feasible) plan that takes the robot from an initial state x_s to a goal state x_g,

where $x_s, x_g \in \mathcal{X}$, while minimizing a cost function c of the plan, such as time or distance:

$$\begin{aligned} \zeta^* &= \arg\min_{\zeta} c(\zeta; \Gamma) \\ \text{s.t.} \quad & f(\zeta(t); \Gamma) = 1 \ \forall t \in [0, 1] \\ & \zeta^*(0) = x_s, \zeta^*(1) = x_g, \end{aligned} \tag{2.1}$$

where, formally, $c : \zeta \to \mathbb{R}_{\geq 0}$ is a function that maps continuous trajectories to positive scalar costs, f is a feasibility function, $f : \mathcal{X} \to \{0, 1\}$, that indicates if a state is non-colliding and dynamically feasible, and Γ is additional inputs, such as the kinematic model and the map.

While the complexity of the optimization in Eq. 2.1 is dependent on problem details, the most general forms of the problem have been shown to be PSPACE-complete [122, 23, 82]. In general, there are two main approaches for solving the optimization: trajectory optimization and discrete planning. While trajectory optimization has been an effective tool for generating plans locally [158, 127], where problem structure restricts the set of ζ that must be considered by the optimizer, most solvers can be computationally expensive to deploy and prone to local minima, especially in large, potentially uncertain environments, like those encountered during outdoor navigation. Recently, a number of approaches have been developed to improve trajectory optimization methods for long length scale, high dimensional planning [94, 95]; however, these approaches are still limited in the environment and robot motion models that they can reason about, and have not been demonstrated in complex, uncertain environments. For this reason, in this thesis, we focus on discrete planning methods for solving the optimization in Eq. 2.1 in large and uncertain environments.

2.3 The Discrete Single-agent Optimal Planning Problem

One common approach for mitigating the complexity of the optimization in Eq. 2.1 is to approximate the continuous plan ζ as a sequence of discrete *primitive* actions $a \in \mathcal{A}$, where we define each action as an explicitly encoded trajectory with a specific beginning state $b(a)$

and a specific ending state $e(a)$, where $b(a), e(a) \in \mathcal{X}$. Formally, we define a sequence of primitive actions as a primitive plan, $p = \{a_0, a_1, ..., a_{n-1}\}$, where n is the number of actions in the plan, and $p \in P$, where P is the space of all possible discretized primitive plans. We also refer to the sequence of actions in a plan as $a_{0:n-1}$. We define the discrete optimal planning problem,

$$\begin{aligned} p^* = \arg\min_{p \in P} \ & \sum_{j=0}^{n-1} c(a_j; \Gamma) \\ \text{s.t.} \quad & f(a_j; \Gamma) = 1 \ \forall j \in \{0, 1, ..., n-1\} \\ & b(a_j) = e(a_{j-1}) \ \forall j \in \{1, ..., n-1\} \\ & b(a_0) = x_s, e(a_{n-1}) = x_g, \end{aligned} \tag{2.2}$$

where f is a partition function that maps every $a \in \mathcal{A}$ to feasibility, $f : \mathcal{A} \to \{0, 1\}$, where $f(a; \Gamma) = 0$ indicates that it is not feasible to execute the action a from action start state $b(a)$, and c is a scalar cost function defined over primitive actions, $c(a; \Gamma) \in \mathbb{R}$. Throughout this work, we will refer to $c(a_j; \Gamma)$ and $f(a_j; \Gamma)$ as primitive *navigation properties* of action a_j.

Given a planning problem, there are two key questions that must be answered in order to solve the optimization in Eq. 2.2:

1. Modeling the Problem: What is the right discrete model of the environment for planning?

2. Solving the Problem: How should an algorithm search for a solution to the problem, given an environment model?

In the following sections, we review common approaches for answering both questions.

2.3.1 Modeling the Problem

First, we discuss approaches for generating a discrete model of the environment that can be used for planning. Common approaches to discrete model generation include regular and

sampling-based discretizations of the state and action spaces. Both approaches require us to define a *state mapping function* [55], which maps (a subset of) the continuous state space into discrete objects (e.g., nodes) that can be ingested by a planning algorithm, and an *action space*, which defines a set of actions that can be applied to objects in the discrete state space, and that result in an updated agent state.

First, we discuss regular discretization techniques, which apply the state mapping function and action space at consistent intervals in the continuous state space. One of the earliest forms of regular modeling was the occupancy grid [38], which represented agent states as discrete locations in a grid-based model of the environment, and agent actions as holonomic movements to adjacent, unoccupied grid cells (i.e., up, down, left, right). Today, occupancy grids are still extremely common in robotics applications, and various modifications have been proposed, including to increase the efficiency of the representation (e.g., by using an octree data structure to represent occupancy [54], or by compressing the octree using information theoretic metrics [76]), and to represent other environmental features in the grid (e.g., semantics [5]). However, grid-aligned, holonomic representations of robot movement can result in poor planning for agents with complex dynamics, or for agents operating in environments with complex obstacles, especially when local adaptations to a holonomic plan (e.g., to obey the kinodynamic constraints of an agent attempting to execute the plan) can result in plan infeasibility. To mitigate this issue, methods that adapt occupancy grids (e.g., by using local trajectory optimizers to smooth grid based-plans [143], or by generating environment and agent-aware state lattices [118]) were developed to model feasible, low-cost trajectories for complex robots and environments while maintaining the benefits of a regularly-applied action set. Since their introduction, various state lattice adaptations have been proposed to increase plan expressivity and reduce plan costs, including updating action end points to better model low-cost paths in the environment [143, 55], and various approaches to reduce the computational cost of state lattice adaptation, including heuristic approaches [106], learned approaches [107], and memoization-based approaches [52].

Unfortunately, existing regular discretization-based planning models can be insufficient to enable low cost, computationally tractable navigation in large, outdoor environments. Methods that directly discretize the configuration space can be inefficient in large environments, as the number of discrete entities required to naively represent the environment scales polynomially in the length scale of the environment and exponentially in the dimension of the environment. While current methods attempt to balance between the fidelity of the environment model and computational tractability for planning, model resolution is often hand-tuned from expert knowledge. Poor resolution selection can result in computational intractability if the representation is too detailed, or can result in high-cost or infeasible plans if the representation is not detailed enough to represent key features of the environment, like a narrow passageway on the shortest path to the goal. When combined with an optimal search strategy, regular discretization-based planning models are *resolution complete* and *resolution optimal*. They are guaranteed to find a valid plan if one exists given the selected discretization resolution r, and they are guaranteed to find an optimal plan in the r-discretized environment model. However, while regular discretization-based models can be effective in environments where it is easy to select a good planning resolution r, it is not obvious how to select a good r for outdoor navigation, as the resolutions of important navigational features vary significantly (e.g., tree stump, bush, lake, field).

One approach to mitigating the challenge of selecting a discretization resolution for planning is sampling-based discretization. In sampling-based discretization, discrete planning nodes are generated by sampling states from a distribution over valid agent configurations. Then, a planning graph is generated by connecting nodes using a deterministic edge generation function. Common connection functions include connecting each state to its k nearest neighbors [66], or connecting each state to all neighbors that lie within a d-ball of the state [65]. Because they do not require a planner to exhaustively consider many states at a given planning resolution, sampling-based planning graphs can often represent long length scale and high dimensional environments with fewer nodes and edges than regular planning graphs;

for example, in an outdoor environment, a sampled graph may contain few samples in a large, open field, but many samples in a forest with dense obstacles. However, to generate these efficient graphs, an agent must encode information about the environment (e.g., the existence of a field or forest) in its sampling distribution. Significant effort has been invested in generating informed sampling strategies for sampling based-motion planning, including using heuristics to improve modeling in narrow passageways and near obstacles [56, 3], using heuristics to quickly find an initial path [155, 130], using heuristics to refine an existing path [44, 62], and learning sampling distributions that are conditioned on environment information [58, 59, 101, 92, 150]; however, these approaches largely require access to global occupancy information. One notable exception is Liu et al. [92]; which developed a sampling distribution for planning in uncertain environments using local, online-generated occupancy and semantic maps; however, the approach only generated a sampling distribution in a local region around the robot, and only considered coarse global information, such as whether a goal was indoors or outdoors, when generating the sampling distribution. While the approach enabled improved local planning, the sampling distribution did not capture globally relevant information for planning. Concurrently, sampling-based approaches have been extended to generate kinodynamically feasible motion graphs, for example, by constraining tree expansion to kinematically feasible paths [47], and by generating new nodes by forward simulating vehicle dynamics from existing nodes [90]. However, while sampling-based motion planners can be an effective tool for planning with complex teams in large and high dimensional known environments, existing approaches are not well suited for long-horizon navigation in long-length scale outdoor environments, where planners must reason about coarse and uncertain global information at long length scales.

2.3.2 Solving the Problem

Given a discrete representation of the environment, various graph-based search methods can be applied to optimally search in the planning representation. The most commonly used

graph search method is A* [119], an optimal algorithm that guides search to low cost nodes in the state space using an admissible heuristic, or a heuristic that underestimates future plan costs. Unfortunately, the computational complexity of the A* algorithm scales linearly in the number of actions considered by the agent at any planning step, or the search *breadth*, and exponentially in the the number of planning steps in the plan, or the search *depth*; formally, search complexity is $\mathcal{O}(breadth^{depth})$. Therefore, in practice, optimizing Eq. 2.2 over all plans $p \in P$ can be intractable as the length of the largest dimension of the configuration space and the potential number of planning steps in any plan increases, as discrete planning complexity scales exponentially with the number of steps in a plan.

2.3.3 Multi-Query Planning

Multi-query planners reduce the computational cost of planning over multiple planning trials by precomputing, recording and reusing planning artifacts during online planning. For instance, Berenson et al. [12] designed a planner that generated and used a library of general manipulation plans over multiple planning queries. When a new query was received, the planner simultaneously attempted to directly solve the query and attempted to retrieve and repair a plan from the existing plan library. Under this paradigm, the agent found, repaired, and executed similar solutions from the plan library when such plans were available, but used directly-calculated plans when no similar solutions existed in the plan library. Phillips et al. [117] developed the experience graph method, which recorded solutions to previous planning queries in a sparse *experience graph*, and then guided agents planning for novel queries to solutions that overlapped with existing solutions. However, these and other multi-query planners are mainly focused on minimizing duplicate computations locally and do not fundamentally reduce the dimensionality of the planning problem or enable enhanced reasoning about the global quality of candidate plans. In Chapter 3, we develop a model that uses online planing results to update a coarse model of global navigation properties in an environment, and then use the model to improve planning efficiency over the course of a

multi-query planning trial.

In the next sections, we introduce two planning paradigms that can be used to reduce the complexity of planning in large, outdoor environments, hierarchical planning and uncertainty-aware planning.

2.4 The Hierarchical Discrete Single-agent Optimal Planning Problem

One method for decreasing the computational complexity of planning at long length scales is hierarchical planning. Hierarchical planning algorithms increase the tractability of solving the optimization in Eq. 2.2 at long length scales by representing the plan space using a combination of primitive and *high-level* actions $\mathbf{a} \in \mathbf{A}$, which are actions based on simplified models of the planning problem that do not necessarily capture all planning constraints, but that can be used to guide a search process towards low cost primitive plans[1]. While various hierarchical planning methods have been proposed (e.g., [143, 18, 11, 20]), one approach is to use high-level actions to generate high-level plans $\mathbf{p} \in \mathbf{P}$ that approximate solutions to Eq. 2.2, where $\mathbf{p} = \{\mathbf{a}_0, \mathbf{a}_1, ..., \mathbf{a}_{\mathbf{n}-1}\}$, $\mathbf{n}$ denotes the number of high-level actions in the high-level plan, and $\mathbf{P}$ denotes the high-level plan space. Formally, we write the high-level planning problem,

$$\begin{aligned} \mathbf{p}^* = \arg\min_{\mathbf{p}\in\mathbf{P}} \; & \sum_{j=0}^{\mathbf{n}-1} \mathbf{c}(\mathbf{a}_j; \mathbf{\Gamma}) \\ \text{s.t.} \quad & \mathbf{f}(\mathbf{a}_j; \mathbf{\Gamma}) = 1 \; \forall j \in \{0, 1, ..., \mathbf{n}-1\} \\ & \mathbf{e}(\mathbf{a}_{j-1}) \Diamond \mathbf{b}(\mathbf{a}_j) \; \forall j \in \{1, ..., \mathbf{n}-1\} \\ & x_s \in \mathbf{b}(\mathbf{a}_0), x_g \in \mathbf{e}(\mathbf{a}_{\mathbf{n}-1}), \end{aligned} \tag{2.3}$$

where $\mathbf{f}$ is a partition function that maps every high-level action in $\mathbf{A}$ to its validity, $\mathbf{f} : \mathbf{A} \rightarrow \{0, 1\}$, $\mathbf{c}$ is a scalar cost function defined over high-level actions, $\mathbf{c}(\mathbf{a}; \mathbf{\Gamma}) \in \mathbb{R}$, $\mathbf{\Gamma}$ is

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

additional inputs, $\mathbf{b}(\mathbf{a}_j)$ and $\mathbf{e}(\mathbf{a}_j)$ denote the set of valid start states and terminal states for high-level action $\mathbf{a}_j$, and $\Diamond$ is a function that tests the connectivity of two high-level actions in a high-level plan[2].

While there are multiple ways to solve the optimizations in Eq. 2.2 and Eq. 2.3, one common approach is to represent the optimizations as two separate graph search problems: a high-level graph search problem and a primitive graph search problem, where graph nodes represent states and weighted, directed graph edges represent actions in the respective optimizations. Planning begins with high-level graph search, and continues until a low-cost high-level plan $\mathbf{p}$ is found. The high-level plan is then used to increase the efficiency of primitive graph search by guiding planning to primitive subgraphs that are likely to contain low-cost primitive solutions (e.g., by using high-level plan costs as heuristics [68], or by constraining primitive search to subgraphs that are consistent with the high-level plan [18]), reducing the total computation required to find such solutions. When a search process uses a high-level plan $\mathbf{p}$ to guide the search for a primitive plan p, we call the resulting primitive plan p a primitive *refinement*. In the following sections, we discuss four types of hierarchical planning paradigms, multi-resolution planning, task-based planning, correct abstractions, and learned abstractions, with a focus on hierarchical model generation and techniques for solving the optimization in Eq. 2.3.

2.4.1 Multi-Resolution Planning

As discussed in Section 2.3.1, one common challenge when developing a model for robot planning is determining the right resolution for planning. Often, robot decision-making is conditioned on multi-resolution environmental information; for example, a robot may only require coarse environment information to navigate across an open field, but would require a more detailed environment model to navigate across a narrow stream using a small bridge.

[2]Note that different hierarchical planning approaches have different high-level plan connectivity requirements. For example, when $\Diamond = \subseteq$, this ensures that high-level action $\mathbf{a}_j$ can be executed from all terminal states of action $\mathbf{a}_{j-1}$.

One approach for enabling intelligent yet efficient path planning in such environments is to generate multi-resolution environment models that provide different levels of environment details in different environmental regions. Some simple versions of the approach focus on maintaining detailed information in regions near the robot; for example, Behnke [11] developed an approach that maintained a fine resolution map in a local window around the robot, and a coarse resolution map in the rest of the environment. This technique provided an agent with a sufficiently detailed map to make local planning decisions (e.g., to avoid local obstacles, like navigating around a rock), while enabling computationally inexpensive high-level planning at longer length scales (e.g., by planning to avoid large obstacle regions, like lakes). More recent approaches have used robot-centric polar maps to model an agent's local environment in a way that captures sensor uncertainty [156]; specifically, polar grid cells increase in volume as distance from the robot increases, so they naturally model increases in sensor uncertainty at long distances from the robot.

While robot-centric multi-resolution mapping approaches have been effective for local planning tasks, like collision avoidance, they reduce the amount of global information available to the robot. To efficiently represent and plan over global information, a number of approaches have developed environment-centric multi-resolution map representations. These approaches have the explicit goal of representing key global environmental features at a fine resolution, while abstracting away other information when possible to maintain computational tractability. Botea et al. [20] used environment topology to define regions for planning, then precomputed local trajectories in the regions. Then, they defined an abstract graph based on the topology of the regions, generated plans in the abstract graph, and refined the plans using the precomputed trajectories during online planning [20]. Additionally, some approaches have used occupancy maps to develop multi-resolution, environment-based cell decompositions for planning (e.g., using octrees [64] and wavelet decompositions [31, 146, 8, 147]). Recently, the Information Bottleneck (IB) has been used to compress an environment model (e.g., an occupancy map, a cost map) in way that minimizes information

loss. Such representations can be used directly for planning [76, 77, 78] (e.g., by directly converting the multi-resolution representation into a planning graph), or they can be used to augment other planning representations. For example, Nelson et al. [108] used an IB method to generate an reduced environment representation over which to calculate an environmental coverage metric, and then used the metric to guide high-level coverage planning. While the metric was computationally expensive to calculate in the original environment model, it was efficient to calculate in the compressed environment representation, and the compressed representation was sufficient to guide the coverage planner to better solutions.

2.4.2 Task-Based Planning

While multi-resolution hierarchical planning approaches can increase planning efficiency by clustering regions of the environment that have similar planning properties, they are fundamentally focused on representing the entire planning space in a way that enables efficient discrete planning. However, even with intelligent clustering strategies, generating compressed representations of large and complex environments can be computationally prohibitive. One approach to mitigating this complexity is task-based planning. The goal of task planning is to generate a sequence of actions, modeled as changes to predicates about the world, that result in predicates about the goal state being satisfied. Importantly, this form of planning does not require the high-level planner to have knowledge about the entire state of the world; it only needs to reason about the partial state relevant to satisfying the goal predicates. In general, we assume access to a high-level model of the planning environment that is capable of representing relevant predicates about the environment in a high-level state, and that contains actions that result in predicate changes. Notably, the actions do not need to map to primitive refinements prior to planning; most task planning-based approaches generate a high-level plan that satisfies the goal predicates using a heuristic, and then attempt to generate a primitive refinement of the high-level plan. One early example of the approach was developed by Kaelbling and Lozano-Pérez [63], which interleaved task planning and task

execution in an online fashion, and maintained completeness guarantees by assuming that all state changes were reversible via (a sequence of) action(s) in the high-level planning domain. More recent work has demonstrated the ability to learn high-level planning abstractions based on sets of primitive skills [71], factored MDPs [27], and even learned planning models [28, 26]; notably, Chitnis et al. [28] developed an abstract planning representation that combined symbolic state abstractions and jointly learned abstract actions, abstract transition models, primitive transition models, and samplers in a unified framework. For more detailed treatments of task and task and motion planning problems, we refer the reader to recent surveys [45, 49]. Finally, task planners have been developed in parallel with formalisms for defining task-based planning problems; currently, the Planning Domain Definition Language (PDDL) [1] and its variants are most commonly used in robotics applications. PDDL and similar languages have been invaluable as a standardization method for TAMP problems, enabling easier testing and benchmarking of algorithms and domains.

While task planning has most commonly been applied to (mobile) manipulation problems in environments with conventional objects (e.g., kitchens and homes, where common actions may include opening a drawer, picking up an object, and placing the object in a specified location), task planning can be applied to navigation tasks. Task-based models for navigation are most common when high-level navigation plans can be represented using logical primitives, such as navigating to a room, or merging to the left. For example, Ding et al. [35] modeled complex autonomous driving actions, like merging and turning, as high-level actions, approximated their cost and safety, and used the approximations during high-level planning to find safe trajectories. Other approaches for indoor navigation used similar high-level plan cost and feasibility calculations to prune a task-based action space for planning [49, 93, 142, 121]. Finally, Edelkamp et al. [37] combined high-level actions and temporal PDDL to solve multi-goal navigation problems with temporal constraints in a warehouse domain.

2.4.3 Correct Abstractions

In non-adversarial settings, it can be overly restrictive to require all or even most refinements of a high-level action to be feasible and low cost. If a planning agent is presented with any feasible, low cost refinement, it can simply select that refinement. Marthi et al. [96, 97, 98] formalized this idea as *angelic abstractions*, or abstractions for which at least one valid primitive refinement exists. Then, they developed hierarchically optimal planning algorithms using angelic abstractions. This work was extended by Vega-Brown and Roy [148], who developed angelic abstractions that were guaranteed to produce optimal or near-optimal primitive solutions for some limited problem types. Specifically, they developed abstractions for navigation in worlds that could be modeled as compositions of convex and ϵ-convex regions[3], and for navigation in a *door puzzle*, a navigation problem where an agent must to navigate to buttons (specific locations) to open doors that lie between the agent and the goal. While angelic abstractions have been demonstrated to enable efficient, optimal planning in some planning domains, unfortunately, it is not obvious how to generate coarse, informative angelic abstractions in large, real-world environments in the presence of non-convex obstacles and geometric uncertainty.

2.4.4 Learned Abstractions

Recently, a number of approaches for learned hierarchical navigation have been proposed. Some approaches use occupancy maps or other overhead representations, such as street maps and satellite images, to learn the properties of high-level subgoals, which can be viewed as an implicit high-level action space. Stein et al. [137] used a partial occupancy map to identify boundaries between known and unknown space as subgoals during goal-directed navigation under uncertainty, and then learned the navigation properties of the subgoals (i.e., the cost of

[3]Intuitively, ϵ-convex regions are regions in which obstacles do not have a significant impact on path costs. Formally, ϵ-convex regions are regions in which the shortest valid path between any two points in the region is no greater than $(1 + \epsilon)$ times the Euclidean distance between the points.

navigating to the goal via the subgoal, the probability of navigation success via the subgoal, and the cost of failing to reach the goal via the subgoal). The learned subgoal properties were used to order the subgoals during planning. The approach has also been modified for temporally extended tasks by applying the idea of a subgoal to product automata [21]. Shah and Levine [128] combined local subgoals, proposed by a local planner, with global geographic 'hints', such as an overhead image, to learn a subgoal heuristic function for guiding low-level subgoal selection. Other approaches generate explicit state- or state- and action-based abstract planning representations, and then learn the navigation properties of the high-level states or state-action pairs. Some approaches calculate high-level navigation properties, like cost and feasibility [69], or transition dynamics [152], that can be incorporated into high-level planning models, while other approaches directly learn a high-level value function [39, 140, 126]. For example, Everett et al. [39] learned an agent cost-to-go function, given a semantic map, as an image-to-image translation problem, and Tamar et al. [140] and Schleich et al. [126] approximated value functions at various levels in abstraction hierarchies using the value iteration network (VIN), a type of neural network that was designed to encode the value iteration algorithm in a differentiable computation [140].

2.5 The Discrete Single-agent Optimal Planning Problem under Uncertainty

The previous formulations assumed that an agent had full knowledge of the planning environment and its own dynamics. In practice, complete information is unlikely to be available for robots operating in novel, natural environments. An agent's local information is limited by the range of its onboard sensing, and while the agent may have some sparse global information (e.g., from remote sensing, like overhead images collected from a satellite or air vehicle), this information is unlikely to provide exact occupancy information for the full environment. While many robots simply assume that unknown areas are free space to maintain

single-step planner admissibility, this assumption can lead to overly optimistic planning and poor online outcomes in uncertain environments.

To avoid such outcomes, robots must be capable of modeling environment uncertainty and using the models of uncertainty during planning. Prior to modeling uncertainty, it is important to understand the different types of uncertainty and their impacts on planning. Here, we discuss *epistemic uncertainty* and *aleatoric uncertainty*. Epistemic uncertainty is uncertainty caused by a lack of information about the true state of an environment; this type of uncertainty can be reduced by observing the environment. For example, uncertainty caused by an agent not having a dense occupancy map of an environment is epistemic uncertainty. While the agent is uncertain about the environment state, the true environment state is static, and the state could be observed by, in this case, building a larger map. In contrast, aleatoric uncertainty is uncertainty caused by random chance, and cannot be reduced by observation. For example, consider an environment where a rainstorm occurs with some probability at each timestep and renders all paths muddy and untraversable. Here, uncertainty due to rainstorms is aleatoric, as no information from the current timestep can be used to reduce uncertainty about rain in future timesteps. For a more careful treatment of uncertainty types, we refer the reader to Hüllermeier and Waegeman [57]. In this work, we focus on modeling and planning in environments subject to epistemic uncertainty. Specifically, we would like to enable agents to navigate quickly and efficiently in static environments based on partial, high-level environment information.

In this work, we assume that we have access to a probabilistic model of environmental uncertainty. Formally, we define a world ω as a specific instantiation of environmental uncertainty, where $\omega \in \Omega$, and Ω is the space of all possible instantiations of environment uncertainty; for example, in a city subject to a natural disaster, ω could be an assignment of impacted bridges to their true traversabilities. Additionally, we assume that we have access to a prior distribution, $\mathcal{P}(\Omega)$, that describes the likelihood of a given world occurring, $\omega \sim \mathcal{P}(\Omega)$; we can then combine the prior model and online observations to generate a belief b,

or a posterior probability distribution over possible worlds, given environment observations.

Given the model of environmental uncertainty, we now develop a method for planning using the model. Because the environment is uncertain, we can no longer optimize a planning objective to minimize agent cost, as the true cost of the plan is only revealed at execution time. Instead, we optimize a policy $\pi \in \Pi$, where $\pi : \mathcal{X} \times B \rightarrow A$, that maps agent states and beliefs to actions, and action quality is calculated in expectation over the agent belief:

$$\begin{aligned} \pi^*(x, b) &= \arg\min_{\pi \in \Pi} \mathop{\mathbb{E}}_{\omega \sim b} \Big[\sum_{j=0}^{n-1} c_\omega(\pi_j; \Gamma) \Big] \\ \text{s.t.} \quad & f(\pi_j; \Gamma) = 1 \ \forall j \in \{0, 1, ..., n-1\} \\ & b(\pi_j) = e(\pi_{j-1}) \ \forall j \in \{1, ..., n-1\} \\ & b(\pi_0) = x_s, e(\pi_{n-1}) = x_g, \end{aligned} \tag{2.4}$$

where c_ω is a world-specific cost function, and Π is the space of all valid single-agent policies.

In general, planning under uncertainty is at least as hard as planning in deterministic environments, and it is often much harder. To make planning tractable, many algorithms for planning under uncertainty are tailored to a specific planning task and environment model. In the following sections, we review recent models and assumptions for navigation under uncertainty.

2.5.1 Learning Properties of Unknown Spaces

One common assumption in the motion planning literature is that unknown or unmapped regions of an environment are free space. In practice, unknown regions are often far more complex, and optimistic assumptions about region costs and traversabilities can lead to poor global planning outcomes once an agent explores the unknown region. Recently, a number of works have been developed to predict the planning properties of environmental regions for which agents do not have dense geometric maps. Specifically, these approaches used offline datasets, generated using complete map information, to learn correlations between partial

maps and planning properties. Stein et al. [137] developed a subgoal-based representation for navigation in unknown indoor environments, where subgoals were defined as boundaries between known and unknown space, and then learned to predict the expected cost of navigating to the goal via a subgoal, the expected cost of failing to reach the goal via a subgoal, and the probability of reaching the goal via the subgoal. The learned properties were used to evaluate the Bellman equation for each subgoal, and the subgoal with the lowest expected cost was selected as the next agent waypoint. Everett et al. [39] learned a cost-to-go function for unmapped regions of an environment as an image-to-image translation problem. The input image was a partially observed semantic map, and the network output, the predicted cost-to-go function, was used in a discrete planner for the last-mile delivery task. Finally, Liu et al. [92] learned a sampling distribution for sampling-based motion planning, conditioned on partially observed occupancy and semantic maps, as well as coarse global information, and demonstrated improved planning performance as compared to a sampling-based motion planner using a uniform sampling strategy. While these examples clearly demonstrate that partial environment structure can be used to predict navigation properties beyond an agent's geometric map, it is unlikely that the approaches would be effective for predicting navigation strategies far from the agent and the partially observed occupancy map, as local environment information is unlikely to be predictive of global environment structure and global planning properties at large length scales.

2.5.2 Modeling Cost Uncertainty on Environment Graphs/Roadmaps

To enable intelligent global navigation under uncertainty, a number of approaches have proposed to model uncertain environments as stochastic graphs. Murphy and Newman [103] used overhead images to generate probabilistic costmaps of unmapped environments [102], and then converted the costmaps into stochastic planning graphs, where uncertain edge costs were modeled as Gaussian distributions; extensions of the work also used prior vehicle experiences in the costmap generation process [105]. The authors then used uncertainty-

aware planning algorithms [116, 104] to generate trajectories that traded off between planning efficiency and expected plan quality in a principled manner. Similar approaches have been extended to the online setting, where dynamically discovered plan costs influence future risk-based planning decisions [29]. Dey et. al. [33] used a Monte Carlo planning approach to generate agent policies for navigation on a graph with uncertain, correlated edge costs, which were modeled as Gaussian distributions drawn from an underlying Gaussian Process (GP) of environment traversability. By explicitly modeling environmental uncertainty and using the information in a planning algorithm, these approaches improved global planning performance as compared to baseline approaches that did not consider environmental uncertainty during planning. However, the approaches made simplifying assumptions about the distributions of path costs (i.e., they assumed costs were Gaussian), which can be a limiting assumption for some problem types, as described in the next section.

2.6 Modeling Traversability Uncertainty on Graphs: The Canadian Traveller's Problem

While Gaussian cost uncertainty can be used to model various real-world planning phenomena, including terrain uncertainty and delays, other uncertainty types, such as traversability uncertainty, cannot be modeled using Gaussian distributions. Consider a robot attempting to navigate in a region under a tree canopy in an overhead image. The unobservable terrain under the canopy may be flat, grassy, and traversable, or it may be rocky, littered with branches, and untraversable. If the terrain is traversable, navigating along the path will be low cost; otherwise, the same action will have high cost. In practice, traversability uncertainty often results in multimodal distributions over plan costs. To model such uncertainty in unknown environments, we adopt the Canadian Traveller's Problem (CTP) formulation [114], a stochastic graph-based planning model with unknown edge traversabilities. This well-known problem was originally designed to model route finding in Canada, where roads

were assumed to be snowed in and impassable with some probability. In the graph, nodes indicated locations, edges indicated possible paths, and each edge was assigned a traversability probability, or a probability that the edge was traversable, during a given planning trial. Formally, the planning model is defined as follows: let G be an undirected graph with vertices V and m edges E, $G = \langle V, E \rangle$, where the cost of traversing edge e by agent λ is given by a known scalar cost function[4], $c_\lambda : E \to \mathbb{R}^+$; we assume c_λ is agent finishing time. Additionally, we assume that the traversability (weather) of each edge w_e is initially unknown but takes a binary value, $w_e \in \{0, 1\}$, and an environment weather $w = \{w_1, w_2, ..., w_m\}$ is defined as a joint assignment of graph edges to weathers w_e. At the beginning of each planning instance, w is drawn from m independent Bernoulli distributions with parameters $1 - \rho_e$, where $\rho_e \in [0, 1)$ is the probability that an edge is blocked, or untraversable, such that $p(w) = \prod_{e \in E} \text{Bern}(1 - \rho_e)$. We assume that the weather changes slowly relative to the timescale of team navigation (e.g., in the aftermath of a single event that changed the environment, like a climate disaster, or due to long timescale infrastructure wear or environmental change), so we assume w is constant during planning and policy execution. We also assume that each agent has a local, noiseless sensor that can acquire ground truth observations $z_e = w_e$ of the traversability of edges incident to the agent's current vertex. These two assumptions mean that the only nondeterminism in the problem is caused by the initially unknown weather, and once the traversability of an edge is (noiselessly) observed, it does not change, allowing agents to act deterministically on the observed subgraph. The goal of the CTP is to find the minimum expected cost policy π for navigating from a start node s to a goal node G in the uncertain graph:

$$\pi^*(v, b) = \arg\max_{\pi \in \Pi} V^\pi(v, b), \tag{2.5}$$

[4]For notational convenience, we overload the cost function to take the endpoints of an edge as input: $c_\lambda : V \times V \to \mathbb{R}^+$.

where S is the discrete state space, $s \in S$, and Π is the space of single-agent policies, $\pi : S \times B \to A$,

$$V^{\pi}(v_a, b_a) = \mathop{\mathbb{E}}_{w \sim b_a} \left[\sum_{t=a}^{\infty} c(v_t^{\pi}, v_{t+1}^{\pi}) \right], \tag{2.6}$$

and v_t^{π} is the vertex reached by the agent at time t when executing policy π from time a until time t.

Additionally, various extensions to the CTP have been proposed, including CTPs with uncertain edge traversal costs instead of stochastic edges [112] and disjoint-path CTPs [15, 16]. Bnaya et al. [16] and Bnaya et al. [15] consider multiagent planning and remote sensing on CTPs, respectively, yet both approaches have limited scope; Bnaya et al. [16] considered the special multiagent planning case when there is no cost for an agent to remain at its current vertex, reducing the approach to a series of sequential CTP problems, and Bnaya et al. [15] developed solutions for a sensing agent with limited dynamics and no independent planning task. Additionally, policies have been developed for multi-trial CTPs [17]. In Chapter 4, we develop an approach for generating multiagent policies for CTPs that minimize team makespans for one ground vehicle/n air vehicle teams [136].

Unfortunately, calculating exact solutions for most CTP formulations has been shown to be PSPACE-Complete [114, 42]. While exact solutions have been developed for small subclasses of stochastic graphs [112, 15, 41], general state of the art solvers reformulate the CTP as a Partially Observable Markov Decision Process (POMDP), and then use Monte Carlo simulation, a common POMDP solution technique, to generate uncertainty-aware agent policies [2, 40]. In the following section, we introduce POMDPs, and then introduce the POMDP formulation of the CTP.

2.7 Partially Observable Markov Decision Processes (POMDPs)

Partially Observable Markov Decision Processes (POMDPs) are a general model for decision making problems under uncertainty. A POMDP is a 7-tuple (S, A, T, R, Z, O, b_0):

1. States S: The set of all valid configurations of the system; the state may be partially unknown.
2. Actions A: The set of all actions that can be applied to the system.
3. Transition function $T : S \times A \times S \to [0, 1]$: A function that determines the probability that an agent will reach system state s' after applying action a at state s.
4. Rewards $R : S \times A \to \mathbb{R}$: The reward received after applying action a at state s.
5. Observations Z: The set of all valid observations that can be received during system execution.
6. Observation function $O : S \times A \times Z \to [0, 1]$: A function that determines the probability of receiving observation z after applying action a at state s.
7. Initial belief b_0: An initial belief, or probability distribution, over the unknown state variables.

The goal of the POMDP is to find a policy π that maximizes the value function V, or expected future reward, of reaching a goal state from the current state s with belief b:

$$\pi^*(s, b) = \arg\max_{\pi \in \Pi} V^{\pi}(s, b), \tag{2.7}$$

where in this work we explicitly differentiate between the known state s and the belief over the unknown state b, as in a Mixed-Observability Markov Decision Process (MOMDP)

formulation [113], although some other works include the known state in the belief variable b.

2.7.1 Types of POMDPs

POMDPs are an extremely general formulation with many different subclasses, and the specific variant of the problem has a large impact on computational tractability and possible solution methods. For example, POMDPs can model problems with discrete or continuous action and observation spaces, long time horizons, and delayed rewards, and various solution techniques have been developed for different model types [131, 139, 132, 141, 51, 91, 85, 4]; see Kurniawati [74] for a comprehensive review. The structure of the CTP problem, which has small, discrete action and observation spaces, enables the CTP to be modeled as a special type of POMDP – a Deterministic POMDP (DetPOMDP) [19] – in which it is feasible to consider long-horizon actions with delayed rewards. Specifically, Deterministic POMDPs are POMDPs that model epistemic uncertainty only; they assume that there is an initially hidden state (e.g., due to traversability uncertainty in a forest) that does not change during the course of a planning trial. While this is not true of all Deterministic POMDPs, in the CTP case, the uncertain state can be observed using sensing during a planning trial. Note that the absence of aleatoric uncertainty, such as action stochasticity that could lead an agent to an imprecise state in the environment, or observation stochasticity, such as noisy sensor readings that could result in incorrect traversability estimates, ensures that the total problem uncertainty only decreases during a planning trial. This fact ensures that the size of the search space for planning is both finite and relatively small, and that the problem is tractable to solve.

2.8 The CTP as a POMDP

Formally, we model the CTP problem as a POMDP [2]:

- Fully observable states S: The set of graph vertices V.
- Partially observable states: The set of graph weathers W.
- Actions: The set of incident, traversable graph edges for the agent at each vertex, including self-loops.
- Observations: The traversability w_e of an edge, $z_e = w_e \in \{0, 1\}$.
- Observable state transition function: Deterministic transitions along observed, incident, traversable graph edges to adjacent vertices.
- Partially observable state transition function: No transition, as the weather w is assumed to be constant during a trial.
- Observation function: Deterministic observations of the traversability w_e of graph edges incident to the agent.
- Reward R: The negative agent cost, $R = -\sum_{t=0}^{\infty} c_(v_t, v_{t+1})$, where, by assuming the reward of the agent remaining in its goal state is zero, we ensure the stationary policy has zero reward and the infinite sum is well defined.
- Initial belief b_0: The prior belief over the weather, $b_0 = \{\text{Bern}(1 - \rho_0), ..., \text{Bern}(1 - \rho_{m-1})\}$, where $b_0 \in B$.

While individual agents can generate uncertainty-aware policies that trade off between exploration and exploitation in novel environments, their long-range planning capabilities are largely constrained by the range and quality of their local sensing. Collaborative multiagent teams have the potential to increase team planning performance by providing non-local information to teammates, for example, by exploring the local environment and sharing traversability observations with teammates. In the following sections, we review the discrete multiagent planning problem and solution techniques generally, and then discuss the collaborative multiagent planning problem in uncertain environments.

2.9 The Discrete Optimal Multiagent Planning Problem

Multiagent planning describes a broad class of algorithms and techniques used to generate plans for more than one agent, and methods vary drastically for different problems and teams. In fact, significant effort has been devoted to categorizing multiagent planning problems based on problem characteristics [46, 72]; some key characteristics include agent objectives (competing, independent, collaborative), planning locations (centralized, distributed), and communication types (no communication, limited communication, full communication). For example, in adversarial settings, agents or teams have competing objectives and often do not communicate, and common solutions include game-theoretic analyses (e.g., pursuit-evasion games, [30]). In Multi-Agent Path Finding (MAPF) [138], agents may be centralized and may communicate, but they largely cooperate to avoid collision (i.e., local collision constraints require agents to plan jointly); the problem is most commonly formulated as conflict-based search, where the planner identifies potential agent collisions and reroutes agents to optimize team performance [129, 86]. In this work, we focus on the problem of collaborative planning, where agents complete individual or collaborative tasks to enable problem completion via teaming or to decrease problem completion time. For example, two agents may lift a heavy object, a mobile manipulator may open a door for a ground vehicle, and a firefighting robot may put out a fire to enable a medic robot to rescue survivors [157].

Formally, we define a collaborative team Λ as a set of l (possibly heterogeneous) robots λ, $\Lambda = \{\lambda_0, ..., \lambda_{l-1}\}$. Let a team plan p_{ma} be a sequence of tuples of individual agent actions at each timestep,

$$p_{ma} = \{(a_{\lambda_0,0}, a_{\lambda_1,0}, ..., a_{\lambda_{l-1},0}), (a_{\lambda_0,1}, a_{\lambda_1,1}, ..., a_{\lambda_{l-1},1}), ..., (a_{\lambda_0,n-1}, a_{\lambda_1,n-1}, ..., a_{\lambda_{l-1},n-1})\}, \tag{2.8}$$

where $a_{\lambda_i,j}$ is the action taken by agent λ_i at timestep j, and $p_{ma} \in P_{ma}$, where P_{ma} is the joint space of team plans. We additionally assume access to a *no-op* action at each agent's

the terminal state that has a duration of 1 timestep and a reward of 0, which enables us to model agent plans of various durations (i.e., when an agent reaches a goal before a teammate) in a single team plan. Then, we can formalize the collaborative multiagent planning problem as an optimization over team plans subject to agent feasibility constraints:

$$
\begin{aligned}
p^*_{ma} &= \arg\min_{p_{ma} \in P_{ma}} \sum_{j=0}^{n-1} c((a_{\lambda_0,j}, a_{\lambda_1,j}, ..., a_{\lambda_{l-1},j}); \Gamma) \\
\text{s.t.} \quad & f(a_{\lambda_i,j}; \Gamma) = 1 \ \forall i \in \{0, 1, ..., l-1\} \ \forall j \in \{0, 1, ..., n-1\} \\
& b(a_{\lambda_i,j}) = e(a_{\lambda_i,j-1}) \ \forall i \in \{0, 1, ..., l-1\} \ \forall j \in \{1, ..., n-1\} \\
& b(a_{\lambda_i,0}) = x_{\lambda_i,s}, e(a_{\lambda_i,n-1}) = x_{\lambda_i,G} \ \forall i \in \{0, 1, ..., l-1\},
\end{aligned}
\tag{2.9}
$$

where $x_{\lambda_i,s}$ and $x_{\lambda_i,G}$ denote the start and goal states of agent λ_i, and we assume that each agent is equipped with a low-level controller capable of avoiding collisions, so we do not explicitly model non-collision constraints between teammates.

Within collaborative planning, there are four commonly studied subproblems used to generate solutions to the optimization in Eq. 2.9: task planning, task assignment, scheduling, and motion planning. The problems have been studied individually, and have been combined in various ways to solve subsets of the full collaborative planning problem, including task and motion planning (task planning and motion planning), multiagent planning (task planning, task allocation, and scheduling), and multi-vehicle routing (task allocation and motion planning) [99]; see Messing et al. [99] for a review.

Current state-of-the-art solution techniques for the problems are as varied as the problems themselves; in this work, we focus on search-based solution methods. For tasks with few or no inter-agent constraints, task planning as described in Section 2.4.2, followed by independent task allocation, scheduling, and motion planning, can lead to high-quality plans. However, in complex domains, it is difficult to develop abstractions that capture all low-level planning constraints, and solvers that assume the *downward refinement property* [7], or that all high-level plans have valid primitive refinements, tend to fail or be inefficient. Conversely,

for tasks with significant inter-agent constraints that can cause local adjustment-based techniques to fail, treating the system as a single high-dimensional robot and solving for all robot plans simultaneously can be a way to find a satisficing solution. For example, Crosby et al. [32] generated a factorization by assigning specific agents to specific task that reduced the multiagent planning problem into sets of single-agent planning problems with temporal constraints that could be combined to recover a full solution, and Kottinger et al. [73] developed a variant of conflict-based search that jointly optimized trajectories for agents with conflicts that could not be resolved using simple heuristics.

Recently, a number of hierarchial, interleaved, search-based planners have been developed to efficiently balance between exploring new high-level plans and exploiting existing high-level plans that are likely to have valid and/or low cost primitive refinements during the search process. For example, the GRSTAPS framework is a hierarchical, interleaved planner that attempts to ground partial task plans and prune infeasible branches early during search. The method also uses heuristics to guide planning at each stage in the planning process [99]. The Task Allocation, Scheduling, and Motion planning modules of GRSTAPS, originally presented as ITAGS [109], have been extended for plan repair in dynamic environments [110], for robust plan selection under agent trait and task requirement uncertainty [115], and for quality-based task allocations [111], which reformulates the binary objective of task completion during task allocation to a performance maximization objective, given makespan constraints.

While many collaborative planning approaches have demonstrated improvements over non-collaborative baselines in known environments, they are not designed to reason about environmental uncertainty. However, for the problem of navigation at long length scales, it is likely that an environment will be at least partially uncertain prior to planning. In the following section, we formalize the discrete optimal multiagent planning problem under uncertainty.

2.10 The Discrete Optimal Multiagent Planning Problem Under Uncertainty

Finally, we consider the multiagent planning problem under uncertainty. Let a team policy π_{ma} be a sequence of tuples of individual agent policies at each timestep,

$$\pi_{ma} = \{(\pi_{\lambda_0,0}, \pi_{\lambda_1,0}, ..., \pi_{\lambda_{l-1},0}), (\pi_{\lambda_0,1}, \pi_{\lambda_1,1}, ..., \pi_{\lambda_{l-1},1}), ..., (\pi_{\lambda_0,n-1}, \pi_{\lambda_1,n-1}, ..., \pi_{\lambda_{l-1},n-1})\}, \quad (2.10)$$

where $\pi_{\lambda_i,j}$ is the policy for agent λ_i at timestep j, and $\pi_{ma} \in \Pi_{ma}$, where Π_{ma} is the joint team policy space. We additionally assume access to a *no-op* action at each agent's the terminal state that has a duration of 1 timestep and a reward of 0, which ensures that the stationary policy of an agent remaining in its goal state has zero cost. Then, we can formalize the collaborative multiagent planning problem under uncertainty as an optimization over team policies:

$$\begin{aligned} \pi^*_{ma}(x, b) = &\arg\min_{\pi_{ma} \in \Pi_{ma}} \mathop{\mathbb{E}}_{\omega \sim b} \left[\sum_{j=0}^{n-1} c((\pi_{\lambda_0,j}, \pi_{\lambda_1,j}, ..., \pi_{\lambda_{l-1},j}); \Gamma) \right] \\ \text{s.t.} \quad & f(\pi_{\lambda_i,j}; \Gamma) = 1 \ \forall i \in \{0, 1, ..., l-1\} \ \forall j \in \{0, 1, ..., n-1\} \\ & b(\pi_{\lambda_i,j}) = e(\pi_{\lambda_i,j-1}) \ \forall i \in \{0, 1, ..., l-1\} \ \forall j \in \{1, ..., n-1\} \\ & b(a_{\lambda_i,0}) = x_{\lambda_i,s}, e(a_{\lambda_i,n-1}) = x_{\lambda_i,G} \ \forall i \in \{0, 1, ..., l-1\}. \end{aligned} \quad (2.11)$$

Methods for collaborative multiagent planning under uncertainty are highly dependent on the type of uncertainty present in the planning problem and the team objective. While a multitude of solutions exist for different problem types, for teams operating in environments with aleatoric uncertainty, planning methods are largely focused on generating robust plans to maintain plan satisfiability or to optimize a planning objective in expectation. For instance, Messing et al. [100] developed an approach for risk-bounded scheduling in the presence of uncertain action durations; the approach used sampled environment realizations to

generate a plan, and then iteratively guaranteed the robustness of the plan makespan using the Sequential Probability Ratio Test (SPRT). Other approaches developed uncertainty-aware task allocation algorithms; for example, Park et al. [115] used the SPRT to generate risk-bounded task allocations for environments with task requirement and agent capability uncertainties, Neville et al. [110] developed a method for efficient task allocation repair in dynamic scenes, and Prorok [120] developed a bounded-suboptimal greedy approach for redundant task allocation in uncertain environments for cases when maximizing expected plan quality at the expense of additional robot effort is paramount. While these approaches mitigated the effects of uncertainty on planning, they did not use robot actions as a method for reducing problem uncertainty.

Conversely, for problems with epistemic uncertainty, multiagent teams often explicitly plan to reduce environmental uncertainty using exploration-based objectives. For example, solutions have been proposed for centralized [133] and decentralized [53, 80, 81] teams, for teams with full [25] and reduced [53, 6, 79, 80, 81, 14, 13] communication, and for various tasks, including the observation of static [53, 14, 13], dynamic [125, 79], or static and dynamic [80, 81] environments and objects. However, these approaches are largely focused on minimizing a specific type of environmental uncertainty (e.g., minimizing model uncertainty of an environmental phenomenon [53, 14], or minimizing positional uncertainty of a target [125]), and do not attempt to minimize uncertainty that is relevant to future robotics tasks in the environment.

One notable exception is Lee et al. [83], which developed an approach for joint exploration and exploitation in a scout-task tracking problem. In their problem setting, a team of scout, task, and scout-task robots receive reward when task robots observe targets, which have initially unknown positions. Scout robots, which can have a large sensor range, are deployed to find and relay target information to task robots, and task robots use target information to guide planning. The authors developed a mutual-information upper confidence bound for use in a decentralized Monte Carlo tree search, which was used to generate team

plans. They demonstrated high-quality planning as compared to an expecti+max planning approach in a simplified simulation, demonstrated the planner for a four-agent team in two simulated environments, and deployed the planner for a four-agent team in a real-world base environment [83, 84].

In Chapter 4, we develop a method for generating team plans that reduce task-relevant planning uncertainty for the long-horizon navigation task. Specifically, by optimizing a team makespan objective in an environment modeled as a stochastic graph, we generate team plans where agents trade off between individual goal-directed navigation and team-oriented exploration, which improves teammate planning performance by sensing to reduce teammate-task-relevant environment uncertainty.

Chapter 3

Online High-Level Model Estimation for Efficient Hierarchical Robot Navigation

In this chapter, we address the problem of how to enable a robot to navigate efficiently and robustly in large, structured environments during multi-query planning. We formulate the problem as a hierarchical planning problem that uses agent trajectories, computed online, to build an abstract representation of the navigation problem that can be generalized to other planning problems in the same environment. Finally, we discuss two variants of a hierarchical planner that can be used to solve new planning queries using the abstract representation. An example showing the motivation for our approach is shown in Fig. 3.1.

3.1 Motivation

We would like to generate abstract models of global planning information that can be used to enable efficient and robust robot navigation in known, structured environments that are large enough to cause traditional planning approaches to incur considerable computational cost. Hierarchical planners can minimize the *curse of scale*, or the exponential relationship between plan length and planning complexity in regular discrete environments, by generating coarse representations of the planning problem whose solutions can be used to guide a detailed

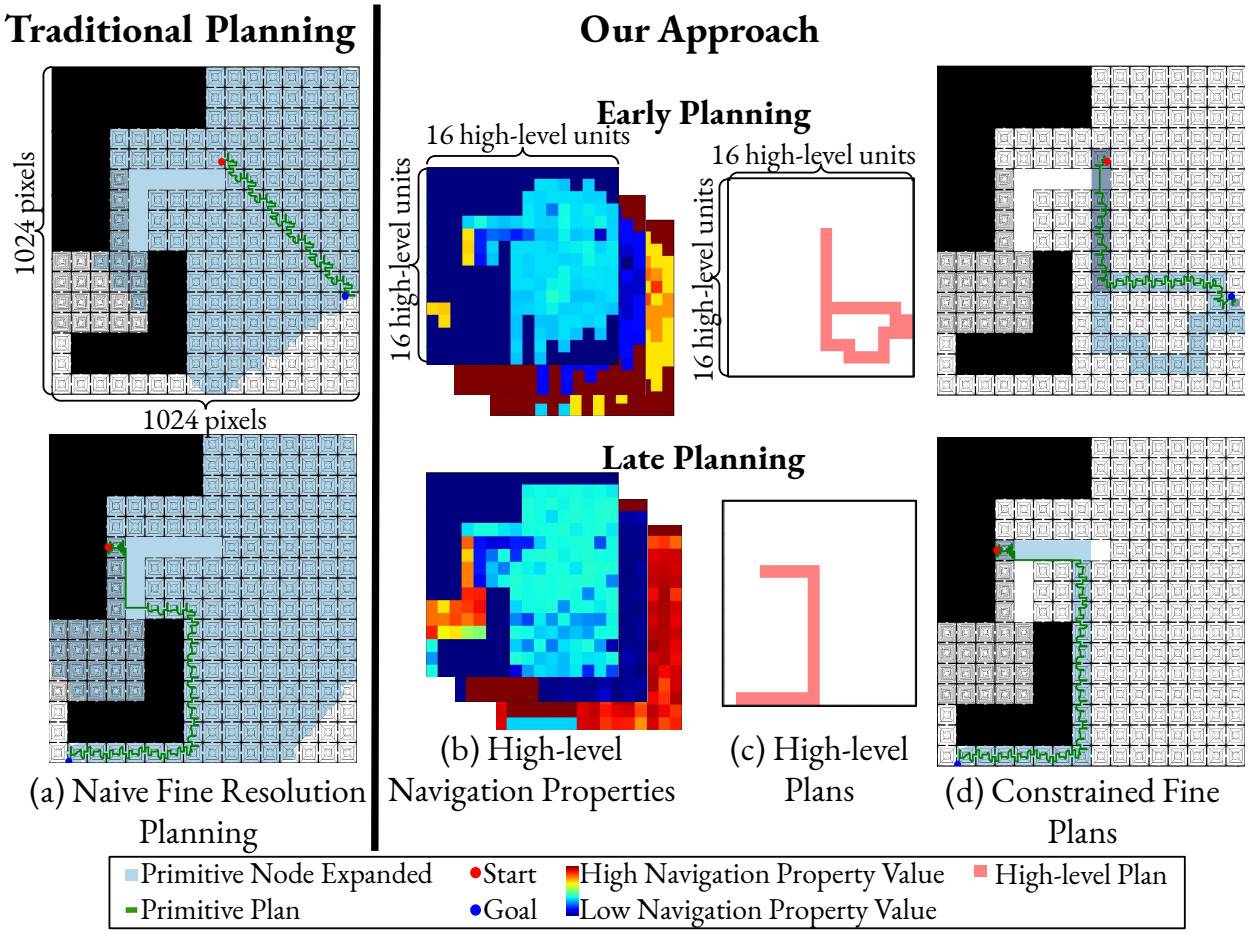


Figure 3.1: **Online high-level model estimation for efficient hierarchical robot navigation.** Rather than planning at a single fine resolution (a), we present an approach that estimates high-level navigation properties (b) online, using only previous planning computations, to find high-level plans (c) in a coarse representation of the planning problem. Then, we use high-level plans to constrain planning in the fine resolution planning problem (d). We demonstrate that our approach increases planner efficiency while ensuring planner robustness over the course of multiple queries to the planner.

planner to regions of the environment that are likely to contain low-cost detailed solutions. However, coarse planning is only useful if the coarse planning problem efficiently captures properties of the environment necessary to inform intelligent detailed planning. Existing hierarchial planning approaches that are guaranteed to accurately summarize plans in the environment [148] can be overly restrictive when the geometry of the environment (e.g., the set of convex regions) is poorly aligned with the geometric features of the environment needed

to compute a feasible motion plan efficiently. Other approaches that use information-based techniques to compress environment representations [147, 76] do not explicitly capture the structure of good plans that agents use to traverse in the environments. Learned approaches for generating abstract representations rely on offline datasets, and often require additional environmental cues, such as semantically segmented overhead images [39] and height maps [69], to enable efficient abstraction.

Additionally, existing hierarchical planners plan using static, coarse representations that are high quality and a good approximation of select environments, but these planners fail to model the lossy nature of plan abstraction for planning in large, complex environments. Fortunately, a number of risk-aware planners have been developed to improve planning performance when the model of a planning problem is known to be incomplete or noisy, such as both offline [103] and online [29] planning over probabilistic costmaps. We hypothesize that explicitly representing uncertainty in coarse or hierarchical representations can enable efficient, robust planning, even when the coarse representation is not initially well aligned with the true costs of plans in the underlying detailed planning environment.

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

3.2 Approach

3.2.1 Models for Hierarchical Planning

Recall that in Sect. 2.4, we defined the hierarchical discrete single-agent optimal planning problem as an optimization over abstract plans $\mathbf{p}$, where abstract plans were defined as sequences of abstract actions $\mathbf{a} \in \mathbf{A}$, $\mathbf{p} = \{\mathbf{a}_0, \mathbf{a}_1, ..., \mathbf{a}_{\mathbf{n}-1}\}$, and the objective of the problem was to minimize the abstract cost $\mathbf{c}(\mathbf{a}; \mathbf{\Gamma})$ of the abstract plan subject to constraints on an abstract feasibility function $\mathbf{f}(\mathbf{a}; \mathbf{\Gamma})$ and the agent start and goal:

$$\begin{aligned} \mathbf{p}^* = \arg\min_{\mathbf{p} \in \mathbf{P}} & \sum_{j=0}^{\mathbf{n}-1} \mathbf{c}(\mathbf{a}_j; \mathbf{\Gamma}) \\ \text{s.t.} \quad & \mathbf{f}(\mathbf{a}_j; \mathbf{\Gamma}) = 1 \ \forall j \in \{0, 1, ..., \mathbf{n}-1\} \\ & \mathbf{e}(\mathbf{a}_{j-1}) \Diamond \mathbf{b}(\mathbf{a}_j) \ \forall j \in \{1, ..., \mathbf{n}-1\} \\ & x_s \in \mathbf{b}(\mathbf{a}_0), x_g \in \mathbf{e}(\mathbf{a}_{\mathbf{n}-1}), \end{aligned} \tag{3.1}$$

where $\mathbf{\Gamma}$ are additional inputs, $\mathbf{b}(\mathbf{a}_j)$ and $\mathbf{e}(\mathbf{a}_j)$ are the set of valid start states and terminal states for high-level action $\mathbf{a}_j$, and $\Diamond$ is a function that tests the connectivity of two high-level actions in a high-level plan.

While the hierarchical planning formulation defined in Eq. 3.1 can result in efficient planning, it is obvious that the quality of the high-level planning model, or the combination of $\mathbf{A}$, $\mathbf{c}(\mathbf{a}; \mathbf{\Gamma})$, and $\mathbf{f}(\mathbf{a}; \mathbf{\Gamma})$, dictates the ability of the high-level search process to effectively guide primitive planning. Intuitively, a good choice of representation is one that ensures *plan similarity*[1], or that low-cost, feasible plans in $\mathbf{P}$ have low-cost, feasible primitive refinements in P. Unfortunately, generating planning representations that exhibit plan similarity can be challenging, and in practice such representations are often hand-designed for specific problems [68], or require significant prior domain knowledge to be generated [69, 39, 148].

[1]This is an extension of *cost similarity* [69] which considers plan feasibility.

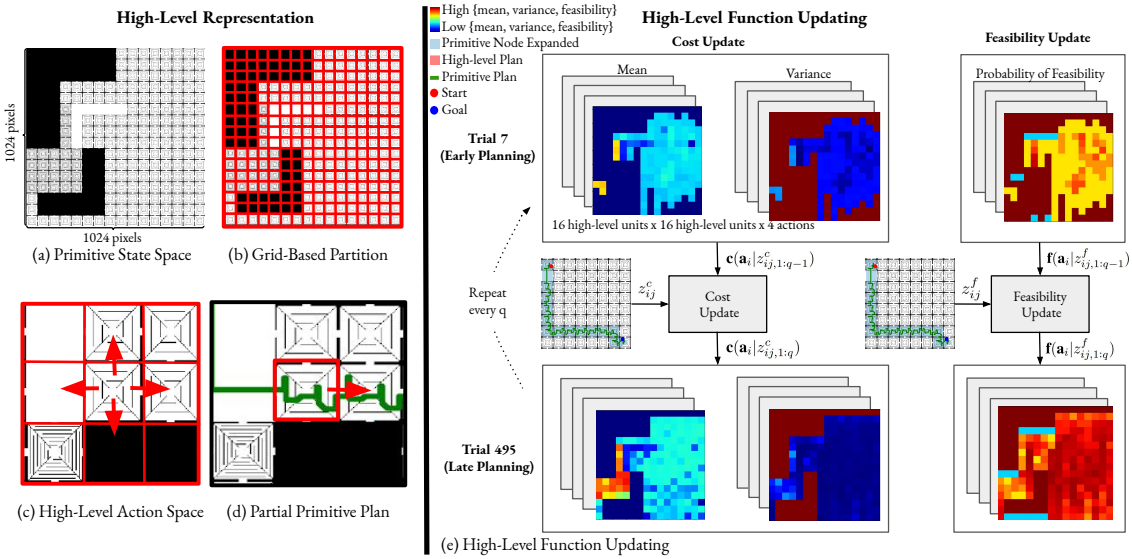


Figure 3.2: **Overview of the approach.** Rather than planning at a single resolution in a large primitive state space (a), we partition primitive state spaces into high-level regions using a grid (b) and define high-level actions as transitions between high-level regions (c). Then, we can treat primitive plans in the environment as example measurements of high-level function properties, like the partial primitive plan in green (d), which shows the left high-level action is feasible and has a cost equal to the plan length in the region. Finally, we use online datasets of primitive navigation properties to estimate high-level functions over time using two recursive estimation techniques, *Averaging* (not shown) and *Bayes Filtering* (e). Results are shown for the *Medium-Risk* planner (see Section 3.3.2).

Instead, we define a simple $\mathbf{A}$ and use online planning results to infer estimates of *high-level navigation properties*, $\mathbf{c}(\mathbf{a}; \mathbf{\Gamma})$ and $\mathbf{f}(\mathbf{a}; \mathbf{\Gamma})$, over the course of a multi-query planning trial. An overview of our approach is shown in Fig. 3.2.

3.2.2 The High-Level Action Space

The goal of the high-level action space is to generate a small (i.e., low-resolution) representation of the planning problem that enables efficient planning without compromising plan quality. In this work, we recognize that different optimal primitive plans in structured environments often share common primitive *subplans*, or partial sequences of primitive actions

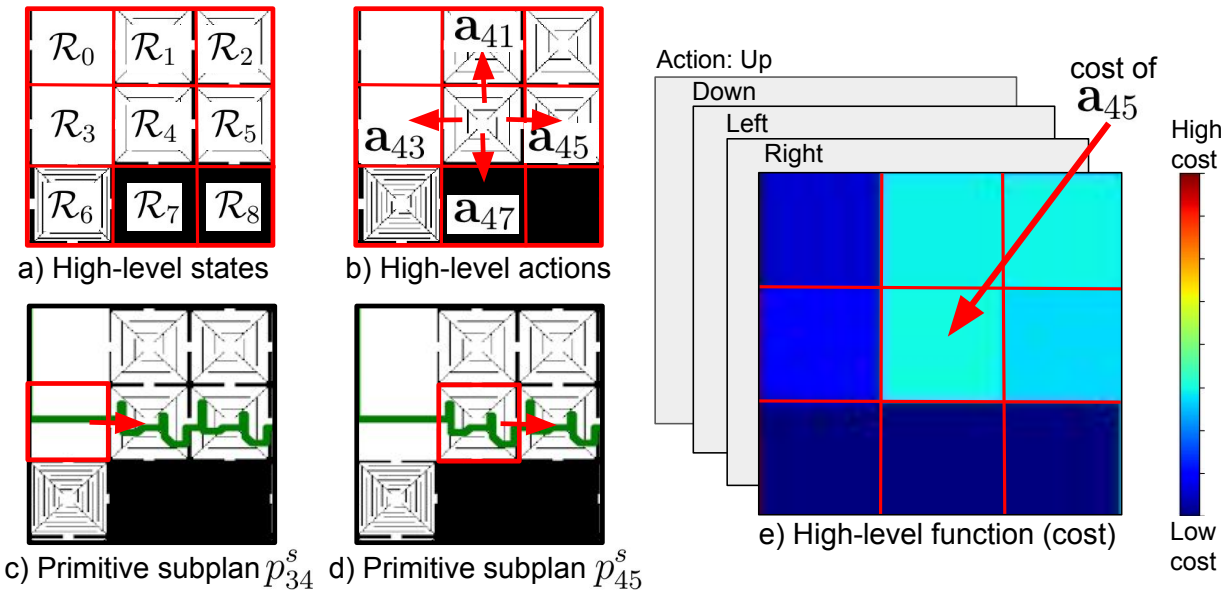


Figure 3.3: **Example regions, actions, primitive subplans, and functions in a simulated outdoor domain.** Our approach generates a simple (a) state- and (b) action-based abstraction, and then treats the properties of primitive subplans in the environment (c-d) as *samples* of (e) high-level function properties. Over time, the high-level functions are updated to increase *plan similarity*; here we show the high-level cost function for the right action, generated using the *Medium-risk* planner (see Section 3.3.2).

$a_{l:m}$, when navigating through the same region in the environment. By grouping these subplans into high-level actions $\mathbf{a}$, we can generate a compact, flexible high-level action space $\mathbf{A}$ that uses the primitive navigation properties of similar, past planning queries to improve the high-level estimates of navigation properties for use in current and future planning queries.

Formally, we define the high-level action space by assuming that the configuration space $\mathcal{X}$ can be decomposed into a finite set of regions[2] $\mathcal{R}$ that an agent can traverse between, and we define a high-level action $\mathbf{a}_{ij}$ as the set of primitive subplans that begin in and remain in region $\mathcal{R}_i$ until terminating in region $\mathcal{R}_j$, similar to the definition given by Vega-Brown and Roy[3] [148],

$$\mathbf{a}_{ij} = \{p | p \in P,\ a_k \in p,\ b(a_k) \in \mathcal{R}_i\ \forall k \in \{0, 1, ..., n-1\}, e(a_{n-1}) \in \mathcal{R}_j\}. \tag{3.2}$$

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

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

Example regions and high-level actions are visualized in Fig. 3.3-a-b.

3.2.3 Primitive Subplans as Instances of High-Level Actions

By representing high-level actions as sets of primitive plans, we can use results of planning in the primitive action space $\mathcal{A}$ (e.g., the occupancy map) to induce costmaps over the high-level action space $\mathbf{A}$. When we generate a primitive refinement p of high-level plan $\mathbf{p}$, where $p \in P$, we observe a sequence of primitive actions in the environment, $p = \{a_0, a_1, ..., a_n\}$, where $a_i \in \mathcal{A}$, subject to the high-level planning constraints, and we also observe whether or not the refinement satisfies the primitive planning constraints in Eq. 2.2. However, because we define high-level actions $\mathbf{a} \in \mathbf{A}$ as sets of primitive subplans, we can also post-process a primitive refinement p into a sequence of primitive subplans, $p = \{p^s_0, p^s_1, ..., p^s_{\mathbf{n}}\}$, where each subplan p^s is a sequence[4] of primitive actions $a_{l:m}$ that obeys the constraints of a high-level action $\mathbf{a}$ as defined in Eq. 3.2,

$$p^s_{ij} = \{a_{l:m} | a_k \in p,\ b(a_k) \in \mathcal{R}_i\ \forall k \in \{l, l+1, ..., m\}, e(a_m) \in \mathcal{R}_j\}. \tag{3.3}$$

We visualize two example primitive subplans in Fig. 3.3-c-d. Our formulation allows us to treat a primitive subplan p^s_{ij} collected in an environment as an example high-level action $\mathbf{a}_{ij}$ that transitions from region $\mathcal{R}_i$ to $\mathcal{R}_j$. We can then consolidate example high-level actions into a dataset for inference of high-level function values. We will also refer to the primitive subplan p^s_{ij} as an *instance* of a high-level action[5] $\mathbf{a}_{ij}$.

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

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

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

over all primitive a in example action p^s_{ij},

$$c(p^s_{ij}) = \sum_{a \in p^s_{ij}} c(a). \tag{3.4}$$

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

$$f(p^s_{ij}) = \min_{a \in p^s_{ij}} f(a), \tag{3.5}$$

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

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

$$\mathcal{D} = \{(p^s_{ij}, c(p^s_{ij}), f(p^s_{ij})), (p^s_{jk}, c(p^s_{jk}), f(p^s_{jk})), ...\}. \tag{3.6}$$

3.2.4 The High-Level Functions

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

$$\mathbf{c}(\mathbf{a}_{ij} | c((p^s_{ij})_{1:q})) = g_c(\mathcal{D}_{ij}) \tag{3.7}$$

$$\mathbf{f}(\mathbf{a}_{ij}|f((p^s_{ij})_{1:q})) = g_f(\mathcal{D}_{ij}), \tag{3.8}$$

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

$$z^c_{ij} = c(p^s_{ij}) \tag{3.9}$$

$$z^f_{ij} = f(p^s_{ij}), \tag{3.10}$$

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

3.2.5 Online High-Level Function Updating as Recursive Estimation

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

Averaging

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

$$\mathbf{c}(\mathbf{a}_i|z^c_{ij,1:q}) = \mathbf{c}(\mathbf{a}_i|z^c_{ij,1:q-1}) + \frac{z^c_{ij} - \mathbf{c}(\mathbf{a}_i|z^c_{ij,1:q-1})}{q} \tag{3.11}$$

$$\mathbf{f}(\mathbf{a}_i|z^f_{ij,1:q}) = \mathbf{f}(\mathbf{a}_i|z^f_{ij,1:q-1}) + \frac{z^f_{ij,q} - \mathbf{f}(\mathbf{a}_i|z^f_{ij,1:q-1})}{q} \tag{3.12}$$

where $\mathbf{c}(\mathbf{a}_i|z^c_{ij,1:q})$ and $\mathbf{f}(\mathbf{a}_i|z^f_{ij,1:q})$ are scalars, and q is the number of observed measurements of the high-level functions of abstract action $\mathbf{a}_i$.

Bayes Filtering

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

$$\Pr(\mathbf{c}(\mathbf{a}_{ij})|z^c_{1:q}) \propto \Pr(z^c_q|\mathbf{c}(\mathbf{a}_{ij}))\Pr(\mathbf{c}(\mathbf{a}_{ij})|z^c_{1:q-1}) \tag{3.13}$$

$$\Pr(\mathbf{f}(\mathbf{a}_{ij})|z^f_{1:q}) \propto \Pr(z^f_q|\mathbf{f}(\mathbf{a}_{ij}))\Pr(\mathbf{f}(\mathbf{a}_{ij})|z^f_{1:q-1}). \tag{3.14}$$

In practice, we find that simple modeling decisions can lead to uncertainty-aware high-level functions that enable robust planning. In the experiments that follow, we model the

cost function belief distribution and cost measurement probability distribution as Gaussian distributions, an approach used by the planning under uncertainty community [116, 103]. We also model the feasibility function belief distribution as a Beta distribution, and the feasibility function measurement probability distribution as a Bernoulli distribution. By selecting distributions that are conjugate, we ensure the computational efficiency of the update step. Additionally, we assume that the variance σ^s of the cost measurement process for the measurement z^c can be modeled as a hand-tuned constant. We note that we initially assume uninformative priors for $\mathbf{c}(\mathbf{a}_{ij})$ and $\mathbf{f}(\mathbf{a}_{ij})$ (high-variance and uniform, respectively).

3.2.6 Hierarchical Planning using High-Level Functions Estimated Online

Finally, we discuss the hierarchical planner used to find primitive solutions to the discrete single-agent planning problem in Eq. 2.2, guided by solutions to the hierarchical planning problem in Eq. 3.1, which are calculated using the high-level representation developed in Sections 3.2.2 to 3.2.5. At its core, the hierarchical planner is a multi-level graph-based forward search algorithm that iterates between uncertainty-aware planning to generate low-cost solutions to the high-level optimization and primitive planning to optimize the primitive optimization. Planning begins in the high-level graph, where we use forward search to optimize the objective in Eq. 3.1. If the high-level cost function is modeled by a distribution, we use the expected cost of the high-level actions in the optimization. Similarly, the high-level traversability function returns 1 if the probability of a high-level action being traversable is greater than a threshold γ, and 0 otherwise. Nodes are expanded according to a priority queue until a goal node is expanded and a high-level plan $\mathbf{p}$ is recovered. Once $\mathbf{p}$ is found, we constrain the search space of the primitive graph to contain only the subgraph of primitive edges that could be possible refinements of the high-level plan $\mathbf{p}$ by temporarily marking all other edges as infeasible. We search in the constrained graph until a primitive plan p is found, or until all reachable nodes in the subgraph have been expanded. If the primitive

plan is feasible and lower cost than existing primitive plans, it is stored. Then, the result of the constrained primitive planning problem is converted into the dataset in Eq. 3.6 using the method in Section 3.2.3, and the high-level functions are updated according to Section 3.2.5. Planning returns to the high level graph, and continues until a termination condition is met. However, because the costs of high-level graph edges can change over time and may be probabilistic, some modifications to standard forward search are required for the high-level planner.

Queue Prioritization

For both the averaging and filtering approaches, we order nodes v on the high-level queue according to the A* evaluation function $f^\dagger$ [50],

$$f^\dagger(v) = h_{cc}(v) + h_{cg}(v) \tag{3.15}$$

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

$$h_{cc}(v) = \mathbb{E}[\mathbf{c}(\mathbf{p})] = \sum_{\mathbf{a}_{ij} \in \mathbf{p}} \mu_{ij}, \tag{3.16}$$

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

$$\mathbf{f}(\mathbf{p}) = \arg\min_{\mathbf{a}_{ij} \in \mathbf{p}} \mathbf{f}(\mathbf{a}_{ij}) \geq \gamma \tag{3.17}$$

Plans with feasibility values less than γ are *deferred* (see Section 3.2.6).

Termination Conditions

Because the costs and feasibilities of high-level graph-edges are uncertain, we cannot assume that we have found the lowest cost high-level plan after expanding a goal node in the high-level graph. For the averaging approach, we choose to terminate when the known cost of the current best primitive plan p is lower than the A* evaluation function value of the lowest-cost node on the high-level queue, $f^{\dagger}(v)$. For the filtering approach, we use the risk-aware termination condition defined by Pearl and Kim [116], which terminates when the expected risk of terminating with the current best primitive plan p with cost $c(p)$ is less than a risk threshold δ. We calculate the expected risk of terminating with a primitive plan cost $c(p)$ and an unexpanded node corresponding to high-level plan $\mathbf{p}_i$ on the queue,

$$R(c(p), \mathbf{p}_i) = \int_{\tau=-\infty}^{c(p)} (c(p) - \tau) p(\tau | \mu_i, \sigma_i) d\tau, \tag{3.18}$$

where μ_i and σ_i are the mean and variance of the high-level plan $\mathbf{p}_i$. Then, rather than selecting an absolute maximum risk threshold, which may vary with plan length, we normalize the risk value based on the current best primitive plan cost and select a risk threshold δ as a percentage of the normalized risk value, as in Pearl and Kim [116]. Finally, we terminate planning when the maximum normalized risk of terminating with any of the unexpanded high-level plans $\mathbf{p}_i$ on the queue is less than δ,

$$\frac{R(c(p), \mathbf{p}_i)}{c(p)} < \delta. \tag{3.19}$$

Soft Pruning of Dominated Nodes from a Priority Queue with Uncertain Costs

In deterministic graph search, planners rely on admissible heuristics and visited lists to ensure that graph nodes are expanded exactly once, and that when a node is expanded, the optimal cost-to-come for the node is recorded. However, when the costs of nodes on a

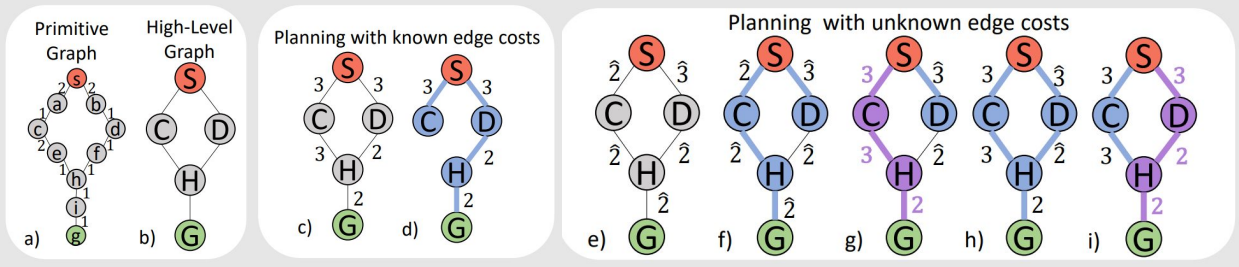


Figure 3.4: **Planning in graphs with known and unknown edge costs.** The goal of the problem is to generate a path from the red start node to the green goal node using Dijkstra's algorithm. The true costs of navigating in the environment are given in the primitive graph (a), and the high-level graph (b) represents planning in the proposed high-level action space, where each region is a node and each high-level action is an edge. When the costs of high-level actions are known, edge costs are deterministic (c), and expanding nodes (blue shading) in order of cost-to-come ensures that the first expansion of the goal node is optimal (d). When the costs of high-level actions are unknown and therefore approximated (represented by the $\hat{}$ symbol, as in (e)), the first expansion of the goal node (f) may not be optimal after plan refinement ((g), where the refined plan is shaded purple). Instead, additional paths to the goal must be expanded and refined (h-i) to ensure search optimality.

queue are uncertain, we cannot assume that the first expansion of a node is optimal (or even feasible) in terms of cost-to-come, as demonstrated in Fig. 3.4. In uncertain environments, this implies that we must maintain all possible paths to all possible nodes in a queue to avoid pruning a path that may be optimal, but this is computationally infeasible. Instead, we introduce the notion of a *dominated* plan, or a plan that is suboptimal given the current beliefs of plan costs and feasibilities, and store dominated plans in a list of *deferred* plans, similar to Vega-Brown and Roy [148], from which plans are not considered for expansion. Then, when the high-level functions are updated, we reevaluate the navigation properties of plans on the priority queue and in the dominated list, and move plans between the queue and the list as necessary. This technique allows the priority queue to focus search on plans in the environment that are likely to lead to low cost high-level solutions, without compromising the ability to reconsider previously high-cost high-level plans, if high-level function values change.

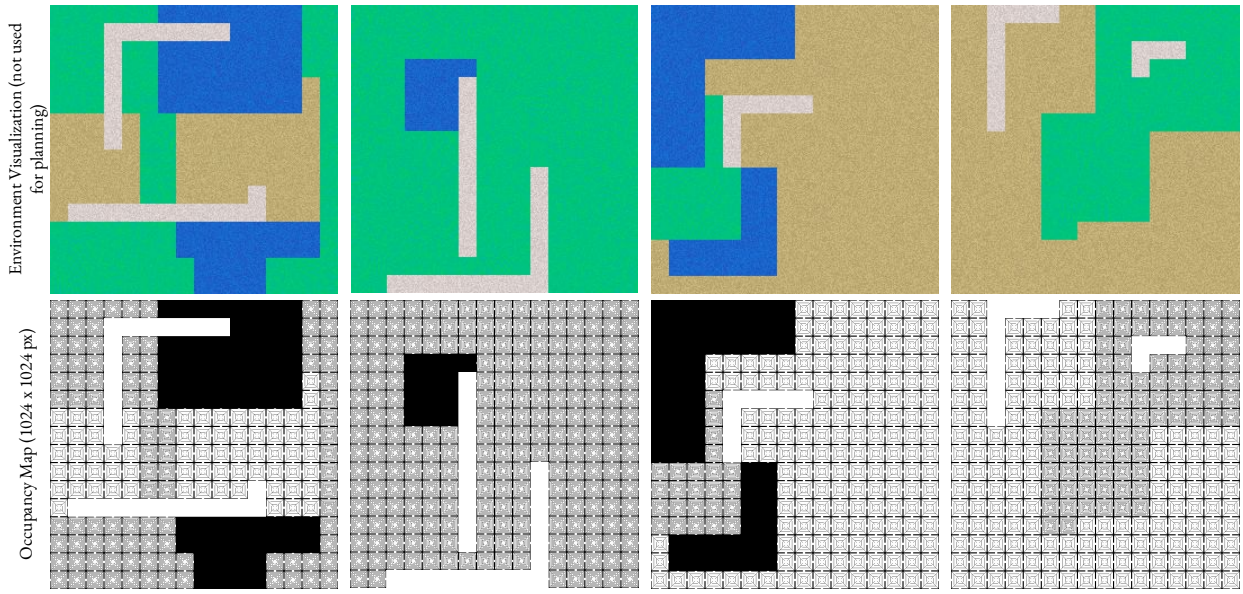


Figure 3.5: **Example overhead images and occupancy maps from the simulated outdoor environment.** Note that the overhead images are used for visualization only, and are not considered during planning.

3.3 Experiments

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

3.3.1 Experimental Setup

We demonstrated our approach in a simulated outdoor environment composed of 100 1024x1024 pixel maps of randomly generated environments consisting of four terrain elements: pavement, sand, forest, and water. Each terrain element was associated with a 64x64 pixel 2D occupancy map, resulting in environments with 256 terrain elements. Terrain element occupancy maps were randomly rotated and flipped to create more diverse scenes.

The environment was modeled such that all terrain elements except water were traversable, and on average, due to the geometric structure of the environment, it was least expensive to navigate in a pavement element and most expensive to navigate in a forest element. Example

occupancy maps and simulated overhead images of the environments are shown in Fig. 3.5; note that the overhead images are for visualization only, and are not used during planning. We assumed a holonomic vehicle with a discrete action set: {up, down, left, right} one pixel with no pose estimate noise. To generate the hierarchical action space, we discretized each environment into 256 equally sized, non-overlapping regions (64x64 pixels per region), resulting in a high-level action space with 1024 actions.

3.3.2 Simulation Evaluation Results

To evaluate our approach, we simulated a robot completing sequences of 500 randomly generated feasible planning tasks in each of the 100 environments using three variants of the hierarchical approach. The first variant (*Average*) used the averaging approach in Section 3.2.5 to update the high-level functions, and used the averaging termination condition in Section 3.2.6. The second (*Medium-Risk*) and third (*High-Risk*) variants of the approach used the Bayes filtering approach in Section 3.2.5 to update the high-level functions, and used the risk-aware termination condition in Section 3.2.6 with medium ($\delta = 0.5$) and high ($\delta = 1.0$) risk thresholds, respectively. All hierarchical planners used the feasibility threshold $\gamma = 0.5$, and the risk-aware planners used the constant variance $\sigma^s = 0.1$. If any of the hierarchical planners expanded 10,000 high-level nodes during the course of a planning trial, the planner timed out and the current lowest cost primitive plan was returned; this occurred for a single *Average* trial. Additionally, one *High-Risk* planning trial failed to find a plan; this trial was removed from aggregate metrics. We compared each variant of our approach to a non-hierarchical planner (*Primitive*) which generated optimal plans by running the A* search algorithm over the primitive, unconstrained occupancy map. We evaluated planner efficiency by comparing the total number of high-level and primitive nodes expanded and the wallclock time taken by the different planners, and we evaluated planner robustness by comparing the final plan costs of the different planners.

We demonstrated that the online hierarchical planners were capable of increasing plan-

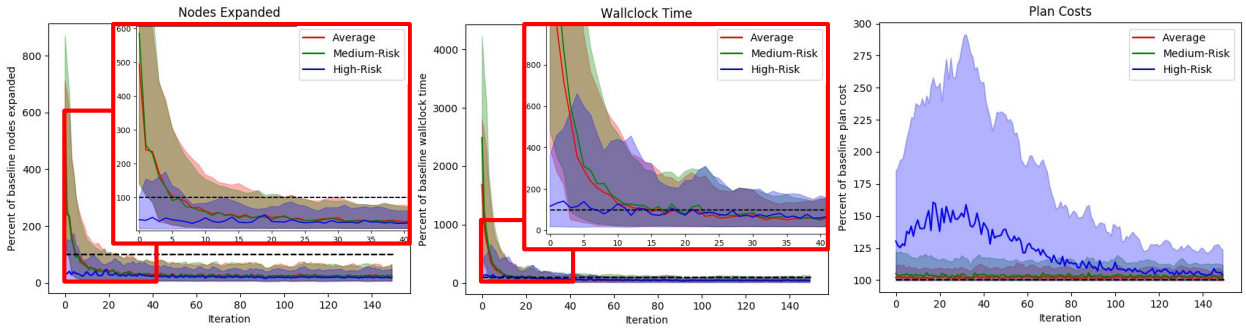


Figure 3.6: **Planning metrics for the hierarchical planners over time as a percentage of *Primitive* planning metrics.** Mean values are plotted as solid lines, and the 5th-95th interquartile range, smoothed using a 4-iteration sliding window for improved visualization, is shaded. We show that all hierarchical planners resulted in increased planning efficiency over time, demonstrated by a decreased number of nodes expanded (a) and decreased wallclock time (b) as compared to the *Primitive* baseline (black dashed line). We also demonstrate that the approaches achieved planner robustness over time, demonstrated by a reduction in plan costs or a maintenance of low-cost plans over time (c). Finally, we show that the risk-based planning formulation enabled risk-aware planning during early planning; for example, the conservative *Average* planner (red) was inefficient during early trials but found low-cost plans, while the *High-Risk* planner (blue) was efficient but found high-cost plans.

ning efficiency while minimizing loss of plan robustness for multi-query planning problems. Planning performance metrics for the hierarchical planners are compared over time in Fig. 3.6 as percentages of *Primitive* performance metrics[6]. In Table 3.1, we compare the total number of nodes expanded, the total wallclock time, and the total plan cost accumulated by the planners during all trials. Across all planning trials, planning with the online representations and hierarchical planners resulted in a 77-86% decrease in the total number of nodes expanded and a 45-66% decrease in the total wallclock planning time as compared to the *Primitive* baseline while incurring only a 2-10% increase in plan costs. In Fig. 3.7, we visually compare high-level cost functions and solutions over time for each of the high-level planners.

We also showed that the risk-based formulation enabled the planner to balance between planning efficiency and robustness in a principled manner during early planning, which we

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

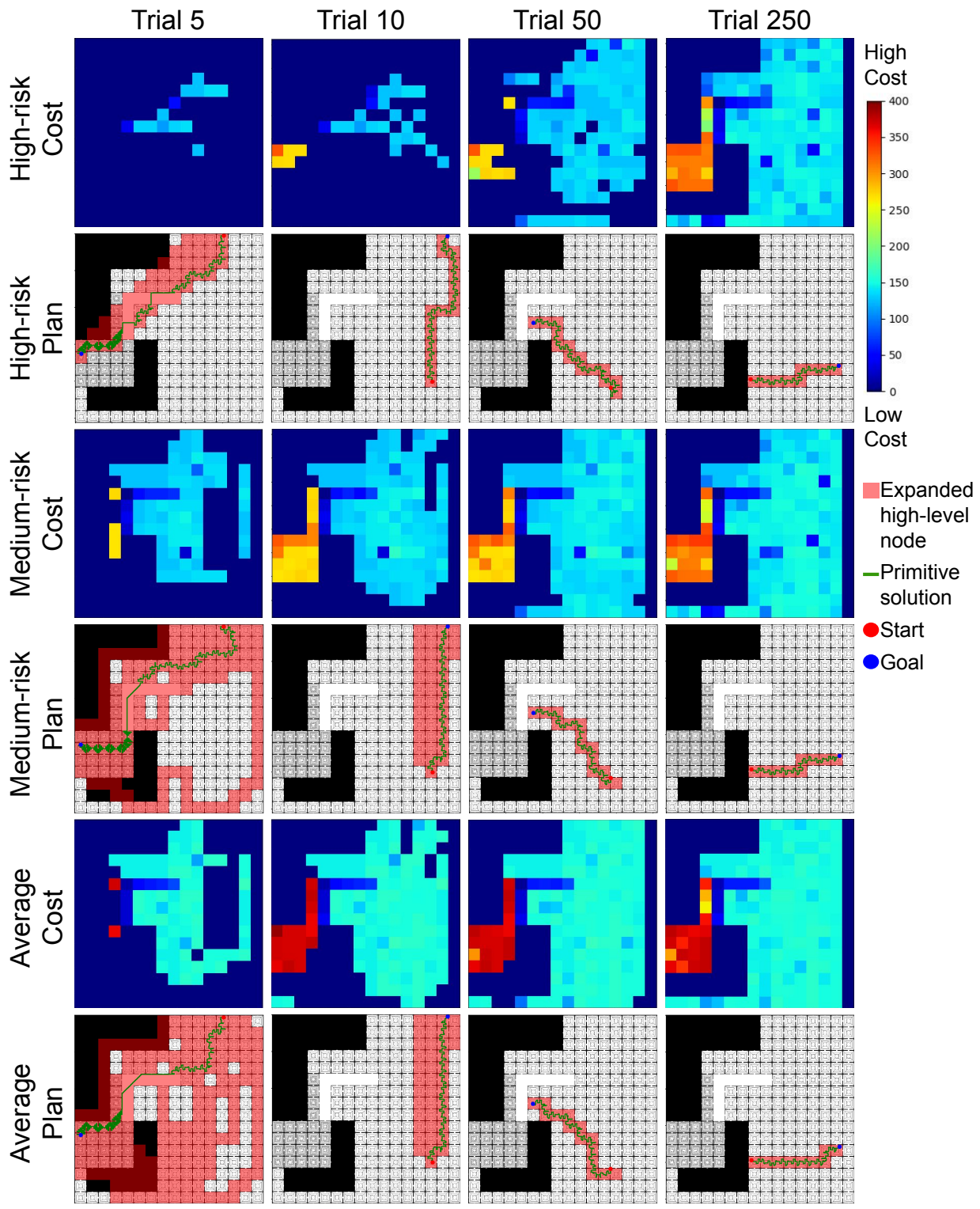


Figure 3.7: **High-level cost functions for the 'right' action and planning solutions for the hierarchical planners over time.** For lower risk trials, agent cost functions exhibit a higher degree of plan similarity and planners expand more high-level nodes (pink squares) during early planning, as the agent explores the environment to reduce uncertainty, and therefore risk. Over time, all planners generate plan similar cost functions, and the agents expand a small number of high-level nodes while finding low-cost solutions (green line).

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

Table 3.1: **Planning performance for the hierarchical planners during different stages of planning.** Metrics were calculated by summing the nodes expanded, wallclock time, and plan costs for all trials in a planning stage; early planning was defined as the first 100 iterations, while late planning was defined as the remaining 400 iterations. Total metrics were the sum of plan metrics for all planning iterations. We demonstrated that the hierarchical planners were efficient and robust over time, and that the risk-aware planner could be used to elicit different balances between planner efficiency and robustness during early planning.

defined as the first 100 planning iterations. We computed planning metrics for early planning only; results are shown in Table 3.1. As expected, the *Average* planner demonstrated the most conservative behavior of the planners, while the *High-Risk* planner demonstrated the most risky behavior. For example, the *High-Risk* planner was most efficient, and expanded 17% of the nodes and took 51% of the time as the *Primitive* planner during early planning, but found plans that were 31% more expensive. Conversely, the average planner was the most robust of the hierarchical planners during early planning, and found plans that were only 2% more expensive than the optimal *Primitive* planner; however, *Average* was less efficient, expanding 40% of the nodes and taking 106% of the time as compared to *Primitive* during early planning. The *Medium-Risk* planner achieved an intermediate risk profile as compared to the other high-level planners, and expanded 36% of the nodes as compared to *Primitive*, while incurring 104% of the plan cost. However, due to the cost of maintaining and using the risk-based representation in a risk-averse way, *Medium-Risk* was slower than the other planners, finding plans in 112% of the time as *Primitive*.

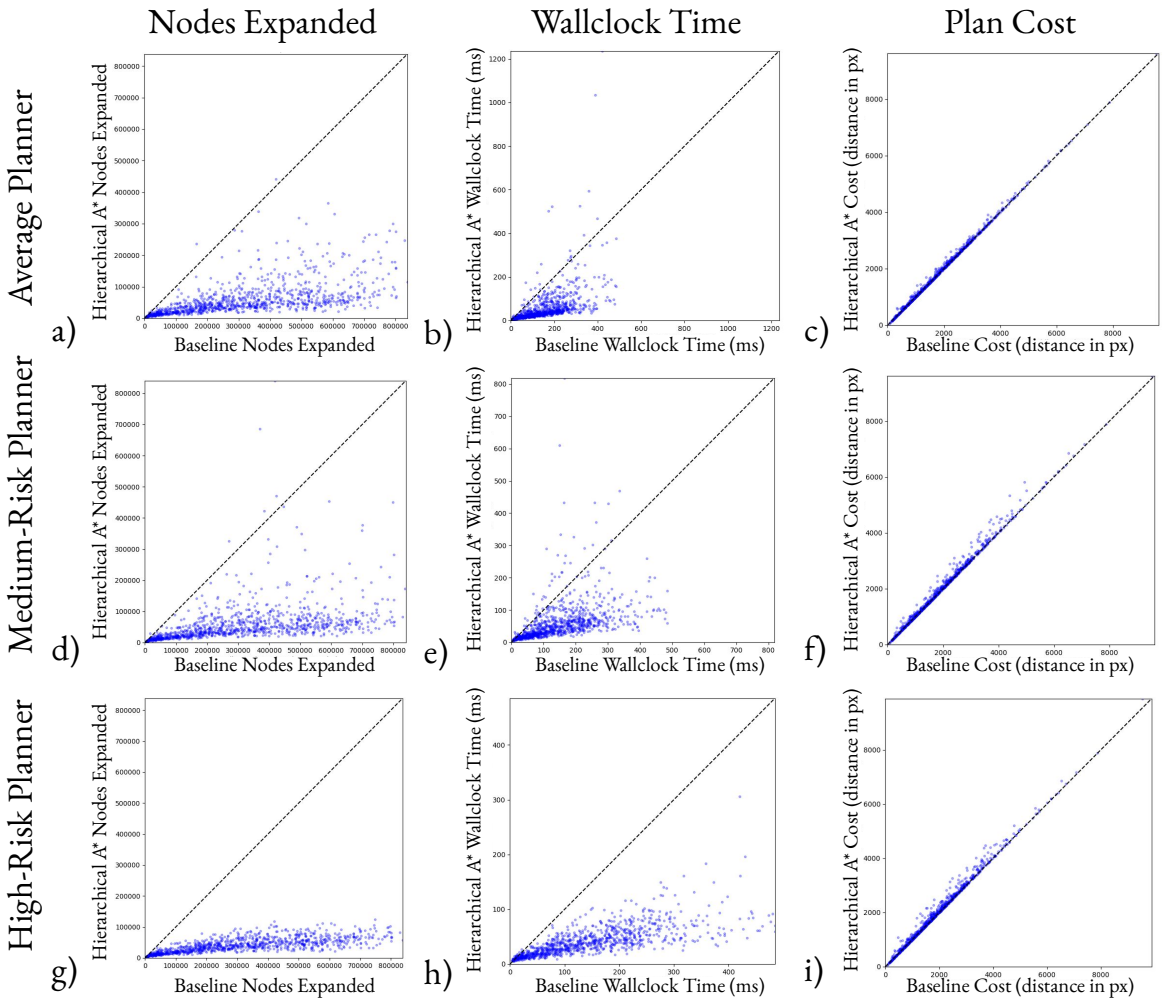


Figure 3.8: **Scatter plots of steady state planning results.** Scatter plots of nodes expanded, wallclock time in milliseconds, and plan cost in simulation units for (a-c) *Medium-Risk* vs. *Primitive*, (d-f) *Medium-Risk* vs. *Primitive* and (g-i) *High-Risk* vs. *Primitive* (blue circles) for the last 10 queries in each of the 100 environments. The black dotted line is an equal value reference.

We also analyzed the late planning performance of the high-level planners, which we defined as planner performance for the last 400 queries of a 500 query trial. Performance metrics are given in Table 3.1; Fig. 3.8 includes example scatter plots comparing nodes expanded, wallclock time, and plan costs for the *Average*, *Medium-Risk*, and *High-Risk* planners as compared to the *Primitive* planner for the last 10 queries in each of the 100 planning environments. Additionally, example solutions found using the *Medium-Risk* planner during

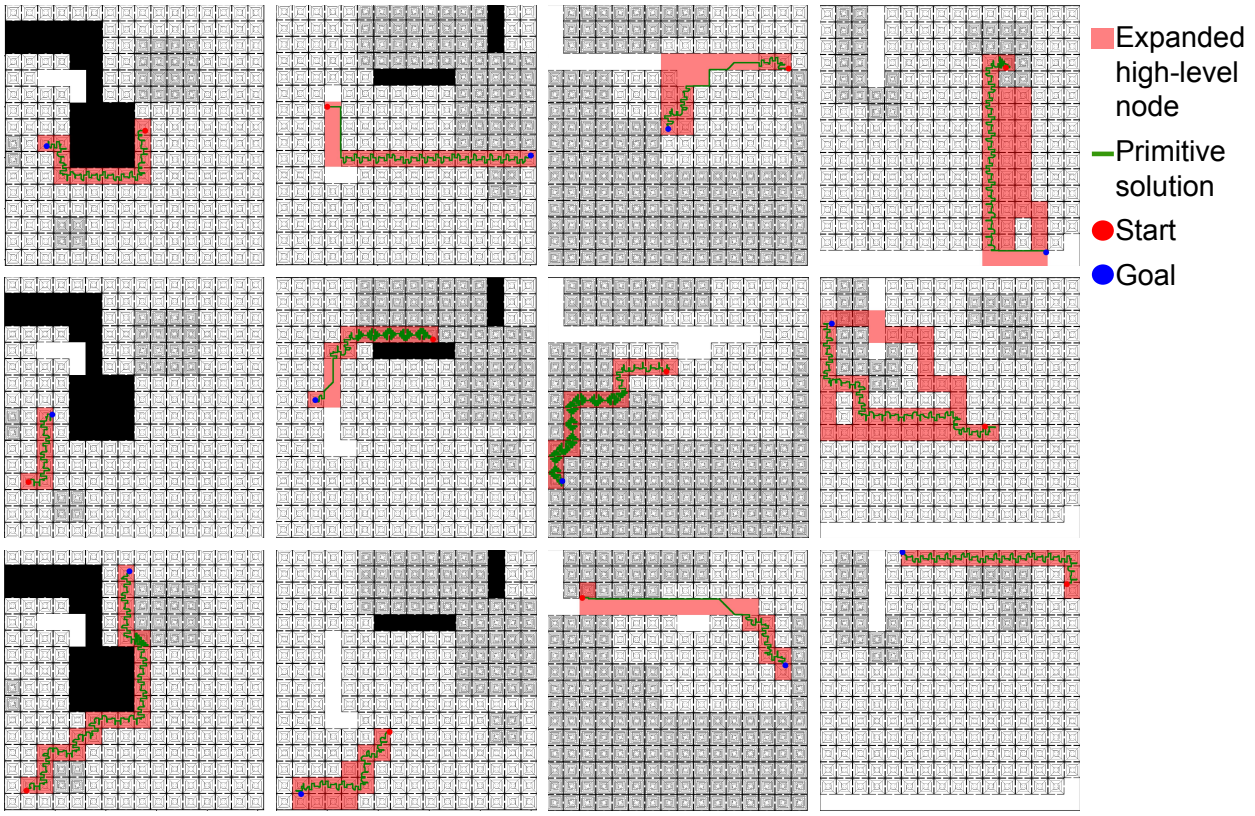


Figure 3.9: **Example *Medium-Risk* solutions during late planning.** At this stage of planning, the high-level functions exhibit a high degree of plan similarity, and the agents quickly find low-cost high-level plans and primitive solutions that navigate around untraversable water regions, avoid high-cost forest regions, and use low-cost road regions to reduce traversal costs. High-level nodes expanded during the planning trials are visualized in pink, and final primitive solutions are shown as green lines.

the last 11 queries in 4 planning environments are shown in Fig. 3.9. We demonstrated that the hierarchical approaches expanded 13-19% of the nodes and executed in 30-41% of the wallclock time as compared to the *Primitive* baseline while incurring only a 2-4% increase in plan cost during late planning. These results indicate that the online representations and hierarchical planners are capable of significantly improving planning efficiency during late planning as compared to the baseline while minimizing loss of planner robustness, as demonstrated by the minimal increase in plan cost.

Finally, we directly evaluated the plan similarity of high-level and primitive plans over time to analyze the ability of the high-level representation to accurately represent primitive navigation properties. When a high-level plan was refined using the primitive planner, we

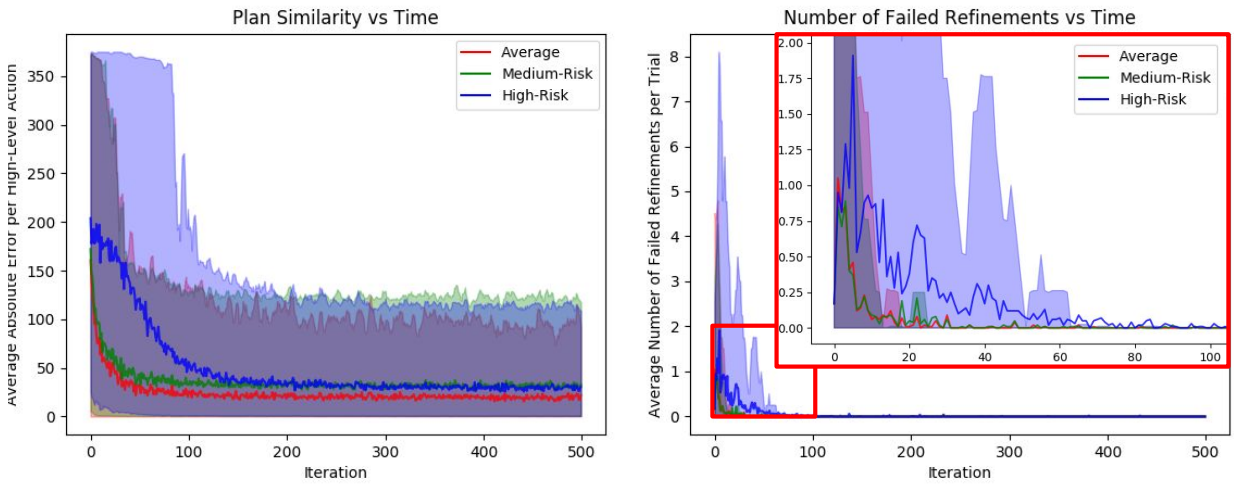


Figure 3.10: **Plan similarity between high-level and primitive plans over time.** Average absolute error between predicted high-level plan costs and primitive plan costs and the average number of failed refinements over time for the three hierarchical planners. Mean values are plotted as solid lines, and the 5th-95th interquartile range, smoothed using a 4-trial sliding window, is shaded. Average absolute error and the average number of failed refinements decreased over time for all planners, indicating that the online planners were capable of learning useful high-level representations over time; *High-Risk* converged least quickly, as it prioritized efficient termination over exploration in the environment.

compared the predicted navigation properties of each high-level action $\mathbf{a}_{ij}$ to the measured navigation properties p^s_{ij} of the resulting example primitive plan. We compared the average absolute error per high-level action and the number of incorrect feasibility assignments for the different planners in Fig. 3.10. All planners demonstrated a decrease in average absolute cost error and number of incorrect feasibility assignments over time. Additionally, we noted that high-level function error decreased most slowly for the *High-Risk* planner, which was expected given that *High-Risk* prioritizes efficient planning over planner robustness during early planning. Overall, we demonstrated that the high-level representations were able to use online planning results to increase plan similarity over multi-query planning trials. In Figs. 3.11, 3.12, and 3.13, we visualize the high-level functions for the *Average*, *Medium-Risk* and *High-Risk* planners over time, respectively.

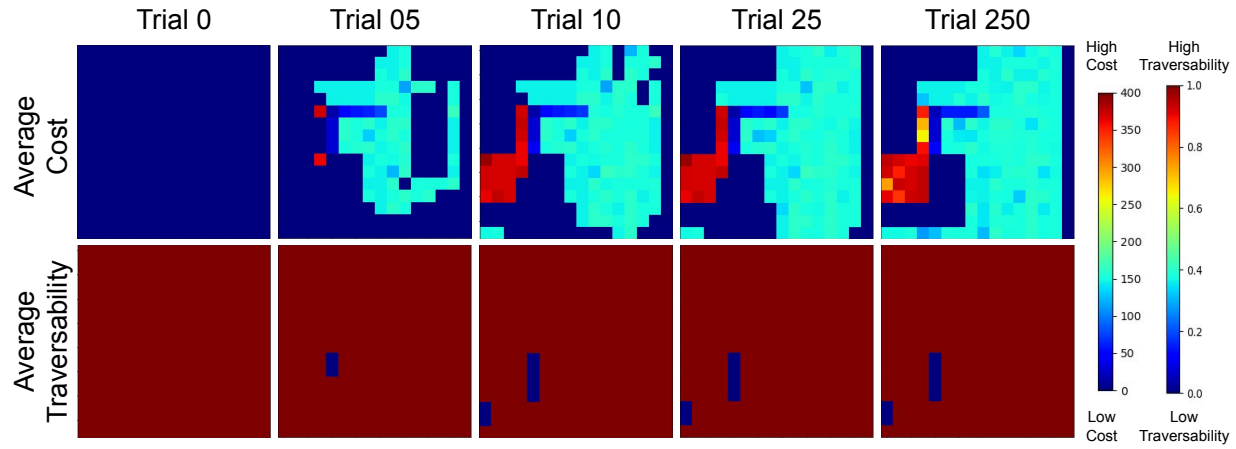


Figure 3.11: **Visualizations of the *Average* high-level functions for the 'right' action over multiple planning queries.** The functions quickly converged to a plan similar representation.

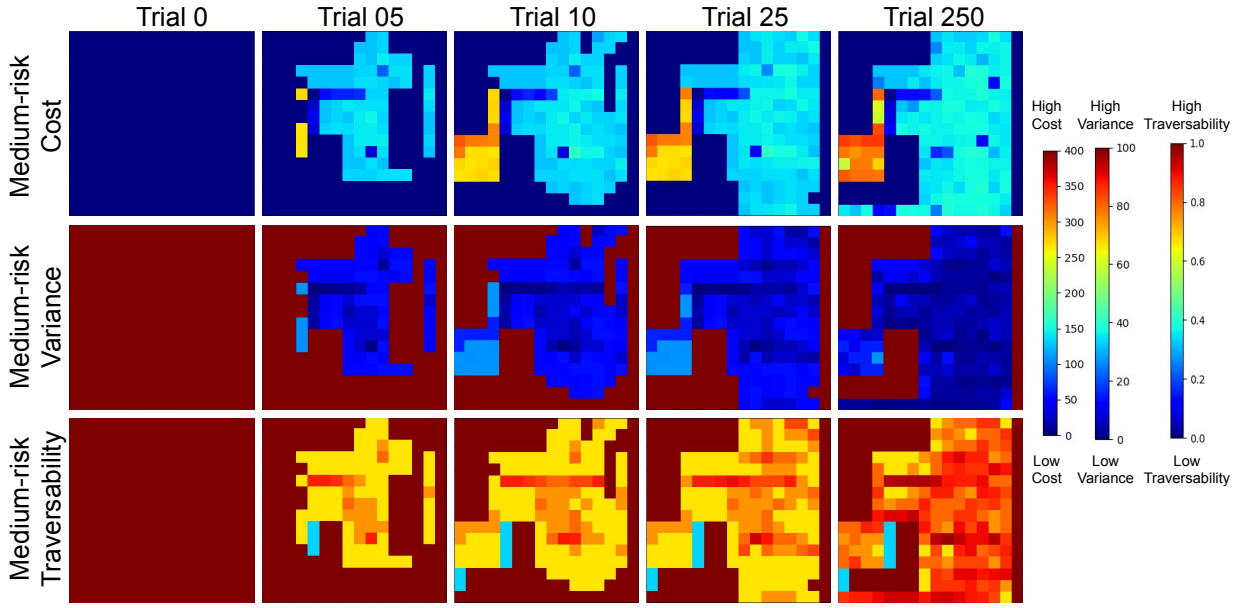


Figure 3.12: **Visualizations of the *Medium-Risk* high-level functions for the 'right' action over multiple planning queries.** The functions quickly converged to a plan similar, uncertainty-aware representation.

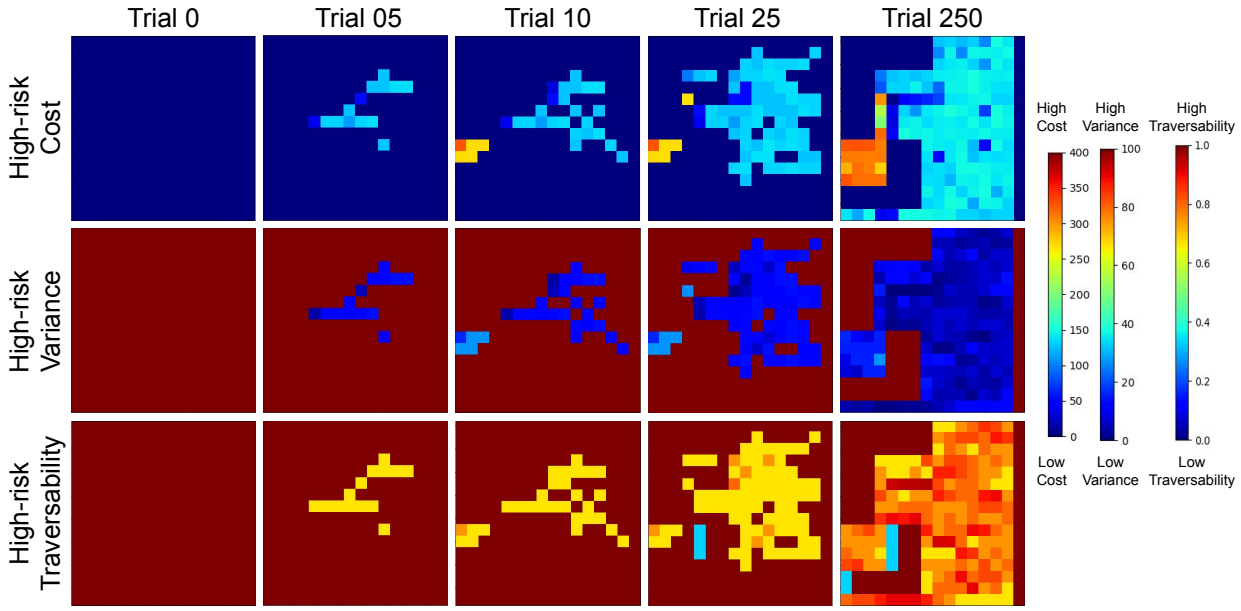


Figure 3.13: **Visualizations of the *High-Risk* high-level functions for the 'right' action over multiple planning queries.** The functions slowly converged to a plan similar, uncertainty-aware representation.

3.4 Conclusion

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

Chapter 4

Approximating the Value of Collaborative Team Actions for Efficient Multiagent Navigation in Uncertain Graphs

While the hierarchical planning representation presented in Chapter 3 was efficient for multi-query planning in known environments, it required an agent to have access to a global occupancy map. In practice, such a map is likely intractable to generate prior to planning in large, uncertain environments. Additionally, the approach relied on a grid-based discretization of the environment. These limitations minimized the applicability of the approach to planning in uncertain environments.

In this chapter, we develop a collaborative multiagent planning formulation for navigation on CTP graphs to solve the multiagent planning under uncertainty optimization in Eq. 2.11. The CTP graph represents global traversability uncertainty, and our approach is designed to enable efficient, uncertainty-aware collaborative multiagent planning on the CTP graph. Specifically, we modeled collaborative team policies for CTPs using macro-actions. We

reduced the number of macro-actions considered during planning for some team types by generating optimistic approximations of candidate future team states, and then generated a small policy class for planning that only included likely high-reward macro-actions. We evaluated our approach in toy and island road network domains, and demonstrated that our approach enabled multiagent teams to find policies with up to 21.0% and 18.1% lower makespans than a baseline approach in which agents planned independently.

4.1 Motivation

We would like to model uncertain global traversability information in a roadmap or motion graph, where some edges may be untraversable, and then develop a planner to enable collaborative multiagent team navigation using the graph. For a single agent, the problem of navigation in a roadmap with uncertain edge traversabilities is known as the Canadian Traveller's Problem (CTP), which can be formulated as a Partially Observable Markov Decision Process (POMDP), where the optimal policy trades off between exploiting paths that are traversable with high probability, but with potentially higher cost, against exploration actions that determine the traversability of potentially lower-cost paths (see Section 2.8 for additional details). A collaborative multiagent team considers the same problem, but has additional flexibility to explore the environment with different team members. This flexibility is especially interesting for teams of agents with heterogeneous capabilities (e.g., an air vehicle (AV)/ground vehicle (GV) team), or for tasks where agents are not co-located in the environment (and can provide non-local information to their teammates). For example, consider a team of air and ground vehicles entering an urban environment after a climate disaster, as in Fig. 4.1. The weather may have rendered some of the routes through the city untraversable for some of the agents (e.g., the GVs), and some of the agents may be able to sense the true state of some paths more quickly than others (e.g., the AVs). A collaborative plan that tasks some of the agents with determining which roadways are impassable due to

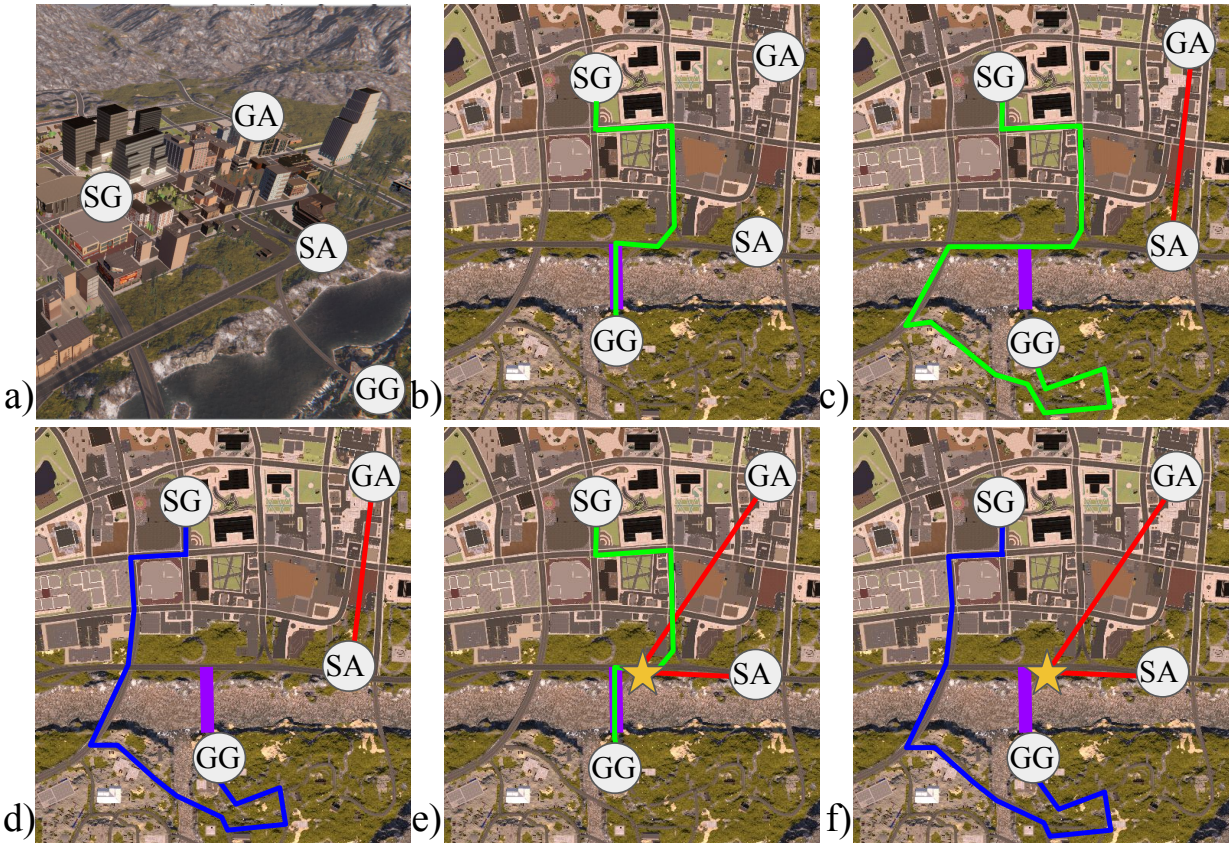


Figure 4.1: **Motivating Example.** Consider an air vehicle/ground vehicle team navigating in a city environment (a) subject to a climate disaster that has impacted the traversability of city infrastructure, like pedestrian bridges (purple). A ground vehicle starting at SG and navigating to GG with no additional information has to select between an unknown trajectory that could lead to either a short path (b, green) or a long detour (c, green) and a medium-length, known trajectory (d, blue), either of which can be suboptimal given the unknown state of the pedestrian bridge. By taking a small diversion to the gold star, the air vehicle (with start SA and goal GA) can sense and share the state of the bridge with the ground vehicle, allowing the ground vehicle to select the best available path to GG for every environmental state (e, green and f, blue) and reducing expected total team navigation cost.

weather will lead to better overall team performance.

While the flexibility of multiagent teams can improve navigation outcomes in unknown environments, generating team plans in this domain requires answering three questions: *what* is useful to explore given the team objective(s), by *when* does the team need the explored information to improve planning performance, and *who* should explore. Answering these questions can require a multiagent team to reason over a POMDP with large action and observation spaces, long time horizons, and delayed rewards, each of which are individually studied as difficult problems in the POMDP literature (e.g., [131, 139, 132, 141, 51, 91, 85, 4]; see Kurniawati [74] for a comprehensive review). We choose to simplify our multiagent planning model by relying on improved terrain scoring algorithms, such as cost-to-go prediction from satellite imagery [39] and traversability prediction from onboard sensors [48], to compactly represent perceptual uncertainty using a sparse graph, enabling us to focus on the challenge of finding good collaborative team plans at long time horizons with delayed rewards.

In this chapter, we model collaborative team policies on stochastic graphs using macro-actions, where each macro-action can consist of multi-step movement actions, sensing actions, and actions to wait for sensor data from other teammates. To reduce the number of macro-actions considered during planning, we generate optimistic approximations of candidate future team states, then generate a small policy class for planning that only includes macro-actions that are likely to lead to high-reward future team states. We optimize team plans over the small policy class. Our approach enables a team to find policies that balance between reducing task-relevant environmental uncertainty and efficiently navigating to goals in toy graph and island road network domains.

4.2 Approach

4.2.1 The Canadian Traveller's Problem

We would like to model a team Λ of l (possibly heterogeneous) robots λ, $\Lambda = \{\lambda_0, ..., \lambda_{l-1}\}$, tasked with goal-directed navigation in a static environment using an imperfect roadmap. The problem of single-agent navigation on uncertain roadmaps is well captured by the probabilistic graph formulation proposed in the Canadian Traveller's Problem (CTP) described in Sect. 2.6. Recall that the CTP is defined as a 5-tuple (G, c, ρ, v^s, v^g):

1. A graph G with vertices V and edges E.
2. Scalar edge costs $c : E \to \mathbb{R}^+$.
3. Edge blocking probabilities $\rho : E \to [0, 1)$.
4. An initial vertex v^s.
5. A goal vertex v^g.

Additionally, the ground truth, binary traversability of each edge in the CTP is encoded in an edge weather $w_e \in \{0, 1\}$, which is constant throughout a planning trial, and it is assumed that an agent can noiselessly sense the traversabilities of edges incident to the agent's current vertex. The goal of the CTP is to find the minimum expected cost policy π for navigating from a start node s to a goal node G in the uncertain graph:

$$\pi^*(v, b) = \arg\max_{\pi \in \Pi} V^\pi(v, b), \tag{4.1}$$

where S is the discrete state space, $s \in S$, and Π is the space of single-agent policies, $\pi : S \times B \to A$,

$$V^\pi(v_a, b_a) = \mathop{\mathbb{E}}_{w \sim b_a} \left[\sum_{t=a}^{\infty} c(v_t^\pi, v_{t+1}^\pi) \right], \tag{4.2}$$

where v_t^π is the vertex reached by the agent at time t when executing policy π from time a until time t.

4.2.2 Multiagent Planning on Unknown Graphs

While the CTP formulation effectively models the global characteristics of an environment with uncertain traversabilities, the formulation was not designed to model navigation for more than one agent. To enable high-quality (e.g., low cost and uncertainty-aware) team planning, we modify the CTP formulation to model collaborative multiagent navigation under uncertainty. Specifically, we modify the objective function defined in Eqs. 4.1 and 4.2 to generate a team policy π that takes each robot in the team from its initial vertex v_i^s to its goal vertex v_i^g, where $v_i^s, v_i^g \in V \ \forall i \in \{0, ..., l-1\}$, and that maximizes a scalar team reward function R (e.g., makespan) in expectation over weathers. Formally, we can model the problem of multiagent planning on a stochastic graph as a specific type of POMDP, the Multi-Agent Deterministic Mixed Observability Markov Decision Process (MADMOMDP), with the following properties:

- Team: The team Λ of l agents, $\{\lambda_0, ..., \lambda_{l-1}\}$.
- Fully observable state S: The set of graph vertices for each agent V^l.
- Partially observable state: The set of weathers W.
- Actions: The set of incident, traversable graph edges for each agent from each vertex, including self-loops.
- Observations: The traversability w_e of an edge, $z_e = w_e \in \{0, 1\}$.
- Observable state transition function: Deterministic transitions along observed, incident, traversable graph edges to adjacent vertices.
- Partially observable state transition function: No transition, as weather w is assumed constant during a trial.

- Observation function: Deterministic observations of the traversability w_e of sensed, incident graph edges.

- Reward R: The negative team makespan, or the negative maximum completion time of any agent in the team, $R = -\max_{\lambda \in \Lambda} \sum_{t=0}^{\infty} c_\lambda(v_{\lambda,t}, v_{\lambda,t+1})$, where, by assuming the reward of an agent remaining in its goal state is zero, we ensure the stationary policy has zero reward and the infinite sum is well defined.

- Initial belief b_0: The prior belief over the weather, $b_0 = \{\text{Bern}(1-\rho_0), ..., \text{Bern}(1-\rho_{m-1})\}$.

We can write the multiagent planning objective as the maximization of the MADMOMDP value function $V^\pi(v, b)$ over multiagent policies $\pi \in \Pi$,

$$\pi^*(v, b) = \arg\max_{\pi \in \Pi} V^\pi(v, b), \tag{4.3}$$

where Π is the space of multiagent policies, $\pi : S \times B \to A$,

$$V^\pi(v_a, b_a) = \mathop{\mathbb{E}}_{w \sim b_a} \left[-\max_{\lambda \in \Lambda} \sum_{t=a}^{\infty} c_\lambda(v^\pi_{\lambda,t}, v^\pi_{\lambda,t+1}) \right], \tag{4.4}$$

and $v^\pi_{\lambda,t}$ is the vertex reached by agent λ at time t when executing policy π from time a until time t.

4.2.3 Macro-actions for Information-gathering Teams

In practice, optimizing Eq. 4.3 over the space of all collaborative team policies Π is intractable due to the size of the team action space and the length of team plans [16]. One approach to increasing planning tractability is using macro-actions $a \in A$, or multi-step actions, which are often selected using domain knowledge, to reduce the effective planning depth. For the single-agent CTP, optimal plans can be generated by optimizing over single-agent macro-action policies $\pi \in \Pi_{sa}$ that consist of only two different types of macro-actions:

1) optimal navigation in the observed subgraph to vertices adjacent to an unknown edge and 2) navigation to the goal [40], where Π_{sa} is the set of policies that can be represented using the two single-agent macro-action types. Intuitively, the single-agent macro-action based polices are optimal because single-agent MADMOMDP policies depend only on the agent's belief b about the unknown weather w. By construction, an agent's belief is constant during the execution of a single-agent macro-action a, so the highest reward agent policy π does not change. Planning using macro-actions reduces policy search depth and breadth, as they reduce the number of timesteps to the goal, and they eliminate policies with suboptimal known-graph traversals from the policy space.

We would like to apply macro-actions to multiagent CTPs to increase the tractability of solving Eq. 4.3. The collaborative nature of multiagent systems means that we need to consider additional macro-actions, specifically 1) sensing an edge and 2) waiting for the edge to be sensed and the result transmitted. However, remote team sensing invalidates the optimality of the single-agent macro-action set, as teammates can update the team belief during one agent's macro-action execution. Additionally, waiting actions can increase the branching factor of macro-action based search, as optimal wait durations are not known before planning, so wait actions with multiple durations must be considered. For example, consider the ground vehicle (GV) in Fig. 4.1; the GV should combine waiting and moving so that it is positioned at the fork in the road where the green and blue paths diverge when the air vehicle (AV) senses the bridge.

Fortunately, while good team policies can be spatially and temporally complex, in many problems of interest, only a small number of collaborative macro-action policies are useful for a specific team planning task. For example, in Fig. 4.1, the only useful collaborative macro-action for the AV is to sense the traversability of the purple bridge on the potentially low cost (green) GV path to the goal; all other sensing actions will not improve team planning performance. Our key realization is that we can use the structure of the MADMOMDP belief space to approximate the value of a hypothetical team belief without explicitly planning

the collaborative actions that lead to the belief. This approximation allows us to focus computation on the (often small) set of collaborative team policies $\hat{\Pi}$ that are likely to lead to high-reward team beliefs, while pruning collaborative team policies that are likely to lead to low-reward team beliefs and are therefore unlikely to improve team planning performance.

4.2.4 Policy Class Augmentation for Planning

In this work, we assume that we are given a small base class of single-agent macro-action policies Π_{sa} that involve moving agents to graph vertices and a much larger class of complex multiagent macro-action policies Π_{ma} that include sensing and waiting actions. We would like to generate a small, high quality policy class $\hat{\Pi}$ for planning by augmenting the base policy class Π_{sa} with a small number of candidate policy classes $\Pi_c \subset \Pi_{ma}$ that are likely to improve team planning performance. While Π_c can in principle be any subset of Π_{ma}, it can be useful when each candidate policy class has a meaningful interpretation in policy space (e.g., Π_c is the set of team policies where an AV senses edge e). In this work, we define candidate policy classes as the set of macro-action policies that include a specific macro-action. The value of a candidate policy class Π_c for planning is the expected increase in planning performance when planning using $\hat{\Pi}$ as compared to planning using the single-agent macro-action policy class Π_{sa}:

$$\Delta(\Pi_{sa}, \Pi_c, v, b) = \max_{\pi \in \Pi_{sa} \cup \Pi_c} V^{\pi}(v, b) - \max_{\pi \in \Pi_{sa}} V^{\pi}(v, b). \tag{4.5}$$

We include a candidate policy class Π_c in $\hat{\Pi}$ if planning using the augmented class results in an improvement in planning performance $\Delta(\Pi_{sa}, \Pi_c, v, b)$:

$$\hat{\Pi}(v, b) = \Pi_{sa} \cup \{\Pi_c | \Pi_c \in \Pi_{ma}; \Delta(\Pi_{sa}, \Pi_c, v, b) > \gamma\}, \tag{4.6}$$

where $\gamma \geq 0$ is a tunable parameter. When γ is low, collaborative actions are added to the policy class if they can lead to any improvement in total team cost; when γ is high, collaborative actions are only added to the policy class if they can lead to a significant

improvement in total team cost. In our experiments, we use a low γ that reduces the impact of floating point errors on policy class selection, but does not otherwise impact action selection.

While the defined policy class $\hat{\Pi}$ is likely to contain a small number of high-reward collaborative team policies and result in efficient, high-reward planning, unfortunately, computing $\Delta(\Pi_{sa}, \Pi_c, v, b)$ for all $\Pi_c \in \Pi_{ma}$ can be expensive, as evaluating each $\Delta(\Pi_{sa}, \Pi_c, v, b)$ may require calculating value functions based on complex multiagent macro-action policies. Next, we 1) discuss the structure of the MADMOMDP belief space, which results from deterministic actions and observations, and 2) discuss team compositions for which we can efficiently generate good approximations of $\Delta(\Pi_{sa}, \Pi_c, v, b)$, and therefore efficiently generate good $\hat{\Pi}$.

4.2.5 Approximations for Policy Selection with One Stochastic Agent

Preliminaries: Belief-based Value Functions

First, we discuss the structure of the MADMOMDP belief space. Consider the beliefs b and b^z, where b^z is the updated team belief after receiving observation z, i.e., $b^z = h(b, z)$ given belief update function h. Note that the observation z is independent of the sensing agent. Because the team operates in a MADMOMDP where the hidden environment weather is constant during planning, the (deterministic) agent observations never increase team environmental uncertainty, i.e., the size of the support of the team belief decreases monotonically over the planning trial [19]. Under the assumption of *rational* planning [124], or that each agent maximizes its reward given its knowledge of the environment, additional environmental information does not decrease planning performance, and the value function for a policy generated using b^z will be at least as good as the value function for a policy generated using b:

$$V^{\pi}(v, b^z) \geq V^{\pi}(v, b). \tag{4.7}$$

We use the assumption of rational planning to generate accurate approximations of the values of collaborative team actions while incurring low computational costs for some teams. Specifically, we develop approximations for teams where one agent acts on the CTP graph with unknown edge traversabilities, like a ground vehicle (GV) navigating on a damaged road network, and one or more agents act on a fully traversable graph of the environment[1], like an air vehicle (AV) flying over damaged terrain. In the remainder of the exposition, we will use the terms stochastic agent/GV and deterministic agent/AV interchangeably.

We focus on the team composition of one GV and multiple AVs because, for a given team plan, the time at which the team belief will change is known – it is the earliest time at which any AV reaches an edge to sense, or when the GV navigates to a vertex with an unknown incident edge. Knowing the team sensing time enables us to use the rational planning assumption to generate efficient approximations of the quality of collaborative team actions during online planning. In future work, we are interested in developing efficient approximations for other team compositions, such as those with multiple agents operating on stochastic graphs.

Value Function Approximations for a 1-GV/n-AV Team

Consider a 1-GV/n-AV team navigating on a CTP graph. We assume that AVs do not observe the traversability of edges while navigating unless specifically tasked to sense, and that the GV immediately observes the traversability of all incident edges of its current vertex.

We would like to generate an augmented policy class $\hat{\Pi}$ that is likely to contain high-quality solutions to the optimization in Eq. 4.1. We begin by identifying the types of macro-action classes $\Pi_c \in \Pi_{ma}$ to consider for planning, assuming a base class Π_{sa} of single-agent, non-collaborative macro-action policies. The first is collaborative sensing macro-action policies Π_{c_s}, in which one or more AV(s) travel to a vertex and sense an incident edge's traversability and transmit the information to the GV. The second is collaborative

[1]The fully traversable graph may include vertices and edges that are never reachable or traversable for the stochastic agent, respectively, and are therefore not included in the CTP graph.

waiting macro-actions Π_{c_w}, in which one or more AV(s) travel to a vertex to sense the environment, and the GV combines movement and waiting macro-actions to optimally take advantage of the sensed information. For example, the GV may select movement and waiting actions to arrive at a fork between two paths just as additional information about the paths is sensed and communicated.

In the following sections, we develop approximations for $\Delta(\Pi_{sa}, \Pi_c, v, b)$ for sensing and waiting macro-actions.

Approximating $\Delta(\Pi_{sa}, \Pi_{c_s}, v, b)$ for Sensing Actions

First, we discuss approximations for sensing policies Π_{c_s}. Each sensing action can be thought of as two separate single-agent actions: an AV sensing the desired edge e with a corresponding cost of traveling to sense the edge, and the GV planning using the sensed information with a potential reduction in overall plan cost resulting from knowing edge traversabilities, both of which can impact team makespan. Because all graph edges are traversable for the AV, the cost c_s of detouring from the optimal AV path (with cost $c(v, G)$) to sense edge e before continuing to the goal can be computed exactly using a shortest paths algorithm,

$$c_s = \underset{v' \in \{v_{e_0}, v_{e_1}\}}{\arg\min} \; c(v, v') + c(v', G), \tag{4.8}$$

where we assume an edge can be sensed at either of its endpoints v_{e_0} and v_{e_1}, and that if either v_{e_0} or v_{e_1} is on the AV's shortest path to the goal, the expected loss due to sensing is zero. We also recover the deterministic edge sensing time,

$$t_s = c(v, v'), \tag{4.9}$$

where v' is the minimizing vertex in Eq. 4.8.

We can now consider how the sensing action reduces the GV cost. To directly calculate the benefit of the sensing action for the GV, we would need to find an optimal joint GV/AV

policy, which may contain a combination of movement, sensing, and waiting actions, and may be computationally intractable to generate. However, we can use Eq. 4.7 to generate bounds for the joint policy that are inexpensive to compute, based on the value functions of an agent that does not receive sensed information and an agent that does receive the sensed information. First, we consider an agent at vertex v with traversability belief b that does not receive additional sensing information and acts using a single-agent policy π_{sa}, with value function $V^{\pi_{sa}}(v, b)$. This value function can be well-approximated using a single-agent CTP policy.

Second, we consider an agent at vertex v with traversability belief b that receives an observation of edge e at time t_s and acts using policy π_{ma}, a multiagent sensing policy, with value function $V^{\pi_{ma}}(v, b, e, t_s)$. Calculating this value function exactly requires reasoning over multiagent sensing policies. To avoid solving this complex planning problem, we instead find an upper bound for the expression by observing that Eq. 4.7 implies that

$$V^{\pi_{ma}}(v, b, e, t_s) \leq V^{\pi_{ma}}(v, b, e, 0), \tag{4.10}$$

or that the value of plan generated by immediately sensing the unknown edge will be at least as good as the value of a plan where the edge is sensed later during planning. Next, we realize that the value of a multiagent policy with immediate sensing is equivalent to the value of a single-agent policy calculated using the updated belief after sensing, b^z,

$$V^{\pi_{ma}}(v, b, e, 0) = \sum\nolimits_{z \in Z} p(z) V^{\pi_{sa}}(v, b^z | z = z), \tag{4.11}$$

where b^z is the agent's traversability belief after receiving an observation of edge e. Combining Eqs. 4.10 and 4.11, we recover the following upper bound,

$$V^{\pi_{ma}}(v, b, e, t_s) \leq \sum\nolimits_{z \in Z} p(z) V^{\pi_{sa}}(v, b^z | z = z), \tag{4.12}$$

which can be well-approximated using a rollout-based single-agent CTP policy. Using Eqs. 4.8 and 4.12, we recover an upper bound for Δ, the expected change in team makespan caused by using sensing actions, that is efficient to compute,

$$\Delta(\Pi_{sa}, \Pi_{c_s}, v, b) \leq \max\Big\{\sum\nolimits_{z \in Z} p(z) V^{\pi_{sa}}(v, b^z | z = z), c_s\Big\} - \max\Big\{V^{\pi_{sa}}(v, b), c(v, G)\Big\}. \quad (4.13)$$

Approximating $\Delta(\Pi_{sa}, \Pi_{c_w}, v, b)$ for Waiting Actions

Next, we consider waiting policies Π_{c_w}. Each waiting action can be thought of as two separate single-agent actions taken by the GV: 1) waiting to begin moving, and 2) following a policy to the goal. First, we consider the wait-to-move action. Because determining the optimal waiting time requires solving the multiagent problem, we can compute a lower bound of the cost of the wait-to-move action by assuming that the agent does not wait to move, and the wait cost is zero. Second, we approximate the value of planning to the goal after waiting. We can again use Eq. 4.7 to develop efficiently computable bounds for the joint policy. To generate the bound, we consider the value functions of two agents: one that waits, and one that does not wait. First, we consider the value function of an agent that receives a new edge observation at t_s and that cannot select wait-to-move actions at any time during planning, $V^{\pi_{ma}}(v, b, e, t_s)$. The value of this policy is equivalent to the value of a two-step single-agent policy that first navigates until the sensing horizon with no information about e or the AV's sensing action, and then replans using the updated belief and navigates from the sensing horizon to the goal:

$$V^{\pi_{ma}}(v, b, e, t_s) = V^{\pi_{sa}}_{t_s}(v, b) + V^{\pi_{sa}}(v_{t_s}, b^z_{t_s}), \quad (4.14)$$

where V_{t_s} indicates the value function for navigating until time t_s with belief b, and v_{t_s} and $b^z_{t_s}$ are the vertex and belief of the agent at t_s, assuming π_{sa} was followed until t_s and observation z was received. Second, we consider the value function of an agent that receives

a new edge observation from the AV executing the sensing action at macro-action sensing time t_s, and that can select wait-to-move actions until some waiting time t_w before or during sensing, $V^{\pi_{ma}}(v, b, e, t_s, t_w)$, where $t_w \leq t_s$. Determining this value function can require considering waiting team actions; we again avoid this computation by generating an upper bound. First, we note that Eq. 4.7 implies that waiting for additional sensing information can only improve the value of future policies (assuming no wait cost):

$$V^{\pi_{ma}}(v, b, e, t_s, t_w) \leq V^{\pi_{ma}}(v, b, e, t_s, t_s). \tag{4.15}$$

Next, we realize that the value of a policy that waits until the sensing time to take an action is equivalent to the value of a policy that senses the edge immediately and does not wait, again assuming no wait cost.

$$V^{\pi_{ma}}(v, b, e, t_s, t_s) = V^{\pi_{ma}}(v, b, e, 0). \tag{4.16}$$

Using Eqs. 4.11, 4.15, and 4.16, we recover a bound for $V^{\pi_{ma}}(v, b, e, t_s, t_w)$,

$$V^{\pi_{ma}}(v, b, e, t_s, t_w) \leq \sum\nolimits_{z \in Z} p(z) V^{\pi_{sa}}(v, b^z | z = z), \tag{4.17}$$

which can be well-approximated using a rollout-based single-agent CTP policy. We recover the tractable Δ bound:

$$\begin{aligned}\Delta(\Pi_{sa}, \Pi_{c_w}, v, b) \leq \max\Big\{\sum\nolimits_{z \in Z} p(z) V^{\pi_{sa}}(v, b^z | z = z), c_s\Big\} \\ - \max\Big\{V_{t_s}^{\pi_{sa}}(v, b) + V^{\pi_{sa}}(v_{t_s}, b_{t_s}^z), c_s\Big\}.\end{aligned} \tag{4.18}$$

4.2.6 Using Problem Structure to Reduce the Number of Candidate Policy Classes

Unfortunately, even calculating approximations of policy quality Δ for all candidate policy classes Π_c to generate $\hat{\Pi}$ can be computationally expensive. However, in many structured environments, there exists a macro-action ordering property that can be used to select an order in which to try to add Π_c to $\hat{\Pi}$, and to prune some candidate policies Π_c without approximation. For example, a policy class with a waiting action can only improve planning performance if it also includes a sensing action that was useful for planning; otherwise, the agent will consider waiting for information that will never come or is not useful. For the single AV case, we first calculate $\Delta(\Pi_{sa}, \Pi_{c_s}, v, b)$ for a sensing policy class Π_{c_s}. If the sensing action does not improve performance, the action is not included in $\hat{\Pi}$, and we do not approximate the value of the corresponding waiting candidate policy class Π_{c_w}. Otherwise, we calculate $\Delta(\Pi_{sa}, \Pi_{c_w}, v, b)$ for waiting macro-action policy class Π_{c_w} to determine if it is beneficial to sense and wait. If waiting improves performance, the waiting macro-action is added to $\hat{\Pi}$; otherwise, only the sensing macro-action is added to $\hat{\Pi}$. For the n-AV case, we calculate $\Delta(\Pi_{sa}, \Pi_{c_s}, v, b)$ for every edge, then greedily assign an edge to each agent (note that the improvement due to sensing is sensing-agent independent, while the cost of the sensing action is sensing-agent dependent). Given the assignment, we calculate the value of sensing and the value of waiting for the multi-agent action, as in the single-agent case, and generate $\hat{\Pi}$. While we consider a single greedy edge assignment per timestep, more complex assignment and approximation strategies could be used.

4.2.7 Planning using Δ and Multiagent Macro-Actions on CTPs

Finally, we discuss a planner that uses a macro-action policy class $\hat{\Pi}$ to optimize Eq. 4.1 in an online, centralized multiagent CTP solver with perfect communication. As input, we take a MADMOMDP, a baseline policy class to use in all planning instances, like single-agent

macro-action policies Π_{sa}, and a set of multiagent macro-action policies to consider, Π_{ma}. In some cases, we are also given an ordering property. We assume that edges incident to GVs at the start of an instance are observed.

Planning begins by approximating $\Delta(\Pi_{sa}, \Pi_c, v_{GV}, b_{GV})$ for every Π_c for the GV (unless previous computations combined with policy orderings guarantee that $\Delta(\Pi_{sa}, \Pi_c, v_{GV}, b_{GV}) < \gamma$) and generating a $\hat{\Pi}$ according to Eq. 4.6. Then, we optimize Eq. 4.1 over $\hat{\Pi}$ to generate a team macro-action policy π, which may include collaborative macro-actions. Team policies are executed as follows: first, each agent begins to execute its current macro-action. When any subset of agents finishes their current macro-actions, they update their belief and communicate their belief with each other and the central planner. The central planner replans, and the finished agents begin their next macro-actions. Additionally, non-finishing agents truncate their macro-actions as soon as possible (i.e., at the end of their current primitive action, or graph edge), communicate with the central planner to get the updated team belief, and replan. This process repeats until all agents reach their respective goals.

4.3 Experiments

We benchmarked our informed, policy-class aware multiagent planner, *Comms+Approx*, against three multiagent planners, *Independent*, *Passive Comms*, and *Comms+No Approx*. In *Comms+Approx* planning, the multiagent team planned using the described approach, and the GV sensed all incident edges during plan execution, while the AV(s) sensed only requested edges. In *Independent* planning, each agent planned independently, and did not share information with the team; for this case, we assumed that the GV sensed all incident edges during plan execution, and the AV(s) did not sense. In *Passive Comms* planning, each agent planned independently, but the agents shared, updated, and planned using a joint belief space; for this case, we assumed that the GV sensed all incident edges during plan execution, and the AV(s) sensed all traversed edges during plan execution. In *Comms+No Approx*, the

team planned using a modified version of *Comms+Approx* that did not generate approximations and planned over the full multiagent macro-action policy class, $\hat{\Pi} = \Pi_{sa} \cup \Pi_{ma}$. We also compared our approach to an oracle, *Oracle*, which planned using Dijkstra's algorithm [34] on a fully observed graph.

We used Monte Carlo simulation to generate value function approximations for planning. As in Eyerich et al. [40], we use the same names to refer to algorithms and policies for planning, where executing an algorithm corresponds to optimizing the corresponding policy at each timestep, and a policy is generated online by selecting the macro-action that maximizes the value function of the macro-action *successor*, or the terminal state of the macro-action. Independent macro-actions were evaluated using the optimistic rollout policy (ORO) [40], which calculated successor value functions by simulating optimistic planning (assuming unknown edges were traversable) from each successor for r different weathers, where r is the number of rollouts used. We also modified ORO to approximate the sensing value functions by revealing information sensed by other agents during optimistic planning, and by triggering replanning when the team belief was updated. Waiting value functions were evaluated using a combined policy, Greedy Rollout - Greedy policy (GR-G), where value functions were calculated for all navigation-based successors and the current state, which is the result of taking the single timestep wait action. The GR-G policy approximated successor costs by averaging the costs of Greedy policy (G) executions for r rollout weathers; the Greedy policy (G) calculated macro-action values by simulating execution of the optimistic policy in a given rollout weather for each successor and the current state. We planned using $r = 100$ rollouts for all policies, as this provided a good trade-off between computation time and solution quality in preliminary experiments, and $\gamma = 1 \times 10^{-10}$. When evaluating value functions to calculate Δ, we used the same rollout weathers to evaluate each term in the approximation, as this reduced errors in Δ caused by rollout weather stochasticity. Note that because the value functions used to generate the bounds in Sect. 4.2.5 are approximated, we cannot guarantee that our implementation does not prune macro-actions that could lead

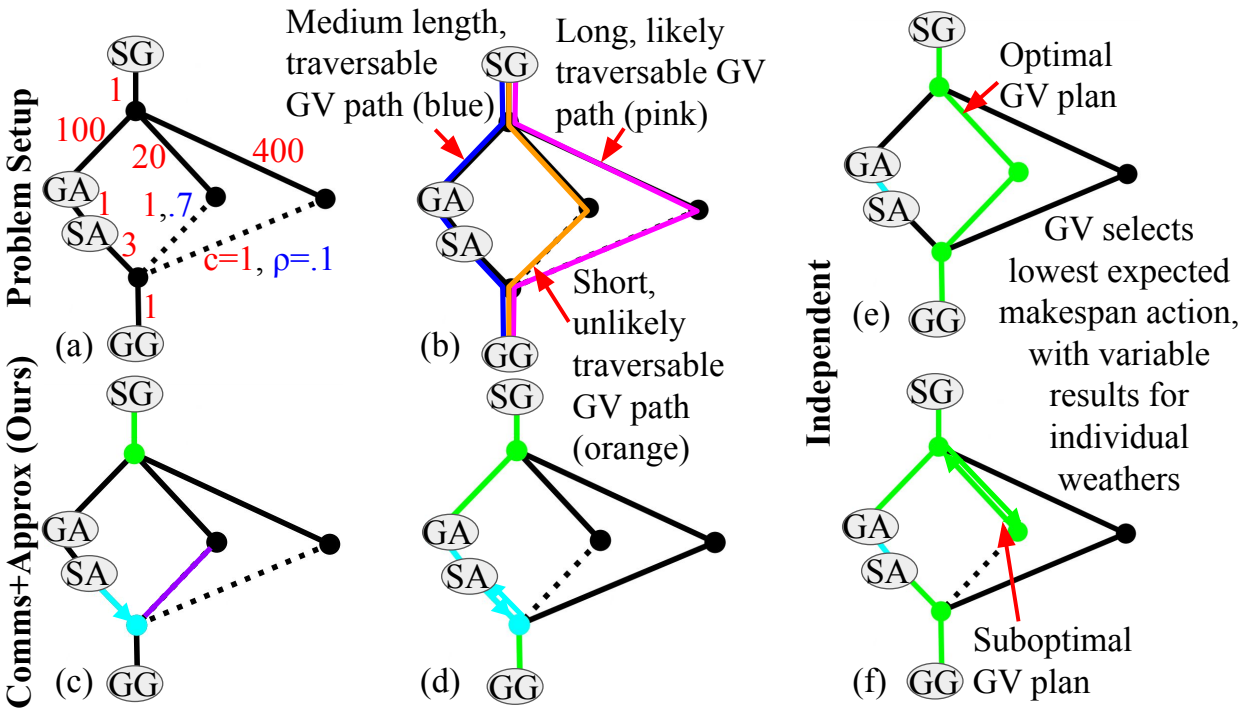


Figure 4.2: **Visual Plan Comparison in the Toy Graph.** Plans executed by the AV (blue) and GV (green) navigating from their respective starts (SA/SG) to goals (GA/GG) in the toy graph (a, b), where edge costs and blocking probabilities are labeled in red and blue, respectively. Solid lines represent traversable edges and dotted lines represent edges with uncertain traversability. In the *Independent* plan, the AV navigated directly to the goal, and the GV selected the plan that minimized the expected makespan over all weathers, resulting in highly variable planning performance in some weathers (e, f). In the *Comms+Approx* plan, the AV diverted to sense the unknown edge (purple) on the short path to the goal, while the GV combined waiting and moving and arrived at the fork between paths when the AV sensed the edge on the short path (c). Then, the GV took the shortest traversable path to the goal (d).

to high-reward plans due to poor value function approximation. While the OPT, ORO, and GR-G rollout policies were selected due to their good performance for low r, a rollout policy with a regret bound, such as UCT [70], could be used to bound the regret of pruning macro-actions using the proposed value function bounds.

4.3.1 Toy Environment

We demonstrated our approach by planning for a 1-GV/1-AV team for 100 trials in the toy scenario in Fig. 4.2-a-b . In all trials, the ground vehicle navigated from start SG to goal GG,

Metric ↓ better	**Oracle**	**Independent**	**Passive Comms**	**Comms + Approx**	**Comms +No Approx**
Avg % Regret	0	81.82	81.07	**4.00**	**4.00**
Avg Makespan	80.27	104.10	103.30	**82.27**	**82.27**
SEM Cost	3.86	3.08	3.02	3.86	3.86
Avg GV Cost	80.27	104.10	103.30	**82.27**	**82.27**
SEM GV Cost	3.86	3.08	3.02	3.86	3.86
Avg AV Cost	1.0	**1.0**	**1.0**	7.0	7.0
SEM AV Cost	0.0	0.0	0.0	0.0	0.0
Avg Time (s)	-	**1.18**	**1.18**	24.29	42.62
SEM Time (s)	-	0.01	0.01	0.06	0.15

Table 4.1: **Toy graph results.** We report the average (Avg) normalized percent regret from optimal, average team makespans, AV costs, and GV costs in simulation units (u), algorithm execution times, and standard errors of the mean (SEM). Our approach, *Comms+Approx*, generated lower makespan plans than the single-agent planning approaches, *Independent* and *Passive Comms*, and was more time efficient than *Comms+No Approx*, which did not use approximations to generate a small policy class $\hat{\Pi}$.

and the air vehicle navigated from start SA to goal GA. A weather was drawn randomly from the distribution over weathers $p(w)$ at the start of each trial, and the GV and AV navigated at the same speed. The GV had three possible paths to the goal: one long, likely traversable path, one medium-length, traversable path, and one short-length, likely untraversable path. The non-collaborative planners selected a GV plan without knowledge of the state of the unknown edge on the short path to the goal, which resulted in highly variable GV planning performance given the weather. The performance in (e) matched what the planner expected, but in (f) the unknown edge on the short path was in fact not traversable and the GV had to backtrack in a way the planner did not anticipate, leading to higher overall cost. In contrast, the collaborative planners predicted the potential cost of the unknown edge and used the AV to sense it, while the GV combined waiting and movement macro-actions to arrive at the decision point between paths when the AV sensed and shared the traversability of the edge (c). This enabled the GV to select the shortest path to the goal for every weather (d).

While the sensing and waiting actions incurred small constant GV and AV costs in all trials, they resulted in more consistent, ultimately lower makespan team plans across weathers. Additionally, the AV did not sense the uncertain edge on the long GV path to the goal, indicating that our approach distinguished between task-relevant and task-irrelevant team sensing macro-actions.

Quantitative results for the trials are shown in Table 4.1. We demonstrated that our approach found plans with 21.0% and 20.4% lower average makespans as compared to the single-agent baselines, *Independent* and *Passive Comms*, respectively, indicating that collaborative planning improved navigation outcomes in the environment. We also showed that our approach found similar makespan plans to *Comms+No Approx* while planning for 43.0% less time on average, demonstrating that the proposed approximations increased planning efficiency without reducing plan quality.

4.3.2 Islands Environment

Next, we demonstrated our approach for a 1-GV/2-AV team navigating in a simulated island environment, modeled as a CTP graph with 90 vertices and 166 edges with Euclidean distance edge costs, shown in Fig. 4.3. We generated 100 environments by sampling edge blocking probabilities uniformly from an allowable range based on edge type. We modeled four different types of edges: highway and on-ramp edges with $\rho \in [0.0, 0.001)$, and local road and bridge edges with $\rho = 0.5$. We generated 10 trials per environment by randomly sampling a start and goal in every graph, then sampling 10 individual trial weathers per graph and start/goal pair, discarding and re-sampling if the start and goal were not connected in the weather. We also discarded and resampled start/goal pairs if the GV trajectory generated by the first weather was not at least 8 edges long to remove short, high traversal probability trajectories from our dataset that were unlikely to benefit from collaborative planning. We also assumed that the AVs moved 8x faster than the GVs.

We quantitatively assessed the *Independent*, *Passive Comms*, and *Comms+Approx* plan-

Metric ↓ better	**Oracle**	**Independent**	**Passive Comms**	**Comms+ Approx**
Avg % Regret	0	35.02	29.33	**12.81**
Avg Makespan	1151.34	1595.42	1519.35	**1306.94**
SEM Makespan	12.39	26.62	23.55	17.04
Avg GV Cost	1151.34	1595.42	1519.35	**1306.29**
SEM GV Cost	12.39	26.62	23.55	17.06
Avg AV Cost	181.89	**181.89**	**181.89**	1011.05
SEM AV Cost	1.04	1.04	1.04	15.7
Avg Time (s)	-	31.13	**27.26**	5203.42
SEM Time (s)	-	0.75	0.55	199.09

Table 4.2: **Islands domain results.** We report the average (Avg) normalized percent regret, average team makespans, AV costs, and GV costs in simulation units (u), algorithm execution times (s), and standard errors of the mean (SEM). Our approach, *Comms+Approx*, resulted in lower makespan plans than *Independent* and *Passive Comms*, often by diverting the AV to sense, resulting in larger AV costs but smaller GV costs, reducing team makespan.

ners in Table 4.2; results for the *Comms+No Approx* approach were omitted due to computational limitations. We also compared to an oracle planner that ran a shortest path algorithm on the fully observed graph for each agent. Our approach found 18.1% and 14.0% lower makespan plans than the *Independent* and *Passive Comms* planners on average. Our approach frequently resulted in shorter GV plans and longer AV plans, indicating that it used AV sensing actions to reduce GV planning uncertainty, which resulted in shorter GV plans and shorter team makespans overall. However, as our approach solved a multiagent POMDP, rather than three independent single-agent POMDPs, it was less time efficient: 167x and 191x slower than *Independent* and *Passive Comms*, respectively.

Example plans generated by the *Independent* and *Comms+Approx* approaches are shown in Fig. 4.3. In *Independent* planning, the GV attempted to reach the goal using an untraversable candidate plan and had to backtrack, while the AVs navigated directly to the goals. In *Comms+Approx* planning, the GV waited while the AVs sensed various candidate paths in the environment, and then took a low cost path to the goal, resulting in a 26.8%

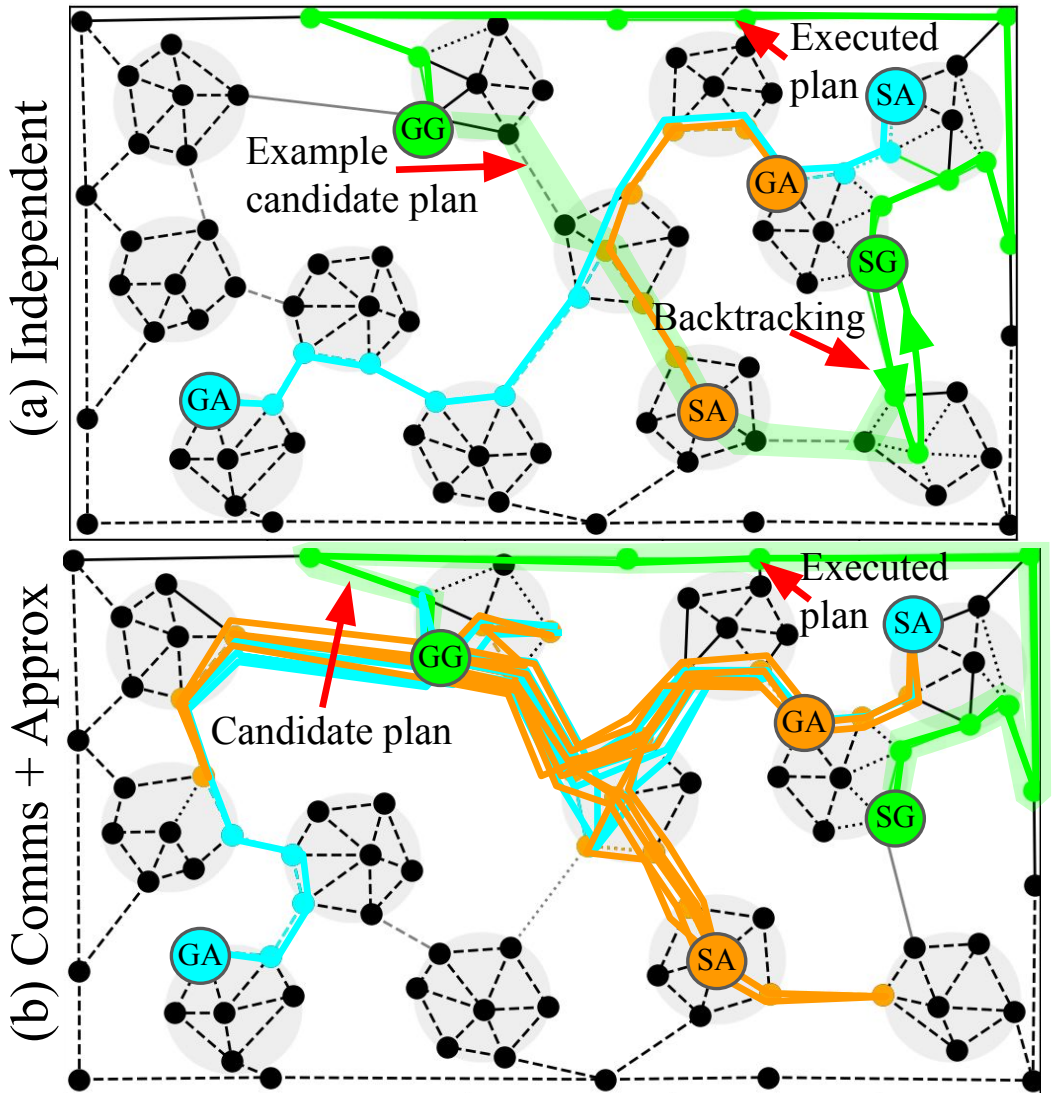


Figure 4.3: **Visual Plan Comparison in the Islands Domain.** Example plans executed by the GV (green) and AVs (blue/orange) navigating from their respective starts (SG/SA) to goals (GG/GA) in the islands domain. Dotted, solid, and dashed lines represent untraversable, traversable, and unknown edges at trial termination. (a) The *Independent* planner selected an untraversable GV path (highlighted green), and the GV backtracked after reaching the dead end. (b) In the *Comms+Approx* plan, the AVs navigated to and sensed uncertain edges, and determined the candidate path in (a) was untraversable. The GV waited for information about the candidate path, then navigated to the goal. Note that the AVs did considerable additional sensing work, traversing a large part of the graph to sense edge traversability for the GV, which drove down the team makespan (e.g., by 26.8% here).

lower team makespan.

4.4 Conclusion

In this chapter, we presented a novel method for improving collaborative team navigation in stochastic, unknown environments. Our approach used a sparse, uncertainty aware graph as a global environment model for planning. Then, we developed uncertainty-aware multiagent macro-actions for the sparse graph, and devised value function approximations, based on the macro-actions, to quickly identify a small set of high-quality collaborative multiagent policies. By explicitly modeling global environmental uncertainty and developing a multiagent planner capable of reasoning about the uncertainty, our approach enabled multiagent teams to collaboratively reduce global task-relevant environmental uncertainty during multiagent navigation. We demonstrated that our planner reduced team navigation costs as compared to agents that planned independently in the uncertain environment, and increased team planning efficiency as compared to teams that optimized the collaborative multiagent planning under uncertainty objective naively over all possible collaborative multiagent policies.

Chapter 5

Real-World Deployment of a Hierarchical Uncertainty-Aware Collaborative Multiagent Planning System

The research described in this chapter was performed jointly with Samuel Prentice and Yasmin Veys.

In Chapter 4, we developed a method for collaborative multiagent planning on abstract roadmaps, or motion graphs, where some graph edges were assumed to be probabilistically traversable [136]. We demonstrated that the approach enabled a collaborative multiagent team to find low-cost team policies that actively balanced between reducing task-relevant environmental uncertainty and efficiently navigating to goals. However, the approach used known, abstract models of the problem to generate high-level solutions for complex teams that were not directly executable in real-world environments.

In this chapter, we develop a hierarchical planning system to deploy our collaborative multiagent planner on a real-world team operating in two different large, outdoor environments. Our system was designed to handle both the theoretical and practical challenges of field robotics deployments, including abstract model generation and alignment, robust

solution execution under uncertainty, and team coordination. Additionally, by developing a planning system that was robust to model mismatch and robot failures at every level of the planning hierarchy, we enabled the team to complete collaborative navigation tasks, even in the presence of imperfect planning abstractions and real-world uncertainty. We demonstrated our approach on a Clearpath Husky-Jackal team navigating in two large, outdoor environments over the course of 18 collaborative and baseline planning trials, and showed that our system resulted in real-world collaborative team planning.

5.1 Motivation

We would like to enable a team of robots to carry out collaborative missions autonomously in unknown environments. Unfortunately, abstract models that are used to generate collaborative team plans are often very different from low-level models that enable agents to execute plans in real-world environments. When selecting an environment model for planning, there is a trade-off to make between model and planner complexity and real-world plan executability; as our models become more abstract or use additional assumptions to enable more efficient high-level planning, model properties that are needed to represent the details of plan execution can become abstracted away. This is especially true when solving complex robotics tasks, as models and planners that are sparse, representative, and efficient for such tasks often abstract away details that do not impact high-level decision making, but these details are often critical for real-world deployments. For example, consider a ground vehicle team collaborating in the semi-structured outdoor environment shown in Fig. 5.1-a. While the presence of an overgrown bush on a path is unlikely to impact high-level decisions about team collaboration, an agent that improperly models the bush could become stuck during low-level navigation, or could mistakenly report that the topology of the environment has changed relative to the abstract graph, when in reality the geometry of the bush is unlikely to cause more than a minor detour. Additionally, other characteristics of real-world de-

Figure 5.1: **Real-world deployment of a collaborative multiagent planner.** a) In this chapter, we discuss the deployment of a collaborative multiagent planner for a two agent team consisting of a Clearpath Jackal and a Clearpath Husky in a real-world semi-structured outdoor environment. b) Consider the Jackal and Husky navigating from starts SJ and SH to goals GJ and GH, respectively, given a navigation graph (green), where one edge has unknown traversability (red). c) When the agents plan independently, the Husky attempts to navigate to the goal via the unknown edge, senses that the edge is untraversable, and then backtracks to the goal via a long, traversable path (trajectory shown in yellow-orange); meanwhile, the Jackal navigates directly to its goal (trajectory shown in blue-green). d) When the agents plan to collaboratively minimize team makespan, the quicker Jackal diverts from the shortest path to its goal to sense the traversability of the unknown edge for the Husky, while the Husky waits in place for additional information. After sensing that the edge is untraversable, both the Jackal and the Husky navigate to their respective goals.

ployments, such as timing delays and robot failures, further stress the relationship between planning abstractions and real-world robot actions.

One method for overcoming the challenges of model mismatch for real-world planning is hierarchical planning. In hierarchical planning, high-level environment models are used to efficiently generate solutions to complex planning problems. Then, in lower levels of the planning hierarchy, the abstract solutions can be refined to reincorporate environment details that are necessary for plan execution. Hierarchical planning approaches can reduce total team computation by enabling agents to use different models for planning when generating different types of plans, and by focusing planning computation on task-relevant information, such as teammate locations during collaborative team planning and detailed occupancy maps during single-agent trajectory execution. Finally, robust planning models at each level in the planning hierarchy often enable agents to recover from unexpected outcomes or incompatible models at lower levels in the planning hierarchy, which can reduce instances of global replanning.

In this chapter, we develop a hierarchical planning system capable of overcoming model mismatch to execute collaborative multiagent planning in real-world environments. Our three-level system generates abstract, collaborative team plans using a sparse, global, uncertainty-aware graph-based representation of the environment, processes single-agent macro-actions using a robust macro-action planner, and executes commands on the robots using primitive action planners, all while maintaining plan consistency throughout the hierarchy and being robust to real-world uncertainty and disturbances. We deployed our approach on a Clearpath Husky-Jackal team navigating in the large, structured outdoor environment shown in Fig. 5.1 and in the large, semi-structured outdoor environment shown in Fig. 5.8, and showed that our system resulted in real-world team collaboration that reduced team total times and execution times by up to 3.5% and 24.6%, respectively.

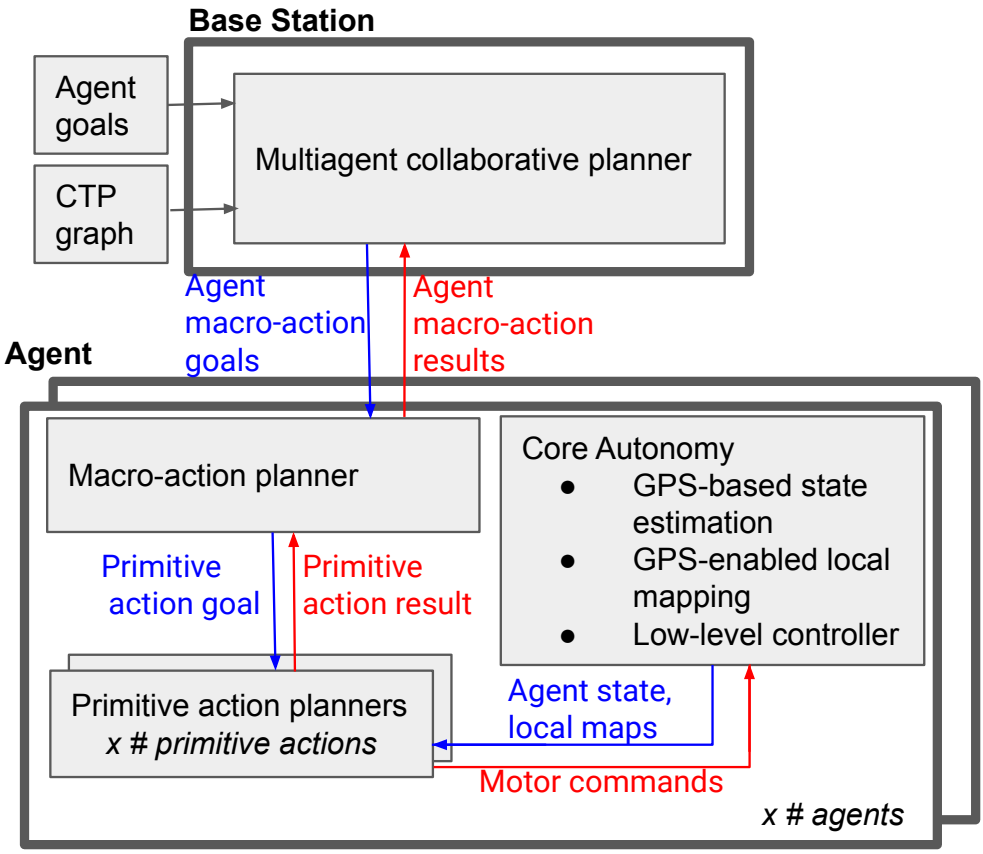


Figure 5.2: **Overview of the hierarchical planning system.** Multiagent planning, which occurred at a centralized base station, generated macro-actions for each agent in the team. The macro-actions were then sent to the individual agents, where they were processed and executed as sequences of primitive actions, using information from each agent's individual core autonomy pipelines.

5.2 Approach

In this section, we discuss the hierarchical planner developed for online, collaborative team planning in the real-world, and highlight challenges and opportunities from deploying the system.

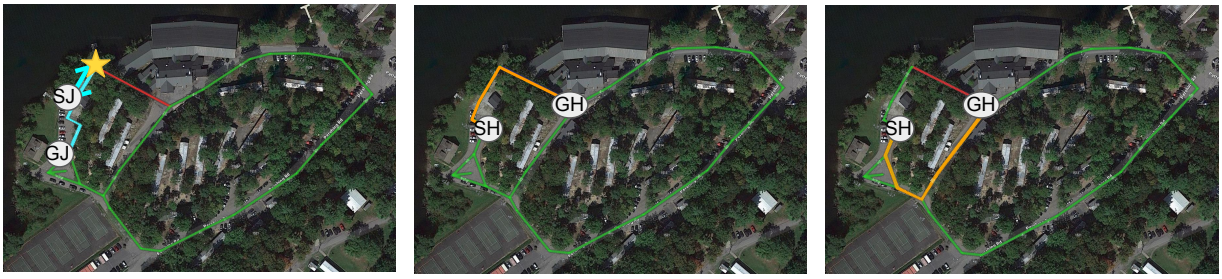
5.2.1 Real-World Planning System Overview

An overview of our hierarchical planning system is shown in Fig. 5.2. Our planner had three levels: a collaborative, multiagent CTP graph-based planning module, a macro-action

execution module, and a primitive action execution module. At the most abstract level of the hierarchy, the collaborative multiagent graph-based planner described in Chapter 4 was used to generate collaborative macro-action-based team plans, given a stochastic navigation graph and agent goals as input; this planner was run at a centralized base station. Then, the macro-action plans were sent to the individual agents for lower-level planning and action execution. At the second level of the planner, macro-action planners running on each robot parsed single-agent macro-action-based plans into sequences of primitive actions, and managed primitive action failures and early termination due to teammates sensing and sharing environment information. Finally, at the lowest level of planning hierarchy, primitive action planners generated executable primitive plans for the agents, including navigation along a graph edge, sensing the traversability of an edge, and waiting in place for information from a teammate. Finally, each agent also used a core autonomy pipeline, which included GPS-based state estimation, GPS-enabled local mapping, and a low-level controller capable of executing motor commands, for primitive action execution. In the following sections, we discuss each of the system components and associated challenges in more detail.

5.2.2 Level 1: Collaborative Multiagent Planning on Abstract Graphs

The first level of the hierarchical planning system was collaborative team planning, which used an abstract motion graph of the environment and pre-specified agent starts and goals to generate collaborative team plans; an example collaborative plan is shown in Fig. 5.3. However, the high-level planner assumed access to an uncertainty-aware route graph prior to planning, and also assumed perfect agent state estimation in the graph-based model for the duration of the planning trial. Unfortunately, in real-world deployments, it can be difficult to build sparse, representative planning models, such as route graphs, prior to planning, and it can be challenging to maintain consistency in different representations in a hierarchy, such as aligning abstract navigation graphs, agents, and environmental features in a common reference frame during trial execution.



a) The planer diverts the Jackal to sense the unknown edge.

b) When the edge is traversable, the planner selects the short Husky path to the goal.

c) When the edge is untraversable, the planner selects the long Husky path to the goal.

Figure 5.3: **An example collaborative team plan.** The Jackal and Husky were tasked with navigating from SJ and SH to GJ and GH, respectively, by executing actions on a predefined abstract motion graph (green) where some edges had unknown traversabilities (red). In collaborative plans, agents took actions that were individually suboptimal to reduce the expected team makespan. Here, the Jackal diverted from its optimal path to the goal to sense and share the traversability of the unknown edge (a), while the Husky waited for the edge traversability information. Then, the Husky used the information to select the best traversable path to its goal, which reduced its expected plan cost (b-c).

In this work, we used two methods to generate sparse, representative abstract route graphs for deployments. In the first deployment, we hand-generated an initial graph of the environment based on overhead imagery, where each node was associated with a GPS coordinate. Additionally, we added deterministic edges between nodes where we expected that a valid, real-world traversal existed (e.g., when nodes were connected via roads), and we also added probabilistically traversable edges (with probability of being untraversable ρ) where we expected that a valid, real-world traversal may exist (e.g., when nodes were connected via a forested area). In the second deployment, we auto-generated motion graphs from hand-generated polygonal environment models, which were derived from overhead imagery, using the technique developed by Veys et al. [149]. In both deployments, we fine-tuned the motion graphs in the field to account for offsets and errors resulting from stale, low-fidelity overhead imagery.

Additionally, throughout the deployments, we used GPS to generate alignments between the route graph, the environment, and the multiagent team, and to establish a global reference frame. While using GPS was effective in our GPS-enabled environment, we are

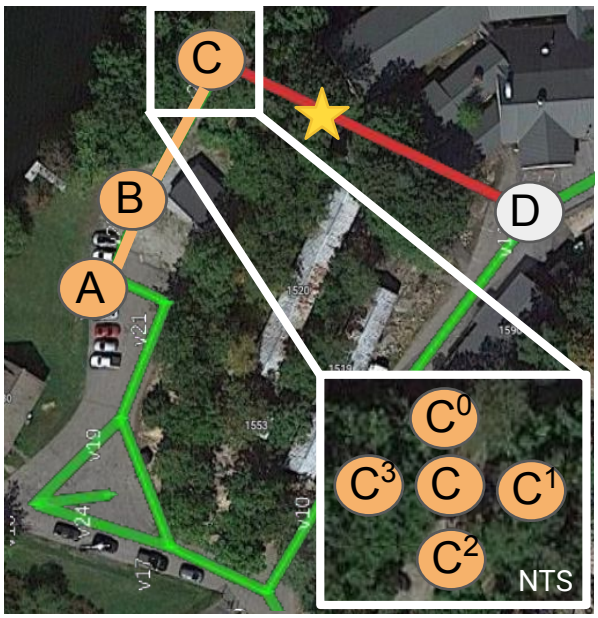


Figure 5.4: **An example macro-action.** An example macro-action (orange) that represents an agent first navigating from node A to B to C, and then sensing the traversability of the unknown edge C-D (gold star). The inlay depicts goal adjustment, which enables an agent to navigate to a location close to the original GPS location associated with the target node, if navigation to the target GPS location is deemed infeasible. This adjustment enables the agent to be robust to various types of uncertainty during planning, including minor pose drift, minor map noise, and small changes in the environment structure.

interested in exploring GPS-denied planning in the future. For example, it may be possible to generate a common team reference frame based on co-observed features at startup, or during planning using a distributed multiagent SLAM pipeline (e.g., using the approach proposed by Tian et al. [144]).

5.2.3 Level 2: Executing Macro-actions for Collaborative Planning

The second level of the hierarchical planner was the macro-action planner, which translated agent macro-actions into subgoals that could be executed in the real world. The planner processed macro-actions into sequences of executable primitive actions, monitored primitive action results, and combined the results of primitive action sequences into macro-action results (i.e., success or failure) that could be processed by the high-level collaborative planner. An example macro-action is shown in Fig. 5.4; the agent navigates from its current location,

node A, to nodes B and C, and then observes the traversability of the edge, C-D.

Unfortunately, naive macro-action planners can result in poor planning performance when macro-action assumptions do not hold in the real world. For example, in the simulated planner, macro-actions were guaranteed to succeed, as they modeled deterministic actions in the known graph. However, in the presence of real-world environmental uncertainty, it is not possible to guarantee that macro-actions will succeed as intended. Additionally, the asynchronous graph-based planner relied on deterministic macro-action completion to determine when agents should stop to share and receive information; however, it is not realistic to pre-plan such interactions in real-world, uncertain environments, as macro-action execution times can vary due to unmodeled environmental uncertainty. In this section, we develop a robust macro-action execution module and a semi-synchronous macro-action planner to enable the robust execution of macro-actions for real-world teams.

Robust Macro-action Execution

First, we address the issue of primitive action failures. For example, agents commonly failed to navigate along edges to reach nodes during primitive navigation actions. However, identifying the cause of an agent failure is important to responding to the failure properly. For example, if a primitive action failure is caused by a significant discrepancy between the graph-based model and the true environment topology, the team should record the failure, update the environment model as necessary, and replan. However, if the primitive plan failure was simply caused by a noisy environment, like a noisy local occupancy map, or a poorly chosen GPS coordinate (e.g., a coordinate that is in an overgrown bush), we would like to avoid the complexity of updating the environment model and replanning.

To mitigate unnecessary replanning due to primitive waypoint-navigation failures, we updated the macro-action planner to be robust to failures caused by small, insignificant differences between the model and environment geometry. Specifically, we modified the macro-action planner to re-attempt similar primitive navigation actions upon primitive ac-

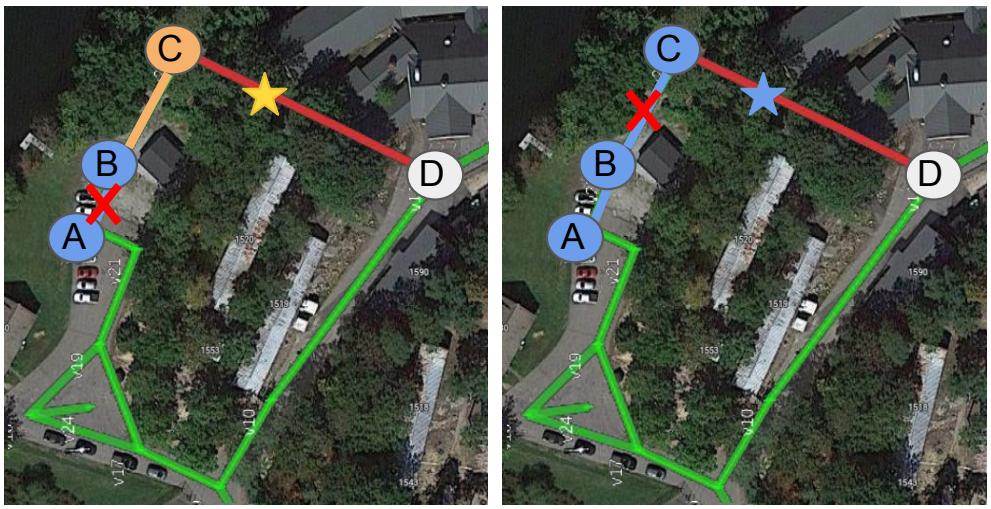
tion failure, rather than directly marking the macro-action as failed and reporting the failure to the abstract planner. The macro-action planner defined four alternative navigation goals for each location (x, y): $(x + \delta, y), (x - \delta, y), (x, y + \delta)$, and $(x, y - \delta)$. If the agent failed to navigate to the original and all alternative goals, we assumed the primitive navigation action was unsuccessful, and the macro-action failed. However, if the agent successfully navigated to the original goal or any of the alternative goals, then the primitive action was considered successful, and the macro-action continued[1]. In the experiments, we let $\delta = 0.5$m. While this macro-action planner modification was simple, it reduced the number of catastrophic failures experienced by the system, and demonstrated the importance of robustness across planning layers in hierarchical planning systems.

Variable-Duration Macro-Actions

Additionally, we developed a semi-synchronous planning method for coordinating teammates executing variable-duration macro-actions, such as executing macro-actions that represent traversals of graph edges with different weights, or executing macro-actions in the presence of real-world environment uncertainty that can impact action completion time. Unfortunately, it is not obvious how the team should proceed when one agent successfully completes a macro-action, but another teammate is in the process of executing a macro-action. While our simulated planner operated entirely asynchronously, the planner relied on being able to precisely calculate action termination times, which is not realistic during real-world deployments in uncertain environments. The goal of the semi-synchronous planner is to enable the agents to stop and replan as quickly as possible when sharing information, while also ensuring that the agents remain in valid configurations in the abstract CTP graph.

We developed a planning approach that enabled the team to execute variable-length macro-actions online. Specifically, when any agent completed a macro-action, it sent the result of the macro-action (navigation success or failure, current state, and any traversability

[1]While this modification is similar to increasing the radius of the navigation goal region, we found that it resulted in more stable primitive navigation.



a) Interrupt while executing edge A-B

b) Interrupt while executing edge B-C

Figure 5.5: **Examples of macro-action interrupts and their impacts on planning.** Examples of macro-action interrupts (received when the agents are at the red Xs) and their impacts on planning. When an agent received an interrupt message during primitive action execution, it completed its current primitive navigation action and any directly succeeding sensing actions, then terminated the current macro-action, communicated with the base station, and waited to receive an updated plan.

observations) to the centralized planner at the base station. Then, all other agents received an *interrupt*, or a message indicating that the agent should terminate its current action as soon as possible, send action results to the base station, and wait to receive a new plan. If an agent received an interrupt message at the terminal state of a primitive action, then the macro-action terminated immediately, and the agent sent the macro-action result to the planner. Otherwise, the agent completed its current primitive navigation action and any observation actions that directly succeeded the navigation action in the macro-action plan, and then terminated the current macro-action and sent the action result to the centralized planner. For example, consider the agent in Fig. 5.5 executing the macro-action of navigating from node A to node C, with the goal of sensing edge C-D. If the agent received an interrupt message when navigating between nodes A and B, the agent completed the primitive navigation action to node B, then terminated the macro-action, communicated with the base station,

and waited for a replan. However, if the agent received an interrupt message when navigating between nodes B and C, the agent completed the primitive navigation action to node C, sensed the traversability of edge C-D, and then terminated the macro-action, communicated with the base station (including the traversability observation of edge C-D), and waited for a replan. Finally, once all agents terminated their current macro-actions, the centralized planner replanned for the entire team and sent new macro-actions to each agent.

The interrupt scheme enabled the team to quickly react to new information while ensuring that each agent remained in a valid configuration in the abstract graph. If an agent terminated its current action immediately after receiving an interrupt, its state may not be valid in the graph, and it would not be obvious how to generate a new graph-based plan for the agent. In future work, we are interested in developing methods that modify the abstract graph representation based on agent locations during interrupts. For example, if an agent is interrupted while navigating along a graph edge to a waypoint, it may be possible to add a node to the abstract graph at the agent interrupt location, and to split the current graph edge into two edges that represent the traversed and untraversed portions of the current edge, respectively. Finally, while this work does not directly aim to address the research problem of intermittent communications, the interrupt-based planner is robust to minor communication delays, and its lightweight messaging is tolerant of non-catastrophic outages.

5.2.4 Level 3: Primitive Actions

Finally, the lowest level of the planner consisted of primitive action planners that generated executable robot commands. Specifically, the planners enabled robots to navigate along graph edges, sense the traversabilities of edges, and wait in place for a specific amount of time.

Navigation Actions

First, we developed a planner for navigating along a graph edge. Given the GPS coordinates corresponding to the source and target nodes of the edge, we set the agent navigation goal to be a region around the GPS coordinate of the target node; in the field trials, we used a circular region with a 3 meter radius. Then, we used a global planner, the Efficiently Adaptive State Lattice (EASL) planner [52], to generate a 2D plan between the agent's current location and the goal, where the planner used the agent's local occupancy map to avoid obstacles. Finally, we used a Model Predictive Path Integral (MPPI) controller [151] to generate agent motor commands to follow the EASL plan to the goal.

The primitive navigation planner was designed to be robust to various forms of real-world environmental uncertainty. By not constraining the agent to directly navigate along the graph edge (i.e., using a Euclidean path), we enabled the agent to react to local environment changes that did not impact global planning. For example, during field trials, an agent was able to navigate around an overgrown bush on the Euclidean path to its next node without impacting other levels of the planner. Additionally, the hierarchical approach enabled us to tune low-level planner parameters for individual agents without impacting the higher levels of the planning pipeline. This was especially useful for the MPPI planner, which used various parameters related to agent dynamics that varied across our vehicles.

Observation Actions

Second, we developed an observation action that enabled an agent to sense the local traversability of an adjacent edge. We used the current agent occupancy map and the EASL planner to generate a 2D plan from the current agent location to a point 20 meters along the unknown edge. Then, we the compared the EASL plan cost to the Euclidean distance between the two points. If the EASL plan cost exceeded 120% of the Euclidean distance cost, the edge was marked untraversable; otherwise, the edge was marked traversable. Intuitively, this method tested if an agent could navigate along a graph edge without taking a significant

detour, based on its local occupancy map. While the edge observation function used in our tests was primitive, other functions that report binary edge traversability based on local observations could be used in our pipeline. In future work, we are interested in generating binary observations based on other local traversability observation functions (e.g. [48]). We are also interested in exploring semantically informed observation functions (e.g., a function that marks an edge as traversable unless a suspicious barrel is present in the local map or agent camera images).

Wait Actions

Finally, we developed a wait action, which caused an agent to wait in place for a specified duration. This enabled an agent to wait in place to receive traversability observations from a teammate before making a decision about its next action. The wait action prevented costly backtracking when an agent did not have sufficient information to immediately make a good planning decision, but a teammate was able to inexpensively sense the relevant information.

5.3 Deployments and Results

We evaluated the system in real-world deployments in one semi-structured outdoor environment (Camp Buckner) and one outdoor park environment (Magazine Beach) over 18 trials and three configurations; images of the deployment environments are shown in Fig. 5.1-a and Fig. 5.8-c, respectively. We evaluated our approach on a heterogeneous, two agent team consisting of a Clearpath Jackal and a Clearpath Husky. Both agents were equipped with an Intel NUC11PH computer, a Microstrain 3DM-GX5-AHRS IMU, and a u-blox M8U GPS module, which provided GPS-based agent position estimates during the trials. Additionally, the Jackal was equipped with an Ouster OS1-32 LIDAR sensor, and the Husky was equipped with a Velodyne VLP-16 LIDAR sensor. Both agents were equipped with a core autonomy pipeline, which included GPS-based state estimation and an Omnimapper-

based [145], GPS-enabled local mapping pipeline. All multiagent planning occurred on a centralized base station laptop, and the base station and agents communicated using a ROS multimaster system over a Silvus radio network. Finally, in the Camp Buckner experiments, it was assumed that the Jackal speed was 8x faster than the Husky speed. In the Magazine Beach experiments, it was assumed that the Jackal speed was 2x faster than the Husky speed, as this value more accurately modeled our experiences with relative agent speeds in the grassy Magazine Beach terrain.

5.3.1 Deployment Results: Camp Buckner

In the Camp Buckner environment, we evaluated qualitative planning performance for the agents when planning using the hierarchical planning system in a semi-structured outdoor environment. In Fig. 5.6, we compare the graph-based macro-actions selected by the collaborative and baseline planners during two successful planning trials. In the baseline, non-collaborative planning trial, the Jackal navigated directly to its goal, while the Husky attempted to navigate to its goal via the unknown edge (red). After the Husky arrived at the edge and sensed that it was untraversable, the Husky backtracked and navigated to its goal via a long, traversable path. In the collaborative planning trial, the Jackal diverted from the shortest path to the goal to sense the traversability of the unknown edge (red) at the gold star and relay the information to the Husky, while the Husky waited for information about the edge. After receiving the Jackal observation that the edge was not traversable, the Husky navigated directly to its goal via the long, traversable path. In Fig. 5.7, we plot the trajectories traversed by the agents while executing the macro-action plans; trajectories, starts, and goals are approximately scaled and hand-aligned over the overhead image. Note that in the baseline trial, the Husky became stuck navigating up a steep hill while attempting to execute the final two edges to the goal. Also, in some trials, MPPI became stuck while trying to navigate around an obstacle (e.g., a car or a bush), and an operator briefly teleoperated the robot to enable the robot to continue to make progress towards completing the

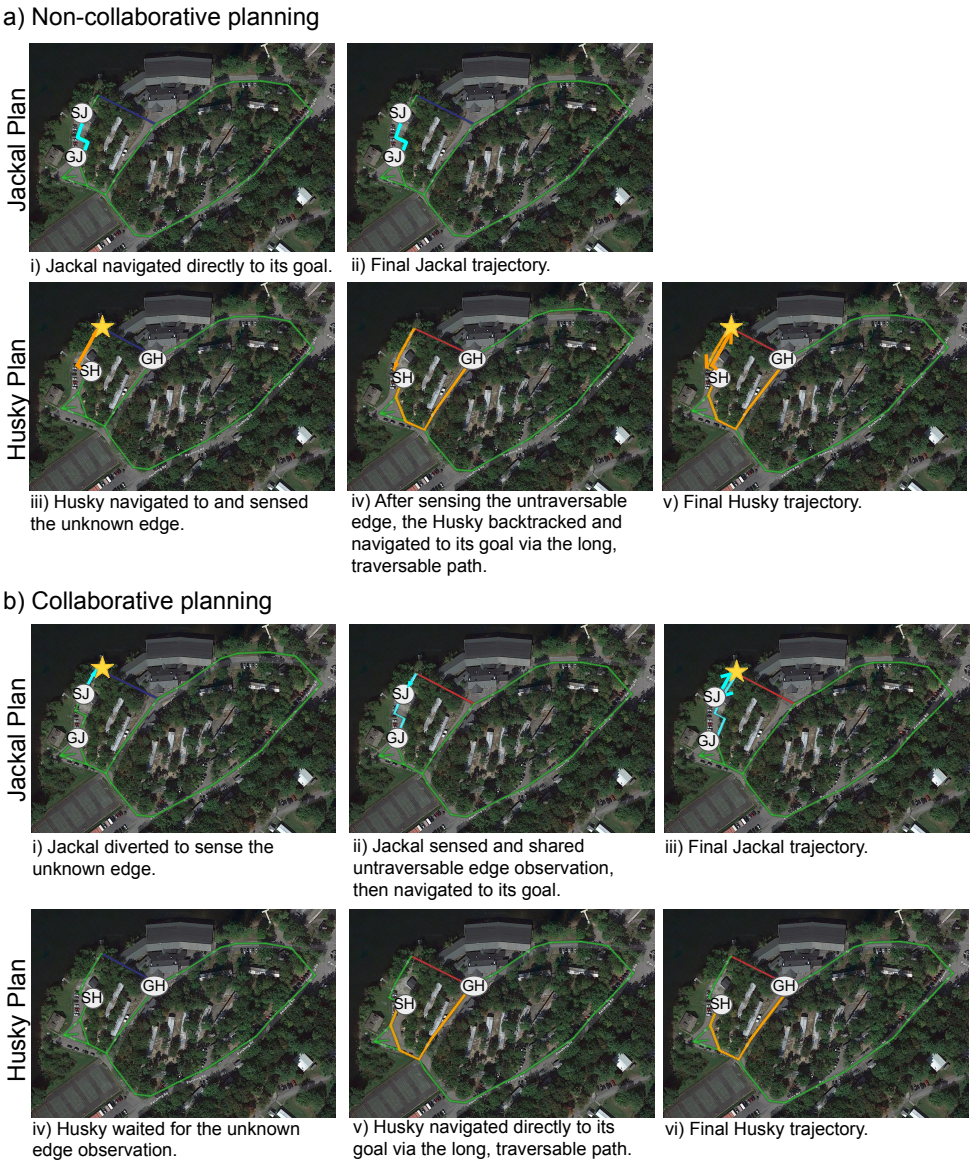
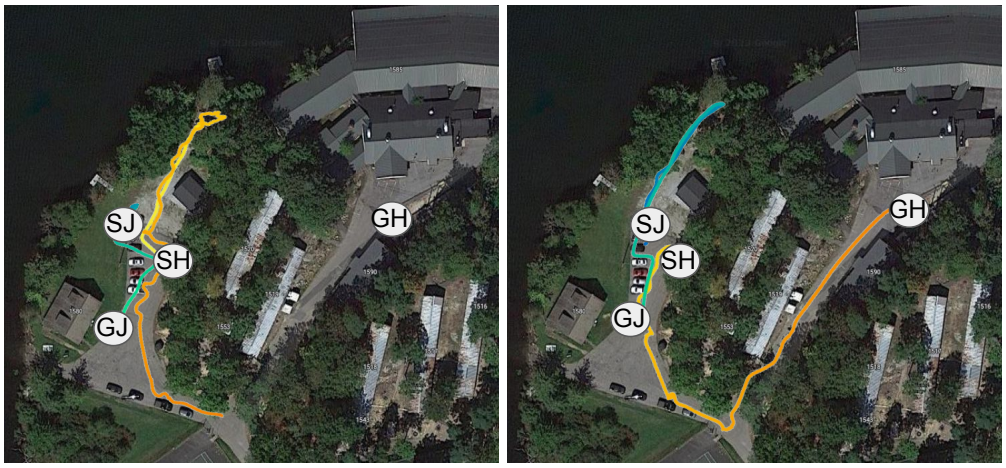


Figure 5.6: **Graph-based planning results, Camp Buckner.** a) Graph-based plans selected by the non-collaborative baseline planner, visualized over the abstract navigation graph. The Jackal (blue) with start SJ and goal GJ navigated directly to its goal, while the Husky (orange) with start SH and goal GH navigated to the unknown edge, sensed that it was untraversable, and then backtracked to the goal via the long, traversable path. b) Graph-based plans selected by the collaborative planner, visualized over the abstract navigation graph. The Jackal (blue) with start SJ and goal GJ diverted to the gold star to sense the unknown edge, and shared the untraversable edge observation with the Husky. Then, the Jackal navigated to its goal. The Husky (orange) with start SH and goal GH waited to receive the edge observation from the Jackal, then navigated directly to its goal via the long, traversable path. Note that some trajectories were hand-annotated from raw agent outputs for clarity and due to agent/ground station communication drops.



a) Non-collaborative trajectories b) Collaborative trajectories

Figure 5.7: **Executed trajectories, Camp Buckner.** Trajectories traversed by the Jackal (blue-green) and Husky (yellow-orange) while executing the graph-based plans described in Fig. 5.6. Real-world trajectories are similar to the graph-based plans. Trajectories, starts, and goals are approximately scaled and hand-aligned on the overhead image.

primitive action. For these reasons, we do not report the total execution times or distances traveled for the agents. However, it is possible to compare the plan costs for the agents in the abstract graph; we report the graph distance of each agent's planned trajectory, as well as the durations of planned wait actions, in Table 5.1. In the collaborative trial, the Jackal traveled further to sense the edge for the Husky, significantly reducing the graph distance traveled by the Husky. In the non-collaborative trials, the Jackal graph distance traveled was very low, but at the expense of a longer Husky trajectory. The results are consistent with the intuition that optimizing a team makespan requires a planner to trade off between the performances of the different agents.

While we were able to demonstrate successful collaborative multiagent planning, we also observed various failures during testing. In Table 5.2, we summarize different failure types that resulted in trial termination and their frequencies.

	Graph Distance traveled, Husky (m)	Graph Distance traveled, Jackal (m)	Husky Wait Time (s)	Jackal Wait Time (s)
Collaborative Planner	156.71	100.04	10.0	0.0
Non-Collaborative Planner	257.97	25.83	0.0	0.0

Table 5.1: **Comparison of collaborative and baseline plans, Camp Buckner.** Agent wait times in seconds and plan costs in meters in the abstract graph when planning using the Collaborative and Non-Collaborative planners.

Quantity	Type
4	Low-level (EASL/MPPI) navigation failure
1	Incorrect observation function output

Table 5.2: **Summary of failed trials, Camp Buckner.**

5.3.2 Deployment Results: Magazine Beach

In the Magazine Beach environment, we evaluated the qualitative and quantitative team planning performance of the agents over 16 collaborative and non-collaborative trials in two different planning scenarios. We demonstrated that our collaborative approach reduced team execution times by 13.3% and 24.6% as compared to the non-collaborative baseline in the scenarios, respectively.

Scenario 1

First, we tested our approach for a scenario where the Husky and Jackal started in the same region of the environment, and the faster Jackal sensed and shared the traversability of an unknown forest edge with the Husky. A diagram of the scenario, named Scenario 1, is shown in Fig. 5.8-a. We executed planning in the scenario five times for both the collaborative and non-collaborative planners, resulting in ten total trials.

We qualitatively evaluated the collaborative and non-collaborative agent plans generated for the scenario. In Figs. 5.9 and 5.10, we compare the graph-based macro-actions selected

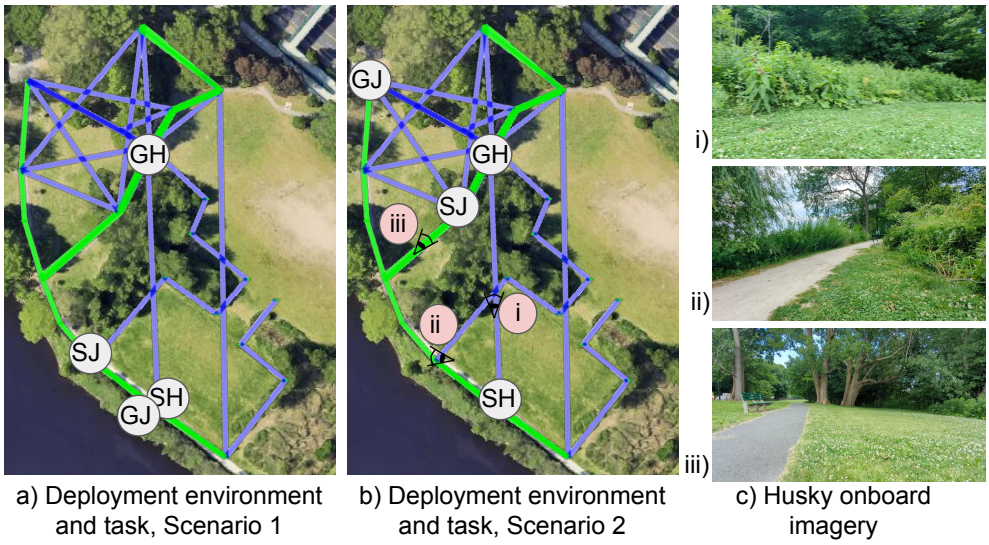


Figure 5.8: **The Magazine Beach environment.** Overhead images of the Magazine Beach deployment environment, overlaid with the automatically generated route graph used during all Magazine Beach planning trials, where green edges are known traversable, and blue edges are probabilistically traversable. a) The task executed in Scenario 1. b) The task executed in Scenario 2. c) Examples of Husky onboard imagery at key locations in the trials; images correspond to the locations labeled in (b).

by the baseline and collaborative planners during a representative planning trial. In the baseline trial, the Jackal navigated directly to its goal, while the Husky attempted to navigate to the goal via the unknown edge through the forest, and then backtracked to the goal after sensing that the edge was untraversable. In the collaborative planning trial, the fast Jackal navigated to and sensed the unknown forest edge while the Husky waited for additional information. After receiving the observation that the forest edge was untraversable, the Husky then navigated directly to the goal via the long but traversable pedestrian pathway.

In Fig. 5.11, we plot the executed agent trajectories for the five collaborative trials and the five baseline trials; note that significant pose drift occurred in some trials. The agent execution traces indicate that the agents successfully completed the macro-action based plans described in Figs. 5.9 and 5.10.

We also quantitatively evaluated total trial times, execution times, and planning times for the five collaborative and non-collaborative trials in the scenario, where total trial times

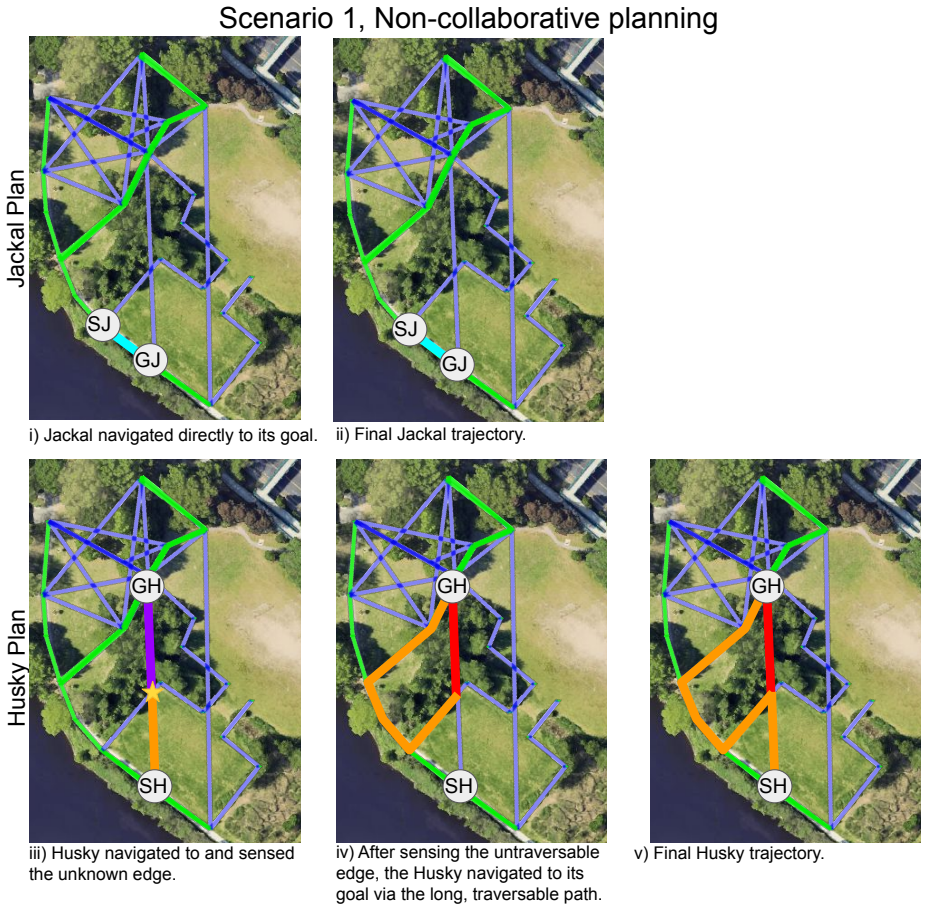


Figure 5.9: **Non-collaborative graph-based planning results, Magazine Beach, Scenario 1.** Graph-based plans selected by the non-collaborative baseline planner, visualized over the abstract navigation graph. The Jackal (blue) with start SJ and goal GJ navigated directly to its goal, while the Husky (orange) with start SH and goal GH navigated to the unknown forest edge, sensed that the edge was untraversable, and then turned around and navigated to the goal via the pedestrian pathway.

were defined as the total elapsed time from the start to the end of the trial, planning times were defined as the time spent computing abstract plans, and execution time was defined as the difference between the total trial time and the planning time. In Table 5.3, we report average times, standard deviations, and standard errors of the mean for the collaborative and non-collaborative planning trials. Scatter plots of total trial times, execution times, and planning times are shown in Fig. 5.12. While the collaborative planning approach resulted in a 13.3% decrease in execution time as compared to the baseline planning approach, the planner generated plans 7.8x more slowly than the non-collaborative planner, and resulted

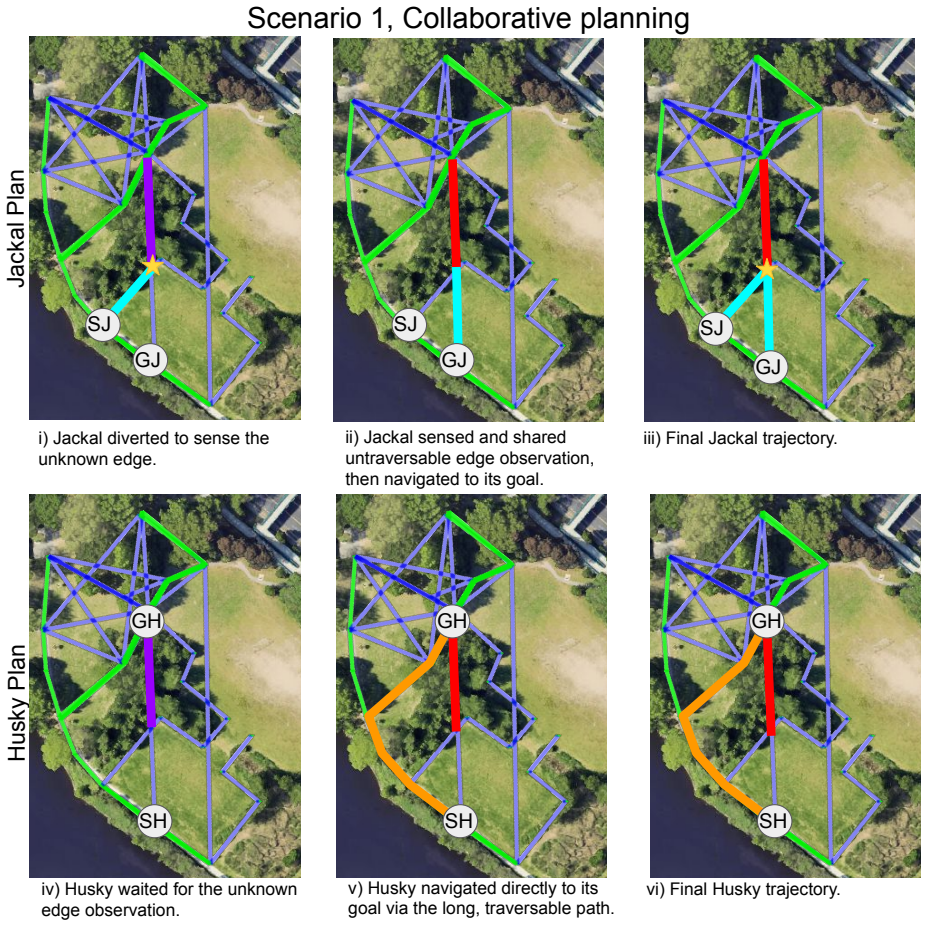


Figure 5.10: **Collaborative graph-based planning results, Magazine Beach, Scenario 1.** Graph-based plans selected by the collaborative planner, visualized over the abstract navigation graph. The Jackal (blue) with start SJ and goal GJ diverted to the gold star to sense the traversability of the unknown forest edge, and then shared the untraversable edge observation with the Husky. Then, the Jackal navigated directly to its goal. The Husky (orange) with start SH and goal GH waited to receive the edge observation from the Jackal, and then navigated directly to its goal via the pedestrian pathway.

in a 0.5% increase in total trial times as compared to the baseline.

a) All non-collaborative trajectories

b) All collaborative trajectories

Figure 5.11: **Executed trajectories, Magazine Beach, Scenario 1.** Trajectories executed by the Jackal (blues) and the Husky (reds) while executing the graph-based plans described in Figs. 5.9 and 5.10. Despite significant pose drift in some trials, the shapes of the trajectories indicate that the agents successfully completed the graph-based plans. Trajectories were plotted in the GPS frame, then approximately scaled and hand-aligned on the overhead image.

	Non-Collaborative			Collaborative		
	Trial Times	Execution Times	Planning Times	Trial Times	Execution Times	Planning Times
Average Time (s)	**261.9**	256.74	**5.16**	263.18	**222.51**	40.66
STD Time (s)	20.14	20.16	0.09	12.23	11.34	6.24
SEM Time (s)	10.07	10.08	0.04	6.12	5.67	3.12

Table 5.3: **Aggregate timing results, Magazine Beach, Scenario 1.** Results were calculated for 5 collaborative and 5 non-collaborative trials. While the collaborative planner resulted in lower team execution times, collaborative planning times were slower than baseline planning times, resulting in similar total trial times for both planners.

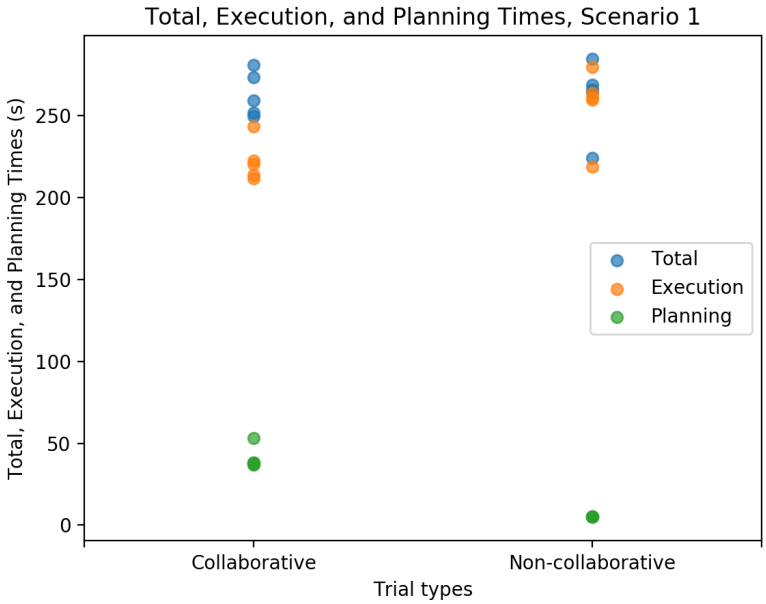


Figure 5.12: **Scatter plots of timing results, Magazine Beach, Scenario 1.** Scatter plots of individual trial times, execution times, and planning times for the five collaborative and five non-collaborative trials in Scenario 1.

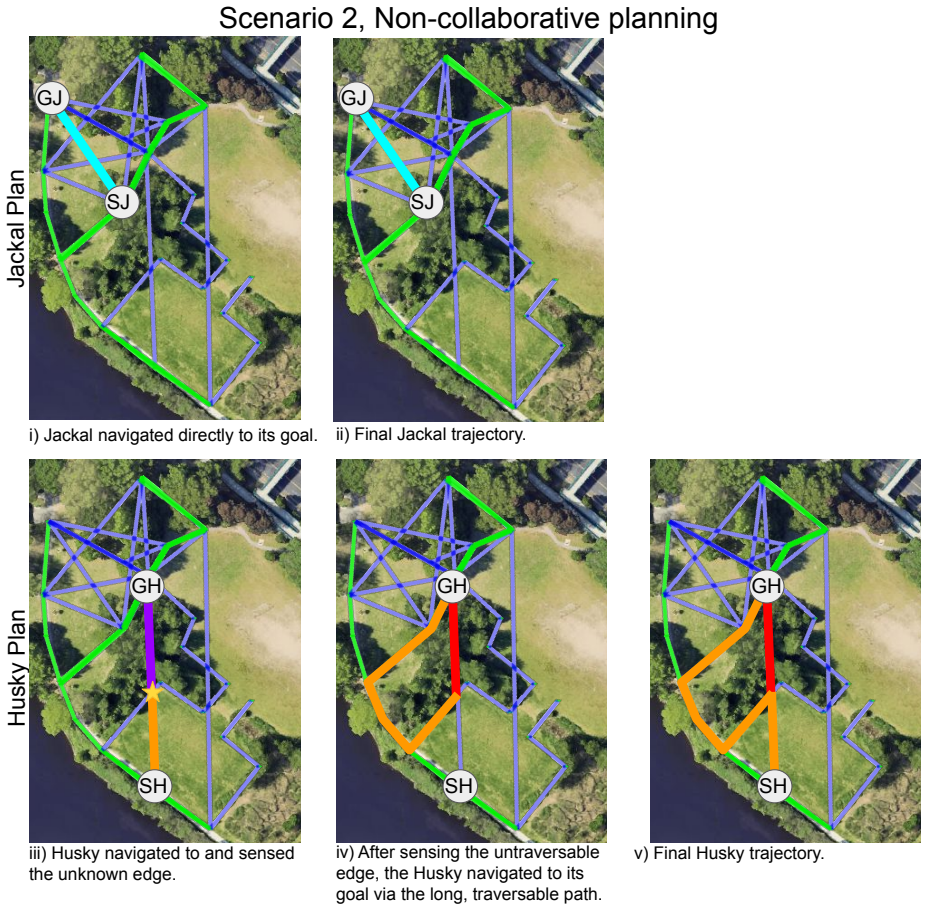


Figure 5.13: **Non-collaborative graph-based planning results, Magazine Beach, Scenario 2.** Graph-based plans selected by the non-collaborative baseline planner, visualized over the abstract navigation graph. The Jackal (blue) with start SJ and goal GJ navigated directly to its goal, while the Husky (orange) with start SH and goal GH navigated to the unknown forest edge, sensed that it was untraversable, then turned around and navigated to the goal via the pedestrian pathway.

Scenario 2

In Scenario 2, the Husky and the Jackal started on opposite sides of a forest, and the Jackal took a small detour to sense the far side of the forest for the Husky; a diagram of the scenario is shown in Fig. 5.8-b. This scenario is especially interesting because it demonstrates the ability of our planner to model and use global information in collaborative planning; in the scenario, the agents did not have line-of-sight to each other while planning for and executing collaborative plans. We executed planning in the scenario three times for both the

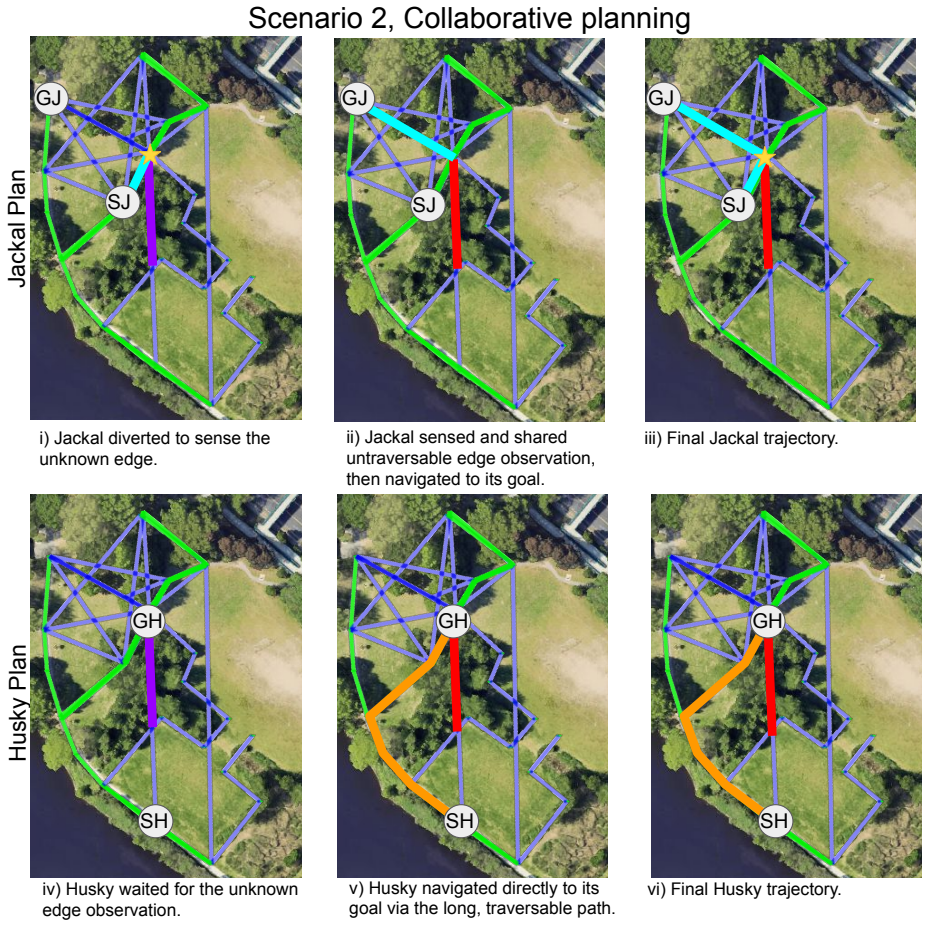


Figure 5.14: **Collaborative graph-based planning results, Magazine Beach, Scenario 2.** Graph-based plans selected by the collaborative planner, visualized over the abstract navigation graph. The Jackal (blue) with start SJ and goal GJ diverted to sense the traversability of the unknown forest edge at the goal star, and then shared the untraversable edge observation with the Husky. Note that the global representation of the uncertain edge enabled the Jackal to sense and share useful environment information from the side of the forest opposite the Husky. Then, the Jackal navigated directly to its goal. The Husky (orange) with start SH and goal GH waited to receive the edge observation from the Jackal, then navigated directly to its goal via the pedestrian pathway.

collaborative and non-collaborative planners, resulting in six total trials.

We qualitatively evaluated collaborative and non-collaborative agent plans in the scenario. In Figs. 5.13 and 5.14, we compare the graph-based macro-actions selected by the baseline and collaborative planners during a representative planning trial. In the baseline trial, the Jackal navigated directly to its goal, while the Husky attempted to navigate to the goal via the unknown edge through the forest, and then backtracked to the goal after sensing

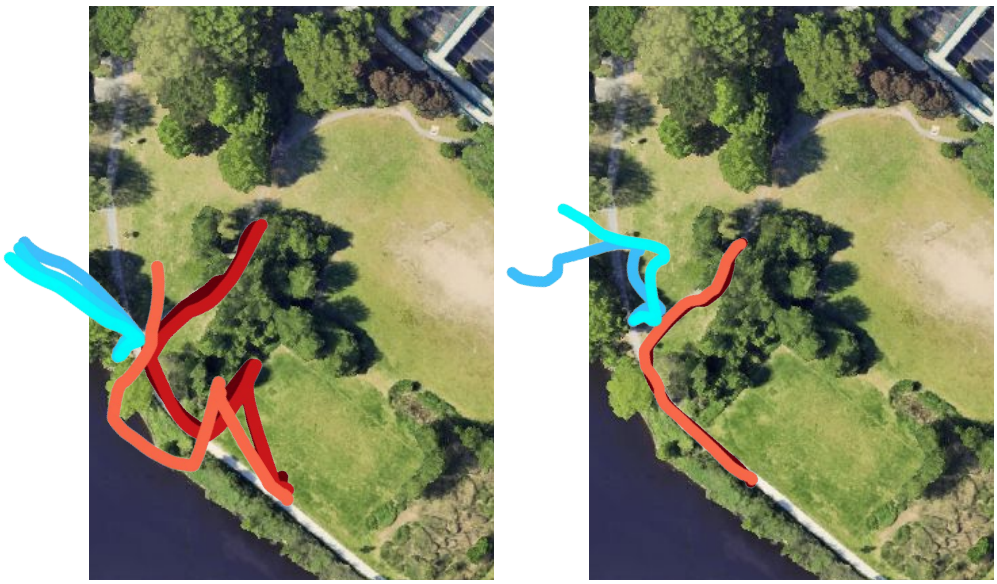


Figure 5.15: **Executed trajectories, Magazine Beach, Scenario 2.** Trajectories executed by the Jackal (blues) and the Husky (reds) while executing the graph-based plans described in Figs. 5.13 and 5.14. Despite significant pose drift in some trials, the shapes of the trajectories indicate that the agents successfully completed the graph-based plans. Trajectories were plotted in the GPS frame, then approximately scaled and hand-aligned on the overhead image. Note that due to a data collection error, only two of the three collaborative trajectories are visualized.

that the edge was untraversable. In the collaborative planning trial, the Jackal navigated to the far side of the forest to sense and share information about the unknown forest edge while the Husky waited for additional information. After sensing the untraversable forest edge, the Jackal navigated directly to its goal, and the Husky navigated via the pedestrian pathway to its goal.

In Fig. 5.15, we plot the executed agent trajectories for two collaborative trials and three baseline trials; note that significant pose drift occurred in some trials, and that due to a data collection error, pose information was only collected during two of the three collaborative trials. The agent execution traces indicate that the agents successfully executed the macro-action based plans described in Figs. 5.13 and 5.14.

Finally, we quantitatively evaluated total trial times, execution times, and planning times

	Non-Collaborative			Collaborative		
	Trial Times	Execution Times	Planning Times	Trial Times	Execution Times	Planning Times
Average Time (s)	308.7	303.1	**5.6**	**298.01**	**228.41**	69.6
STD Time (s)	6.08	5.94	0.14	5.25	4.79	0.93
SEM Time (s)	4.3	4.2	0.1	3.71	3.38	0.66

Table 5.4: **Aggregate timing results, Scenario 2.** Results were calculated for 3 collaborative and 3 non-collaborative trials. The collaborative planner resulted in lower execution times and marginally lower total trial times as compared to the non-collaborative baseline.

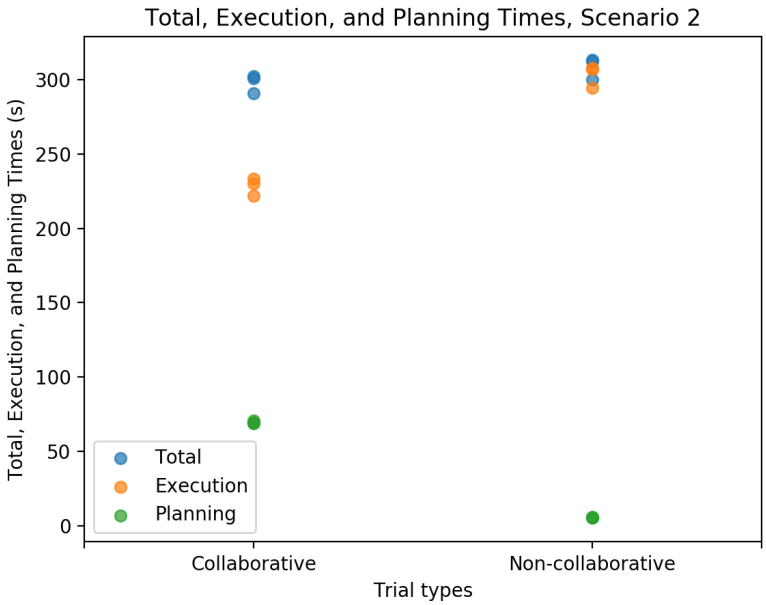


Figure 5.16: **Scatter plots of timing results, Magazine Beach, Scenario 2.** Scatter plots of individual trial times, execution times, and planning times for the three collaborative and three non-collaborative trials in Scenario 2.

over the three collaborative and three non-collaborative trials. In Table 5.4, we report average times, standard deviations, and standard errors of the mean for the trials. Scatter plots of total trial times, execution times, and planning times are shown in Fig. 5.16. In this scenario, the collaborative planning approach resulted in a 3.5% decrease in total trial times, and a 24.6% decrease in total execution times as compared to the non-collaborative baseline; however, the collaborative planner was 12.4x slower than the baseline.

5.4 Discussion

While abstract planning representations are often necessary for efficiently solving complex planning problems, like problems in large or uncertain environments, or problems that involve teams of agents, significant challenges arise when executing abstract plans on real agents in real-world environments. First, building a representative abstraction of a real-world environment that is efficient for high-level planning, but that produces plans that can be translated into real-world executions, is an open research question. We are exploring methods for automatically generating abstract navigation graphs from polygonal environment models [149] and overhead images. While this will reduce operator burden, it is unlikely that the auto-generated graphs, like our hand-drawn graphs, will be fully representative of planning environments unless they have access to additional environment information, some of which may only be available during planning. We are exploring techniques to modify the navigation graph abstraction online based on local sensor data.

This work also demonstrated that robust hierarchical planning systems can overcome some of the challenges that arise from imperfect planning abstractions. Specifically, by increasing the robustness of planners at different levels of the planning hierarchy, we were able to overcome small discrepancies in the planning abstraction (like having a navigation waypoint in a bush next to a road, instead of on the road itself) without causing catastrophic planning failures. In future work, we are interested in exploring other ways to make our hierarchical planning system more robust, including reconsidering traditional definitions of plan failure at each level in the planning hierarchy.

Finally, the proposed approach was developed for a centralized team with full communication. While our macro-action message passing scheme was lightweight, and our interrupt planner was capable of handling minor communication delays, in future work, we would like to modify the approach to explicitly handle more challenging communication environments.

5.5 Conclusion

As roboticists move towards deployments of bigger teams in larger, more complex uncertain environments, it will be necessary to develop abstract representations for complex planning problems that efficiently model long length scale, uncertainty-aware global information for planning, but that are still amenable to robust plan execution in real-world environments. In this work, we demonstrated the ability of a global, uncertainty-aware hierarchical planning system to bridge the gap between abstract planning and real-world execution. We deployed a collaborative, uncertainty-aware multiagent planner on a Jackal and Husky team using the hierarchical planning system, and demonstrated collaborative planning performance in two complex, real-world outdoor environments.

Chapter 6

Learned Value Function Approximations for Efficient Collaborative Multiagent Planning on Uncertain Graphs

In Chapter 4, we developed a planner that used multiagent macro-actions and single-agent value function approximations for efficient multiagent collaboration in unknown graphs. However, we used Monte Carlo (MC) simulations to calculate the single-agent value functions used in our approximations, which were calculated for each macro-action at each planning step. Unfortunately, generating MC-based approximations is expensive for large teams and environments, and can lead to computational intractability, especially when the approximations are used as a subroutine in a complex planning algorithm.

In this chapter, we present a preliminary method for learning a single-agent value function approximator to increase the efficiency of the uncertainty-aware collaborative multiagent planner described in Chapter 4. We develop a method for learning belief-dependent value function approximations on uncertain graphs. We demonstrate the approximations in the collaborative planner in toy graph and island road network domains, and demonstrate a 36.6% improvement in planning time as compared to a Monte Carlo (MC) rollout-based

approximation approach for team planning in the toy domain, while finding plans of similar quality.

6.1 Motivation

We would like to enable a collaborative multiagent team to navigate efficiently in an uncertain environment, modeled as a roadmap or a stochastic motion graph where some edges may be untraversable. In Chapter 4, we enabled more tractable collaborative planning for such a team by showing how specific kinds of multiagent macro-actions could reduce the planning depth at which multiagent collaborations occurred. We designed approximations, based on single-agent value functions with and without information gained through collaboration, to quickly identify a small set of multiagent macro-actions that were likely to improve team planning performance.

We used Monte Carlo (MC) simulations to calculate the single-agent value functions used in our approximations, which were calculated for each macro-action at each planning step. However, generating MC-based approximations is expensive for large teams and can lead to computational intractability, especially when used as a subroutine in a complex planning algorithm. Recently, approximation techniques that explicitly reason about environment structure, rather than implicitly discovering the structure via planning in rollouts, have been used to calculate value functions in structured environments [39, 140].

We hypothesize that our graph-based problem representation is suitable for learning single-agent value functions, and that learned value function approximators are more efficient for planning than other simulation-based approximation techniques in domains of interest. We propose to learn single-agent value functions on stochastic graphs, and then use the learned value functions in the multiagent macro-action selection step of the collaborative multiagent planner under uncertainty described in Chapter 4. The learned value functions capture the benefit of sensing information for individual agents, which can be used

to prioritize collaborative sensing actions for the team. Our approach uses an inefficient, simulation-based approximator to generate a dataset of graph and value function pairs, and then uses supervised learning to train a learned value function approximator for use in online planning. Finally, we demonstrate the learned approximations in the collaborative planner described in Chapter 4.

6.2 Problem Statement

In Chapter 4, we used single-agent value function approximations to quickly identify collaborative macro-actions that were likely to improve collaborative team navigation in uncertain environments. The macro-actions and approximations enabled a planner to quickly identify collaborative sensing behaviors that were likely to improve team planning performance. Unfortunately, computing the single-agent value functions required by the multiagent planner in Eq. 4.13, which approximate planning performance with and without immediate traversability information about an edge e, $V^{b^{z_e}} = \sum_{z_e \in Z} p(z_e) V^{\pi_{sa}}(v, b^{z_e} | z = z_e)$ and $V^b = V^{\pi_{sa}}(v, b)$, was computationally expensive[1]. Monte Carlo (MC) simulation was used to calculate the approximate value functions, $\hat{V}^{b^{z_e}} \approx V^{b^{z_e}}$ and $\hat{V}^b \approx V^b$. However, generating $\hat{V}^{b^{z_e}}$ and $\hat{V}^b$ required the planner to simulate multi-step planning in many environment realizations. The computational burden of MC simulation increased as the size of the environment and team increased, since the number of value function approximations computed per planning iteration scales linearly with both the number of unknown edges and the number of stochastic agents in the problem.

[1]For notational simplicity, we adopt value function shorthands for the remainder of the section. We additionally assume that observations are of edge traversabilities, and use the shorthand z_e to represent such observations.

6.3 Learned Approximations for Multiagent Planning Under Uncertainty

To increase planning efficiency and enable scaling to larger environments and teams, we propose to learn the single-agent value functions used in the approximation of Eq. 4.5. In many environments of interest, the structure of the planning problem, such as the structure of the underlying graph G, the agent's current belief b, location v, and goal v_G, and other problem properties Γ (e.g., graph edge costs c and traversability probabilities ρ), correlate with the single-agent value function approximations with and without access to observations z_e of the traversability of edge e, $\hat{V}^{b^{z_e}}$ and $\hat{V}^b$,

$$\hat{V}^{b^{z_e}}, \hat{V}^b = f(G, b, v, v_G, \Gamma), \tag{6.1}$$

where the agent's belief b over edge weathers captures the history of prior edge observations.

Prior work indicates that it may be possible to learn value functions from data [39]. For our problem, we can use expensive MC simulation during offline planning to generate a dataset $\mathcal{D}$ of planning information and value function pairs,

$$\mathcal{D} = \{(G, v, v_G, \Gamma, \hat{V}^{b^{z_e}}, \hat{V}^b)_1, (G, v, v_G, \Gamma, \hat{V}^{b^{z_e}}, \hat{V}^b)_2, ..., (G, v, v_G, \Gamma, \hat{V}^{b^{z_e}}, \hat{V}^b)_d\}, \tag{6.2}$$

where d is the number of examples in the dataset. Then, we use the dataset to learn a value function predictor ϕ that is efficient for use during online planning:

$$\tilde{V}^{b^{z_e}}, \tilde{V}^b \approx \phi(G, v, v_G, \Gamma, \alpha), \tag{6.3}$$

where $\tilde{V}^{b^{z_e}}$ and $\tilde{V}^b$ are the predictions of the approximations $\hat{V}^{b^{z_e}}$ and $\hat{V}^b$, respectively, and α are the parameters of the learned predictor. The learned predictions then replace the Monte Carlo-based predictions in the multiagent planner.

6.3.1 Graph Neural Networks for Stochastic Motion Planning

Graph Neural Networks

In this work, we select a Graph Neural Network as the form of ϕ. Graph Neural Networks (GNNs) are a powerful deep learning tool that have been demonstrated to effectively capture relationships in graph-based data [153, 36]. Since their introduction, GNNs have been used to model correlations on graphs for problems in chemistry and materials science (e.g. predicting formation energy of bulk crystals [43], absorption energy of alloy surfaces [43], chemical toxicity [123], and adverse drug side effects [123]), physics (e.g., predicting future states and estimating potential energy of n-body systems, and predicting future states and mass for bouncing balls [24, 9, 10]), social network modeling [89], planning [67, 88, 87, 61, 36], and more.

The flow of information in GNNs is built on *message passing*, which can be thought of as a generalization of an image-space convolutional filter to graph data. Each iteration of message passing includes two distinct steps, a feature generation step and feature aggregation step. In the feature generation step, a function is applied to node embeddings to generate a message, or a low-dimensional representation of information to share with neighbors; one common feature function is linear projection [36]. In the feature aggregation step, nodes receive messages from their neighbors, and apply an aggregation function to incorporate the new messages into the current node embedding; some common aggregation functions include summation, minimization, and maximization [36]. By iteratively sharing messages over multiple layers in the GNN, nodes send and receive information within an ℓ-hop neighborhood, where ℓ is the number of graph layers. In practice, a number of more complex feature and aggregation functions have been proposed; we refer the reader to [36] for a detailed discussion of different GNN layers.

GNNs have been demonstrated to be particularly effective in the context of navigation. GNNs have been used to solve routing and routing-adjacent problems on graphs, like the

shortest paths problem and the Travelling Salesman Problem (TSP) [61, 36]. Additionally, the internal structure of the GNN has been shown to align well with the computational structure of dynamic programming, an algorithm that is commonly used to calculate solutions to the shortest paths problem [154].

Graph Neural Networks for Value Function Approximation

To solve the problem of value function approximation on stochastic graphs, we select a Gated Graph Convolutional Network (GatedGCN) [22] as the structure of ϕ, as this GNN variant maintains graph edge features, which enables the network to explicitly represent and reason about edge properties such as edge traversability and cost. Additionally, we take advantage of the ability of GNNs to apply graph node and edge operations in parallel to parallelize our value function computations. Our objective is to learn value functions when agents do or do not have access to information about a specific edge, and the GNN can parallelize calculations of the approximations for each edge. Formally, we can model the parallel value function approximation procedure:

$$\tilde{\mathbf{V}}^{\mathbf{bze}}, \tilde{\mathbf{V}}^{\mathbf{b}} \approx \phi(G, v, v_G, \Gamma, \alpha), \tag{6.4}$$

where $\tilde{\mathbf{V}}^{\mathbf{bze}} = (\tilde{V}^{b_{ze_0}}, \tilde{V}^{b_{ze_1}}, ..., \tilde{V}^{b_{ze_{m-1}}})$, and $\tilde{\mathbf{V}}^{\mathbf{b}} = (\tilde{V}_0^b, \tilde{V}_1^b, ..., \tilde{V}_{m-1}^b)$. Note that with perfect prediction, all value function approximations that do not have access to additional information will have equal value, i.e., $\tilde{V}^b = \tilde{V}_0^b = \tilde{V}_1^b = ... = \tilde{V}_{m-1}^b$.

6.3.2 Neural Network Structure and Optimization

Network Structure

The network inputs are encoded as an undirected graph with the structure of G. Each vertex and edge is augmented with a feature vector that encodes the additional network inputs Γ. Vertex feature vectors encode the metric x and y locations of the current vertex, the agent

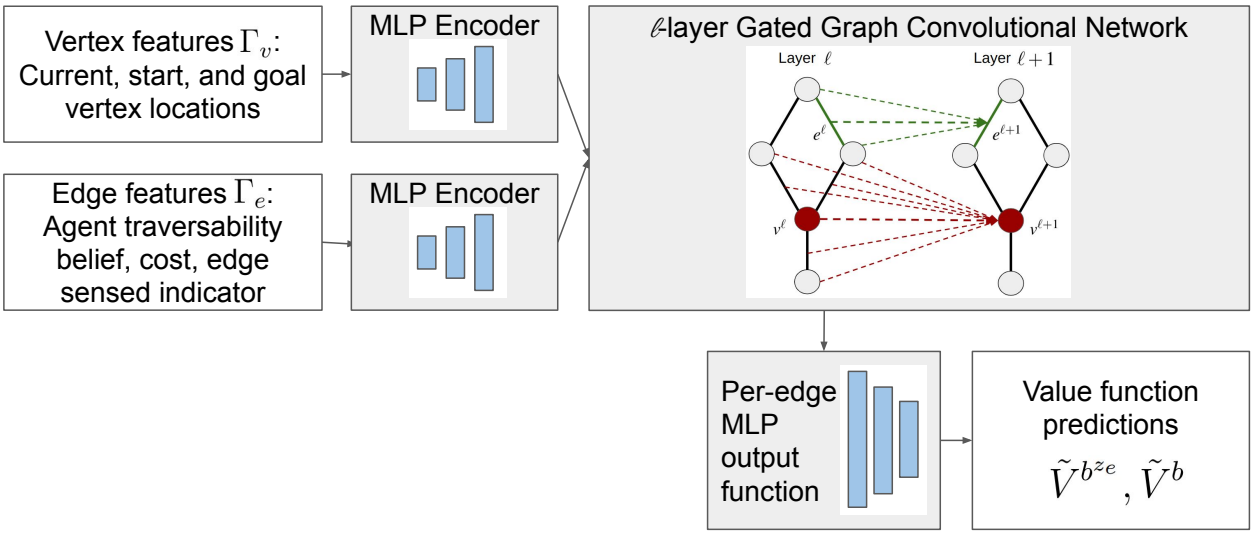


Figure 6.1: **GNN structure.** The model conducts message passing using gated graph convolutional layers, and then outputs per-edge value functions by passing local node and edge features into an output MLP. Image of GNN layers adapted from Joshi et al. [60].

start, and the agent goal, $\Gamma_v = \{x_v, y_v, x_{v_s}, y_{v_s}, x_{v_G}, y_{v_G}\}$. Edge feature vectors encode the edge cost, edge traversability belief, and an indicator denoting whether or not the edge has been sensed, $\Gamma_e = \{c_e, b_{\rho_e}, \mathbb{1}_e^{sensed}\}$. Network inputs are projected into feature space, then passed into an ℓ-layer GatedGCN, which applies ℓ graph convolutions to generate updated node and edge feature vectors. Then, a 1-layer Multilayer Perceptron (MLP) is applied to edge and adjacent-node feature vectors to recover per-edge value functions. Our network structure is visualized in Fig. 6.1.

Loss Function

Our network optimizes α to minimize the mean squared error between predicted and ground truth value functions via back-propagation and stochastic gradient descent with mini-batches. Additionally, to encourage the network to generate predictions that accurately model the differences between value functions with different beliefs, we introduce an additional term in the lost function that penalizes inaccuracies in the difference between value functions for

planning with and without additional information. Formally, we optimize the loss function:

$$L = [\tilde{V}^{b^{ze}} - \hat{V}^{b^{ze}}]^2 + [\tilde{V}^{b} - \hat{V}^{b}]^2 + \beta \times \left[[\tilde{V}^{b^{ze}} - \tilde{V}^{b}] - [\hat{V}^{b^{ze}} - \hat{V}^{b}]\right]^2, \tag{6.5}$$

where β is a weighting factor.

6.3.3 Using Value Function Approximations for Online Planning

The learned value function approximations can be used in two ways for planning. First, we can use the approximations to identify high-value information-gathering policies from the space of policies considered during multiagent planning by calculating approximations of Δ, selecting a threshold γ, and using only macro-actions where $\Delta > \gamma$. Second, we can interpret the learned functions as heuristics for greedy multiagent planning. Here, the learned functions are used to approximate expected team makespans with and without each of the information-gathering macro-actions, and the information-gathering macro-actions that most reduce makespan, if any, are greedily selected. We also used Monte Carlo approximations to determine whether or not the agents should wait for more information, and compared planning results when using the approximations to generate a policy class for replanning and when using the approximations to greedily decide whether or not an agent should wait for more information.[2]

6.4 Experiments

In this section, we describe the experimental procedure used to benchmark our learned value function approximator against a Monte Carlo simulation-based value function approximator in two domains, demonstrate the use of the learned functions in multiagent CTP planners, and report aggregate metrics.

[2]While these functions could also be learned, the number of wait actions scales linearly with the number of stochastic agents in the environment, and does not scale with the size of the environment, so learning these approximations was not prioritized.

6.4.1 Clusters Environment Setup

First, we demonstrated our approach for a 1-GV/2-AV team navigating in a simulated graph dataset, where each graph consisted of 1-7 clusters, and each cluster consisted of 5-11 parallel paths. Each path consisted of a traversable edge with cost $c = 10$ from the cluster start to an intermediate vertex, an uncertain edge to a second intermediate vertex, and a traversable, $c = 1$ path from the second intermediate vertex to the goal vertex. The uncertain edge costs c and traversability probabilities ρ were randomly drawn from the intervals $c \sim [50, 100]$ and $\rho \sim [0.0, 0.5]$. We generated 16,000 total graphs, and split the dataset into 15,000 training problems and 1,000 test problems. An example problem is shown in Fig. 6.2-a.

6.4.2 Dataset Generation

Next, we describe the process of generating the dataset of problem and value function pairs in Eq. 6.2. For each of the 15,000 training problems, we ran the computationally expensive MC approximation-based multiagent planner [136] for a 1GV/1AV team to generate training examples. At each planning step in each trial, we recorded a data point, which included the current problem information (graph structure, agent states, and the team traversability belief) and the MC-based value function approximations, $\hat{V}^{b^{ze}}$ and $\hat{V}^{b}$, for each collaborative action considered at the planning step. Then, we selected 130,000 data points to use during neural network training (of 135,875 total generated), split the data into 100,000 train examples, 15,000 test examples, and 15,000 validation examples, and trained the value function approximation model in Eq. 6.4 with $\ell = 16$ using the loss function in Eq. 6.5 with $\beta = 100$.

6.4.3 Clusters Environment Results

First, we evaluated the accuracy of the learned value functions as compared to the value functions approximated using MC simulation. In Fig. 6.2-b, we directly compare learned and MC-based value functions, where the black dashed line indicates equal value. Overall,

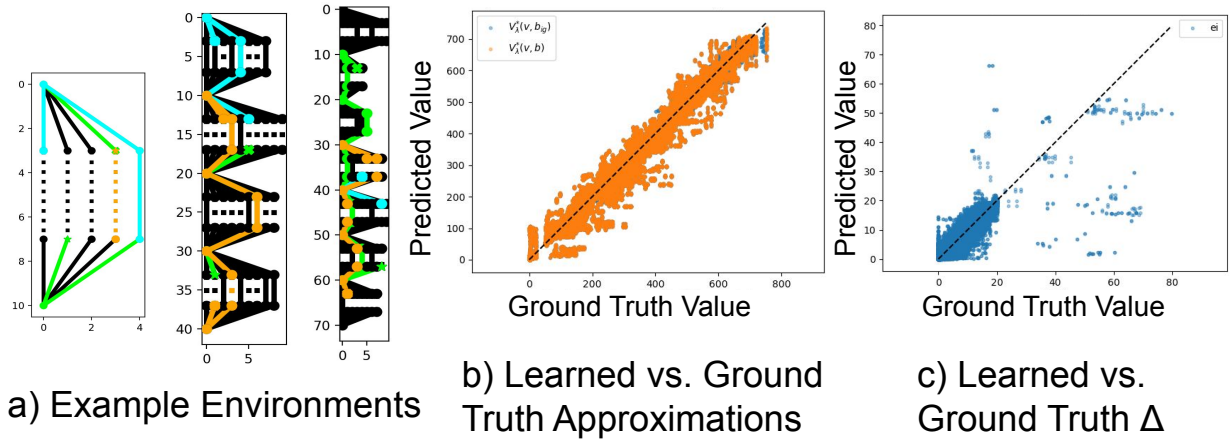


Figure 6.2: **Learning results in the cluster dataset.** a) Example problems in the cluster dataset. (b-c) Comparison of the *Monte Carlo* and *Learned* approximations on the test dataset. We plot both (b) the approximation values and (c) the difference between the values, or Δ. Values are similar, indicating that our learned network is capable of predicting value functions.

the learned value functions are similar to the MC-based value functions, and the average difference between the learned and predicted values in the test set was 1.73 units. The absolute accuracy of the value function is important because it enables the planner to directly compare value functions for different agents in the team; note that because the planner optimizes a makespan objective, the macro-action that most minimizes an individual agent's cost many not be the macro-action that most improves team planning performance. In Fig. 6.2-c, we compare the difference between the two value functions, Δ, for the learned and MC-based approximations, and demonstrate some correlation between the values. This metric is important for high-quality planning because it enables the planner to determine the relative importance of sensing different edges for an agent.

Next, we tested our learned value functions in the loop during online multiagent planning for a 1-GV/2-AV team navigating in the simulated toy environment. We compared our collaborative approach, *Learned*, to two baselines – a non-collaborative planner *Non-Collaborative* where agents planned independently but sensed and shared the traversability of traversed edges in a centralized team belief, and the collaborative MC-based planner used during training data generation [136], *Monte Carlo*. Both of the collaborative approaches

	Oracle	Non-Collaborative	Monte Carlo	Learned	Greedy Monte Carlo	Greedy Learned
Avg % Regret	0.0	11.56	6.02	5.08	6.05	5.08
Average Makespan, MA (units)	117.08	135.5	127.49	125.24	127.55	125.24
SEM Cost, MA (units)	3.14	3.82	3.54	3.42	3.55	3.42
Average Time (s)	0.0	1.75	5088.01	3224.49	260.37	49.21
SEM Time (s)	0.0	0.07	551.67	381.92	14.03	2.8
Num examples	0.0	1000.0	1000.0	1000.0	1000.0	1000.0

Table 6.1: **Planning results in the cluster domain.** Comparison of planning results using non-collaborative, Monte Carlo based, and learning based planners in the cluster domain.

used the approximated value functions to greedily select which edges to sense with $\gamma = 0.5$, but conservatively replanned to decide if any agent should wait for additional information. We quantitatively assess the *Non-Collaborative*, *Monte Carlo* and *Learned* approaches in Table 6.1. The *Monte Carlo* and *Learned* approaches found 6.02% and 5.08% lower cost plans than the *Non-Collaborative* approach on average, although the performance improvements caused significant increases in computation time (2907x and 1842x, respectively). However, the *Learned* approach was more efficient than the *Monte Carlo* approach, finding plans 36.6% more quickly than *Monte Carlo* on average; this is a significant improvement in a small environment that does not stress the scale of the environment or team. Overall, the *Learned* approach found low cost, uncertainty-aware collaborative plans while being more efficient than the *Monte Carlo* approach.

Additionally, we compared the Monte Carlo and learned approaches when greedily selecting both sensing and waiting actions, *Greedy Monte Carlo* and *Greedy Learned*, respectively; results are shown in Table 6.1. The *Greedy Monte Carlo* and *Greedy Learned* approaches found 6.05% and 5.08% lower cost plans than the *Non-Collaborative* approach on average, demonstrating that even greedy collaborative planning results in lower cost plans than non-collaborative planning. However, the greedy planners were 94.9% and 98.5% faster than their non-greedy counterparts while finding similar cost plans. The *Greedy Learned* planner was the most efficient of the collaborative planners, with an average planning time of only

49.21 seconds. Overall, the greedy approaches found low cost, uncertainty-aware collaborative plans in a fraction of the planning time as compared to the non-greedy collaborative planning approaches.

6.4.4 Islands Experiment Setup

Next, we demonstrated our approach for a 1-GV/2-AV team navigating in the simulated islands environment presented in [136], which consisted of a CTP graph with 90 vertices and 166 edges (not including self-loops). Graph edges were classified into to one of four classes: highway and highway bridge edges, and local road and local bridge edges. All edges had Euclidean costs, highway edge traversability probabilities were drawn from the distribution $\rho \sim (0.0, 0.001]$, and local edge probabilities were 0.5. We generated 16,000 total graphs, and split the dataset into 15,000 train problems and 1,000 test problems. An example problem is shown in Fig. 6.3-a. Then, we ran the expensive MC approximation planner on the train graphs to generate the dataset in Eq. 6.2, selected 90,000 data points to use during neural network training (from 96,818 generated total), split the data into 72,000 train examples, 9,000 test examples, and 9,000 validation examples, and trained the value function approximation model in Eq. 6.4 with $\ell = 20$ using the loss function in Eq. 6.5 with $\beta = 100$. Note that due to computational constraints, not all training problems ran to completion, and we selected the training data points from completed and partially completed trials.

6.4.5 Islands Environment Results

First, we evaluated the accuracy of the learned value functions as compared to the MC value functions. In Fig. 6.3-b, we directly compare learned and MC-based value functions, where the black dashed line indicates equal value. Some of the learned value functions are similar to the MC-based value functions, but there are significant outliers that must be studied further. In Fig. 6.3-c, we compare the difference between the two value functions, Δ, for the learned and MC-based approximations; the values are significantly different, and require

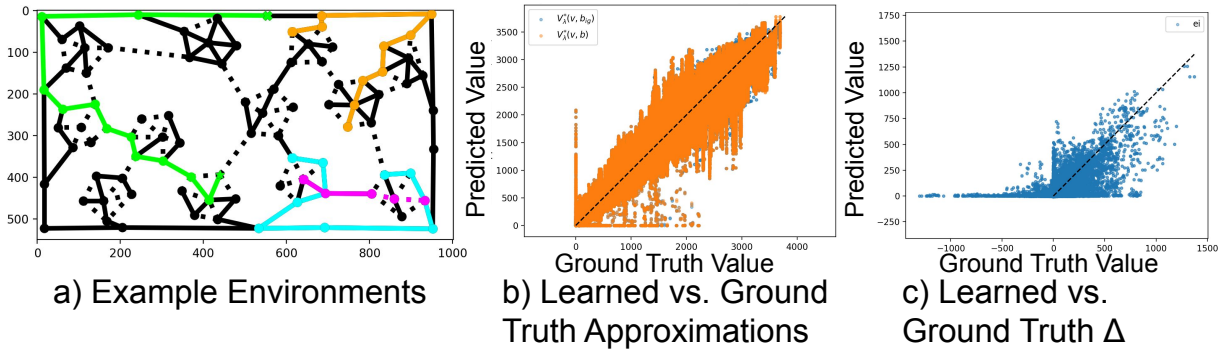


Figure 6.3: **Learning results in the islands dataset.** a) Example problems in the islands dataset. (b-c) Comparison of the *Monte Carlo* and *Learned* approximations on the test dataset. We plot both (b) the approximation values and (c) the difference between the values, or Δ. While some learned approximations are accurate, others are significantly different than the ground truth approximations and require further study.

	Oracle	Non-Collaborative	Monte Carlo	Greedy Monte Carlo	Learned	Greedy Learned
Avg Percent Regret	0.0	25.19	16.47	16.47	25.18	25.18
Average Makespan, MA (units)	955.53	1214.55	1134.12	1134.14	1197.05	1197.05
SEM Cost, MA (units)	15.53	28.09	20.83	20.83	21.83	21.83
Average Time (s)	0.0	3.16	11903.5	6910.8	3147.32	166.99
SEM Time (s)	0.0	0.08	336.45	163.46	138.72	4.21
Num examples	0.0	1000.0	1000.0	1000.0	1000.0	1000.0

Table 6.2: **Planning results in the islands domain.** Comparison of planning results using non-collaborative, Monte Carlo based, and learning based planners in the islands domain.

additional analysis.

We also evaluated the performance of the multiagent planners with $\gamma = 0.5$ for all collaborative planners. While the *Monte Carlo* and *Greedy Monte Carlo* planners resulted in significantly lower plan costs than the *Non-Collaborative* planner, the *Learned* and *Greedy Learned* planners did not improve planning performance as compared to the non-collaborative baseline. However, in the large environment, the learned planners were significantly more time efficient than the MC-based planners (73% and 97.6% more efficient, respectively). This supports our hypothesis that the learned value functions could significantly reduce collaborative multiagent planning times, if made more accurate.

6.5 Conclusion

In this work, we developed a method for learning single-agent value functions approximations on stochastic graphs that contained long length scale, uncertainty-aware global information, and applied the learned approximations to uncertainty-aware collaborative multiagent planning. Our approach used expensive Monte Carlo approximations, calculated offline, to train an efficient online value function approximator. We demonstrated our approach in a toy graph domain, and showed a 36.6% improvement in planning time as compared to a state-of-the-art collaborative planner, while finding plans of similar quality. However, additional work is required to scale the learned approximations to larger domains. In future work, we would like to apply the approach to larger multiagent teams, and to deploy the approach in a real-world environment.

Chapter 7

Conclusion

In this thesis, we developed models and planners that used global information to improve the efficiency of single-agent and multiagent navigation in large and uncertain environments. Our modeling methods focused on representing global information for navigation to reduce computation time and increase plan quality, on representing global environmental uncertainty to enable risk-aware planning, and on using environment and team structure to generate high-quality problem abstractions and reduce model complexity. Our planning methods focused on developing planners that were able to generate high-quality collaborative team plans by quickly identifying and resolving task-relevant environmental uncertainty during the planning process. In each method, we took advantage of problem structure and global information to improve navigation quality, and to reduce the computation required to generate high-quality plans.

In Chapter 3, we presented a novel hierarchical model for multi-query planning at long length scales. We developed a simple state- and action-based abstraction of the planning environment, and then used prior planning experience in the environment to calculate the *navigation properties* of the abstraction. Specifically, we calculated the navigation properties of prior primitive plans when the agent was operating in regions that corresponded to our abstraction, treated the properties as *samples* drawn from uncertain distributions over

abstract planning properties in the abstract representation, and used recursive estimation to update the abstract representation over the course of multi-query planning trials. During planning, we used the model in a hierarchical, uncertainty-aware planner, and demonstrated that our approach reduced nodes expanded and wallclock planning time as compared to non-hierarchical approaches, while finding similar quality plans. We also demonstrated the ability to use a risk parameter in the planner to trade off between planner between computation time and plan quality.

In Chapter 4, we presented a novel collaborative multiagent planner for team navigation on graphs with uncertain edge traversabilities, modeled as Canadian Traveller's Problem graphs. We developed a set of collaboration-aware multiagent macro-actions that reduced the planning horizon at which collaborative multiagent team actions occurred, therefore reducing the computation required to reason about the collaborative actions. Additionally, we developed single-agent value function approximations when agents had or did not have access to information gained from potential future sensing actions, and used the approximations to quickly identify collaborative actions that were likely to improve team planning performance. We demonstrated that our approach generated lower makespan plans than planners that did not explicitly collaborate, and generated plans more quickly than planners that did not use approximations to reduce the size of the collaborative policy space for planning.

In Chapter 5, we presented a system for deploying the collaborative multiagent planner in a real-world outdoor environment. While the abstract CTP-graph-based planner was efficient and enabled planning for collaboration, it generated abstract team plans that were not directly executable in a real-world environment. We developed a three-level hierarchical system to deploy the planner; the system included an abstract CTP graph planner, a macro-action planner that sequenced and monitored primitive actions, and primitive action planners that generated directly executable plans for the robot team. Our system maintained plan consistency throughout the planning hierarchy, and used robust planning modules to minimize the impact of minor low-level model misspecifications on abstract planning. We

deployed the system in two large, outdoor environments, and demonstrated collaborative team navigation.

In Chapter 6, we presented preliminary results of a novel technique for learning single-agent value functions from graph data to increase the efficiency of collaborative multiagent planning. While the approach in Chapter 4 enabled collaborative team behaviors, the computation required to calculate the single-agent value function approximations increased as the number of uncertain edges and uncertain agents in the environment increased. To enable our approach to be scaled to larger teams and environments, we proposed to learn the single-agent value function approximations from data. Specifically, we generated a dataset of value function approximations using an expensive Monte Carlo simulation-based planner offline, and then trained a graph neural network to predict the approximations during online planning. Our approach demonstrated the efficiency of a learned value function approximation approach, and demonstrated similar plan quality in some domains.

7.1 Future Work: Hierarchical Model Estimation for Multi-Query Planning

In Chapter 3, we developed a discretization-based hierarchical planning model, learned the properties of the model online, and applied the model for risk-aware planning.

While the approach enabled efficient, high-quality planning over the course of a multi-query planning trial, the abstract navigation properties of the model were initially set to an uninformative prior. This resulted in myopic planning during early stages of planning, as the abstract navigation properties did not accurately model the true costs of navigating in the environment. While the uncertainty-aware model and planner were able to identify the model uncertainty and use additional computation to find high-quality plans in the face of uncertainty, this resulted in significant additional computational cost during early planning. To mitigate this effect, it may be possible to use other types of global information, such

as overhead imagery, to generate an informed prior for the high-level model. For example, it may be possible to learn a prior over navigation properties in an abstract representation based on prior planning experience in similar environments.

Additionally, the approach required the environment to be discretized at a hand-defined resolution. Environment discretization can be computationally expensive in large environments, as the size of a discretized abstraction scales with the size of the environment. Additionally, selecting a good resolution for discretization for navigation can be challenging, as important features for navigation in an environment exist at various scales (e.g., roads, rivers, fields, forests, trees, bridges). Single-resolution discretizations must trade off between representing small navigation cues in detail and reducing model complexity to maintain computational tractability. To mitigate these issues, it may be possible to generate a variable-resolution abstraction, for example, by compressing the abstraction using information-aware compression techniques; however, a high-quality compression method would need to be uncertainty-aware, to enable the model to avoid over-compression when an agent does not have sufficient information about an abstract state. Additionally, it may be possible to apply recursive estimation for online updating of abstract navigation properties in other hierarchical planning representations that are not discretization-based. However, one challenge of this approach is generating a mapping between real-world planning experiences and the abstract model.

7.2 Future Work: Collaborative Multiagent Planning under Uncertainty

In Chapter 4, we developed multiagent macro-actions and macro-action based value function approximations to enable collaborative multiagent planning on stochastic graphs with uncertain edge traversabilities.

While our approach enabled efficient team collaboration on stochastic graphs, in practice,

the stochastic graphs were hand-generated. In Veys et al. [149], we developed a method for automatically generating high-quality stochastic graphs from polygonal environment representations, where each polygon was assigned a traversability probability. While the approach is a step towards automatically generating route graphs for planning under uncertainty, there are still a number of open questions in the space. For example, we are interested in automatically generating the polygonal representation from overhead imagery and other environment databases. Additionally, we are interested in generating efficient information-based approximations for planning with additional stochastic agents, and for teams with delayed and otherwise imperfect communication. It may also be interesting to explore multiagent collaboration on CTP-like graphs with uncertain edge costs. Finally, we are interested in extending the approach to agents with more complex capabilities that can change the state of the environment for a teammate. For example, a mobile manipulator may be capable of opening a door for a teammate, making a previously untraversable route traversable, or an inspection robot may be able to inspect and, if necessary, disarm a suspicious object on a low-cost path to the goal, enabling a teammate to navigate along the path.

7.3 Future Work: Real-World Deployments of Collaborative Multiagent Planning under Uncertainty

In Chapter 5, we developed a hierarchical planning system to enable the deployment of the collaborative multiagent planner in a real-world environment. In future work, we would like to test the system in deployments with larger teams and environments with more uncertain edges; this will likely require additional techniques to reduce the computational complexity of the existing planner. Additionally, we are interested in using the uncertainty-aware collaborative multiagent planner as the first phase of a larger team operation. For example, maps built and traversability observations detected during CTP-guided navigation could be used as preliminary models for executing more complex tasks in the environment, such as

suspicious region coverage, suspicious object identification and removal, and more. Finally, we are interested in techniques for identifying and correcting model mismatch in the abstract CTP graph representation during online planning. For example, it may be desirable to adjust the abstract representation of a specific vertex slightly to avoid collision with an unmapped obstacle.

7.4 Future Work: Learned Value Function Approximations on Uncertain Graphs

In Chapter 6, we developed a preliminary method for learning single-agent value functions for navigation under uncertainty on CTP graphs. In future work, we would like to reduce the error in the value function predictions, especially in larger graphs. Additionally, we would like to improve the generalization of the approach to unseen graphs. Finally, we are interested in exploring alternative learning methods for the problem, such as reinforcement learning, to improve system performance.

References

[1] C. Aeronautiques, A. Howe, C. Knoblock, I. D. McDermott, A. Ram, M. Veloso, D. Weld, D. W. Sri, A. Barrett, D. Christianson, et al. PDDL: The planning domain definition language. *Technical report*, 1998.

[2] V. Aksakalli, O. F. Sahin, and I. Ari. An AO* based exact algorithm for the Canadian traveler problem. *INFORMS Journal on Computing*, 28(1):96–111, 2016.

[3] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo. OBPRM: An obstacle-based PRM for 3D workspaces. In *International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, pages 155–168, 1998.

[4] J. A. Arjona-Medina, M. Gillhofer, M. Widrich, T. Unterthiner, J. Brandstetter, and S. Hochreiter. Rudder: Return decomposition for delayed rewards. *Advances in Neural Information Processing Systems*, 32, 2019.

[5] A. Asgharivaskasi and N. Atanasov. Semantic octree mapping and Shannon mutual information computation for robot exploration. *IEEE Transactions on Robotics*, 39 (3):1910–1928, 2023.

[6] N. Atanasov, J. Le Ny, K. Daniilidis, and G. J. Pappas. Decentralized active information acquisition: Theory and application to multi-robot SLAM. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4775–4782. IEEE, 2015.

[7] F. Bacchus and Q. Yang. The downward refinement property. In *International Joint Conference on Artificial Intelligence*, pages 286–293, 1991.

[8] E. Bakolas and P. Tsiotras. Multiresolution path planning via sector decompositions compatible to on-board sensor data. In *AIAA Guidance, Navigation and Control Conference and Exhibit*, page 7238, 2008.

[9] P. Battaglia, R. Pascanu, M. Lai, D. Jimenez Rezende, and K. Kavukcuoglu. Interaction networks for learning about objects, relations and physics. *Advances in Neural Information Processing Systems*, 29, 2016.

[10] P. Battaglia, J. B. C. Hamrick, V. Bapst, A. Sanchez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. E. Dahl, A. Vaswani, K. Allen, C. Nash, V. J. Langston, C. Dyer,

N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu. Relational inductive biases, deep learning, and graph networks. *arXiv*, 2018.

[11] S. Behnke. Local multiresolution path planning. In *Robot Soccer World Cup*, pages 332–343. Springer, 2003.

[12] D. Berenson, P. Abbeel, and K. Goldberg. A robot path planning framework that learns from experience. In *2012 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3671–3678. IEEE, 2012.

[13] G. Best and G. A. Hollinger. Decentralised self-organising maps for multi-robot information gathering. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4790–4797. IEEE, 2020.

[14] G. Best, O. M. Cliff, T. Patten, R. R. Mettu, and R. Fitch. Dec-MCTS: Decentralized planning for multi-robot active perception. *The International Journal of Robotics Research*, 38(2-3):316–337, 2019.

[15] Z. Bnaya, A. Felner, and S. E. Shimony. Canadian traveler problem with remote sensing. In *International Joint Conference on Artificial Intelligence*, pages 437–442, 2009.

[16] Z. Bnaya, A. Felner, D. Fried, O. Maksin, and S. Shimony. Repeated-task Canadian traveler problem. In *International Symposium on Combinatorial Search*, pages 24–30, 2011.

[17] Z. Bnaya, A. Felner, D. Fried, O. Maksin, and S. E. Shimony. Repeated-task Canadian traveler problem. *AI Communications*, 28(3):453–477, 2015.

[18] R. Bohlin. Path planning in practice; lazy evaluation on a multi-resolution grid. In *2001 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 49–54 vol.1. IEEE, 2001.

[19] B. Bonet. Deterministic POMDPs revisited. In *Conference on Uncertainty in Artificial Intelligence*, page 59–66, 2009.

[20] A. Botea, M. Müller, and J. Schaeffer. Near optimal hierarchical path-finding. *Journal of Game Development*, 1(1):1–30, 2004.

[21] C. Bradley, A. Pacheck, G. J. Stein, S. Castro, H. Kress-Gazit, and N. Roy. Learning and planning for temporally extended tasks in unknown environments. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4830–4836. IEEE, 2021.

[22] X. Bresson and T. Laurent. Residual gated graph convnets. In *International Conference on Learning Representations (ICLR) Workshop Track*, 2018.

[23] J. Canny. The complexity of robot motion planning. *MIT Press*, 1988.

[24] M. B. Chang, T. Ullman, A. Torralba, and J. B. Tenenbaum. A compositional object-based approach to learning physical dynamics. *International Conference on Learning Representations*, 2017.

[25] B. Charrow, V. Kumar, and N. Michael. Approximate representations for multi-robot control policies that maximize mutual information. *Autonomous Robots*, 37:383–400, 2014.

[26] R. Chitnis. Learning state and action abstractions for effective and efficient planning. *Ph.D thesis, Massachusetts Institute of Technology*, 2022.

[27] R. Chitnis, T. Silver, B. Kim, L. Kaelbling, and T. Lozano-Perez. CAMPs: Learning context-specific abstractions for efficient planning in factored MDPs. In *Conference on Robot Learning*, pages 64–79, 2021.

[28] R. Chitnis, T. Silver, J. B. Tenenbaum, T. Lozano-Pérez, and L. P. Kaelbling. Learning neuro-symbolic relational transition models for bilevel planning. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4166–4173. IEEE, 2022.

[29] J. J. Chung, A. J. Smith, R. Skeele, and G. A. Hollinger. Risk-aware graph search with dynamic edge cost discovery. *The International Journal of Robotics Research*, 38 (2-3):182–195, 2019.

[30] T. H. Chung, G. A. Hollinger, and V. Isler. Search and pursuit-evasion in mobile robotics: A survey. *Autonomous robots*, 31:299–316, 2011.

[31] R. V. Cowlagi and P. Tsiotras. Beyond quadtrees: Cell decompositions for path planning using wavelet transforms. In *IEEE Conference on Decision and Control*, pages 1392–1397. IEEE, 2007.

[32] M. Crosby, A. Jonsson, and M. Rovatsos. A single-agent approach to multiagent planning. In *European Conference on Artificial Intelligence*, pages 237–242, 2014.

[33] D. Dey, A. Kolobov, R. Caruana, E. Kamar, E. Horvitz, and A. Kapoor. Gauss meets Canadian traveler: Shortest-path problems with correlated natural dynamics. In *International Conference on Autonomous Agents and Multi-agent Systems*, pages 1101–1108, 2014.

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

[35] Y. Ding, X. Zhang, X. Zhan, and S. Zhang. Task-motion planning for safe and efficient urban driving. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2119–2125. IEEE, 2020.

[36] V. P. Dwivedi, C. K. Joshi, A. T. Luu, T. Laurent, Y. Bengio, and X. Bresson. Benchmarking graph neural networks. *Journal of Machine Learning Research*, 24(43):1–48, 2023.

[37] S. Edelkamp, M. Lahijanian, D. Magazzeni, and E. Plaku. Integrating temporal reasoning and sampling-based motion planning for multigoal problems with dynamics and time windows. *IEEE Robotics and Automation Letters*, 3(4):3473–3480, 2018.

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

[39] M. Everett, J. Miller, and J. P. How. Planning beyond the sensing horizon using a learned context. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1064–1071. IEEE, 2019.

[40] P. Eyerich, T. Keller, and M. Helmert. High-quality policies for the Canadian traveler's problem. In *AAAI Conference on Artificial Intelligence*, volume 24, pages 51–58, 2010.

[41] D. Fried, S. E. Shimony, and A. Felner. Optimal policies for special cases of the Canadian traveler problem. *Preprint*, 2010.

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

[43] V. Fung, J. Zhang, E. Juarez, and B. G. Sumpter. Benchmarking graph neural networks for materials chemistry. *npj Computational Materials*, 7(1):84, 2021.

[44] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot. Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2997–3004. IEEE, 2014.

[45] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez. Integrated task and motion planning. *Annual Review of Control, Robotics, and Autonomous Systems*, 4:265–293, 2021.

[46] B. P. Gerkey and M. J. Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, 23(9):939–954, 2004.

[47] D. Ghosh, G. Nandakumar, K. Narayanan, V. Honkote, and S. Sharma. Kinematic constraints based bi-directional RRT (KB-RRT) with parameterized trajectories for robot path planning in cluttered environment. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8627–8633. IEEE, 2019.

[48] T. Guan, Z. He, R. Song, D. Manocha, and L. Zhang. TNS: Terrain traversability mapping and navigation system for autonomous excavators. In *Robotics: Science and Systems*, 2022.

[49] H. Guo, F. Wu, Y. Qin, R. Li, K. Li, and K. Li. Recent trends in task and motion planning for robotics: A survey. *ACM Computing Surveys*, 55(13s):1–36, 2023.

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

[51] R. He, E. Brunskill, and N. Roy. Efficient planning under uncertainty with macro-actions. *Journal of Artificial Intelligence Research*, 40:523–570, 2011.

[52] B. Hedegaard, E. Fahnestock, J. Arkin, A. Menon, and T. M. Howard. Discrete optimization of adaptive state lattices for iterative motion planning on unmanned ground vehicles. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5764–5771. IEEE, 2021.

[53] G. A. Hollinger, S. Yerramalli, S. Singh, U. Mitra, and G. S. Sukhatme. Distributed coordination and data fusion for underwater search. In *2011 IEEE International Conference on Robotics and Automation*, pages 349–355. IEEE, 2011.

[54] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013.

[55] T. M. Howard. Adaptive model-predictive motion planning for navigation in complex environments. *Ph.D thesis, Carnegie Mellon University*, 2009.

[56] D. Hsu, Tingting Jiang, J. Reif, and Zheng Sun. The bridge test for sampling narrow passages with probabilistic roadmap planners. In *2003 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4420–4426 vol.3. IEEE, 2003.

[57] E. Hüllermeier and W. Waegeman. Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods. *Machine learning*, 110(3):457–506, 2021.

[58] B. Ichter, J. Harrison, and M. Pavone. Learning sampling distributions for robot motion planning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7087–7094. IEEE, 2018.

[59] B. Ichter, E. Schmerling, T.-W. E. Lee, and A. Faust. Learned critical probabilistic roadmaps for robotic motion planning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9535–9541. IEEE, 2020.

[60] C. K. Joshi, T. Laurent, and X. Bresson. An efficient graph convolutional network technique for the travelling salesman problem. *arXiv*, 2019.

[61] C. K. Joshi, Q. Cappart, L.-M. Rousseau, and T. Laurent. Learning the travelling salesperson problem requires rethinking generalization. *Constraints*, 27(1):70–98, 2022.

[62] S. S. Joshi and P. Tsiotras. Relevant region exploration on general cost-maps for sampling-based motion planning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6689–6695. IEEE, 2020.

[63] L. P. Kaelbling and T. Lozano-Pérez. Hierarchical task and motion planning in the now. In *2011 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1470–1477. IEEE, 2011.

[64] S. Kambhampati and L. Davis. Multiresolution path planning for mobile robots. *IEEE Journal on Robotics and Automation*, 2(3):135–145, 1986.

[65] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[66] L. E. Kavraki, P. Svestka, J. . Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[67] A. Khan, A. Ribeiro, V. Kumar, and A. G. Francis. Graph neural networks for motion planning. *arXiv*, 2020.

[68] T. Klamt and S. Behnke. Planning hybrid driving-stepping locomotion on multiple levels of abstraction. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1695–1702. IEEE, 2018.

[69] T. Klamt and S. Behnke. Towards learning abstract representations for locomotion planning in high-dimensional state spaces. In *2019 IEEE International Conference on Robotics and Automation (ICRA)*, pages 922–928. IEEE, 2019.

[70] L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo planning. In *European Conference on Machine Learning*, pages 282–293, 2006.

[71] G. Konidaris, L. P. Kaelbling, and T. Lozano-Perez. From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, 61:215–289, 2018.

[72] G. A. Korsah, A. Stentz, and M. B. Dias. A comprehensive taxonomy for multi-robot task allocation. *The International Journal of Robotics Research*, 32(12):1495–1512, 2013.

[73] J. Kottinger, S. Almagor, and M. Lahijanian. Conflict-based search for multi-robot motion planning with kinodynamic constraints. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 13494–13499. IEEE, 2022.

[74] H. Kurniawati. Partially observable markov decision processes and robotics. *Annual Review of Control, Robotics, and Autonomous Systems*, 5(1):253–277, 2022.

[75] M. S. Kurtz, S. Prentice, Y. Veys, L. Quang, C. Nieto-Granda, M. Novitzky, E. Stump, and N. Roy. Real-world deployment of a hierarchical uncertainty-aware collaborative multiagent planning system. In *2024 IEEE International Conference on Robotics and Automation Workshop on Field Robotics*, 2024.

[76] D. T. Larsson, D. Maity, and P. Tsiotras. Q-tree search: An information-theoretic approach toward hierarchical abstractions for agents with computational limitations. *IEEE Transactions on Robotics*, 36(6):1669–1685, 2020.

[77] D. T. Larsson, D. Maity, and P. Tsiotras. Information-theoretic abstractions for planning in agents with computational constraints. *IEEE Robotics and Automation Letters*, 6(4):7651–7658, 2021.

[78] D. T. Larsson, A. Asgharivaskasi, J. Lim, N. Atanasov, and P. Tsiotras. Information-theoretic abstraction of semantic octree models for integrated perception and planning. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6937–6943, 2023.

[79] M. Lauri, E. Heinänen, and S. Frintrop. Multi-robot active information gathering with periodic communication. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 851–856. IEEE, 2017.

[80] M. Lauri, J. Pajarinen, and J. Peters. Information gathering in decentralized POMDPs by policy graph improvement. *International Conference on Autonomous Agents and Multi-Agent Systems*, 2019.

[81] M. Lauri, J. Pajarinen, and J. Peters. Multi-agent active information gathering in discrete and continuous-state decentralized POMDPs by policy graph improvement. *Autonomous Agents and Multi-Agent Systems*, 34(2):42, 2020.

[82] S. M. LaValle. *Planning algorithms*. Cambridge University Press, 2006.

[83] K. M. B. Lee, F. Kong, R. Cannizzaro, J. L. Palmer, D. Johnson, C. Yoo, and R. Fitch. An upper confidence bound for simultaneous exploration and exploitation in heterogeneous multi-robot systems. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8685–8691. IEEE, 2021.

[84] K. M. B. Lee, F. H. Kong, R. Cannizzaro, J. L. Palmer, D. Johnson, C. Yoo, and R. Fitch. Decentralised intelligence, surveillance, and reconnaissance in unknown environments with heterogeneous multi-robot systems. *2021 IEEE International Conference on Robotics and Automation Workshop on Robot Swarms in the Real World: From Design to Deployment*, 2021.

[85] Y. Lee, P. Cai, and D. Hsu. MAGIC: Learning macro-actions for online POMDP planning. In *Robotics: Science and Systems*, 2021.

[86] J. Li. Efficient and effective techniques for large-scale multi-agent path finding. *Ph.D thesis, University of Southern California*, 2022.

[87] Q. Li, F. Gama, A. Ribeiro, and A. Prorok. Graph neural networks for decentralized path planning. In *International Conference on Autonomous Agents and Multi-Agent Systems*, pages 1901–1903, 2020.

[88] Q. Li, F. Gama, A. Ribeiro, and A. Prorok. Graph neural networks for decentralized multi-robot path planning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11785–11792. IEEE, 2020.

[89] X. Li, L. Sun, M. Ling, and Y. Peng. A survey of graph neural network based recommendation in social networks. *Neurocomputing*, 549:126441, 2023.

[90] Y. Li, Z. Littlefield, and K. E. Bekris. Asymptotically optimal sampling-based kinodynamic planning. *The International Journal of Robotics Research*, 35(5):528–564, 2016.

[91] Z. Lim, L. Sun, and D. Hsu. Monte Carlo value iteration with macro-actions. *Advances in Neural Information Processing Systems*, 24, 2011.

[92] K. Liu, M. Stadler, and N. Roy. Learned sampling distributions for efficient planning in hybrid geometric and object-level representations. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9555–9562. IEEE, 2020.

[93] S.-Y. Lo, S. Zhang, and P. Stone. PETLON: Planning efficiently for task-level-optimal navigation. In *International Conference on Autonomous Agents and Multi-Agent Systems*, pages 220–228, 2018.

[94] A. Majumdar and R. Tedrake. Funnel libraries for real-time robust feedback motion planning. *The International Journal of Robotics Research*, 36(8):947–982, 2017.

[95] T. Marcucci, M. Petersen, D. von Wrangel, and R. Tedrake. Motion planning around obstacles with convex optimization. *Science Robotics*, 8(84), 2023.

[96] B. Marthi, S. Russell, and J. A. Wolfe. Angelic semantics for high-level actions. In *International Conference on Automated Planning and Scheduling*, pages 232–239, 2007.

[97] B. Marthi, S. J. Russell, and J. A. Wolfe. Angelic hierarchical planning: Optimal and online algorithms. In *International Conference on Automated Planning and Scheduling*, pages 222–231, 2008.

[98] B. Marthi, S. Russell, J. Wolfe, et al. Angelic hierarchical planning: Optimal and online algorithms (revised). *Electrical Engineering*, 2009.

[99] A. Messing, G. Neville, S. Chernova, S. Hutchinson, and H. Ravichandar. GRSTAPS: Graphically recursive simultaneous task allocation, planning, and scheduling. *The International Journal of Robotics Research*, 41(2):232–256, 2022.

[100] A. Messing, J. Banfi, M. Stadler, E. Stump, H. Ravichandar, N. Roy, and S. Hutchinson. A sampling-based approach for heterogeneous coalition scheduling with temporal uncertainty. In *Robotics: Science and Systems*, 2023.

[101] D. Molina, K. Kumar, and S. Srivastava. Learn and link: Learning critical regions for efficient planning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10605–10611. IEEE, 2020.

[102] L. Murphy and P. Newman. Planning most-likely paths from overhead imagery. In *2010 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3059–3064. IEEE, 2010.

[103] L. Murphy and P. Newman. Risky planning on probabilistic costmaps for path planning in outdoor environments. *IEEE Transactions on Robotics*, 29(2):445–457, 2013.

[104] L. Murphy, P. Corke, and P. Newman. Choosing landmarks for risky planning. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3868–3873. IEEE, 2011.

[105] L. Murphy, S. Martin, and P. Corke. Creating and using probabilistic costmaps from vehicle experience. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4689–4694. IEEE, 2012.

[106] M. E. Napoli, H. Biggie, and T. M. Howard. On the performance of selective adaptation in state lattices for mobile robot motion planning in cluttered environments. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4436–4443. IEEE, 2017.

[107] M. E. Napoli, H. Biggie, and T. M. Howard. Learning models for predictive adaptation in state lattices. In *Field and Service Robotics: Results of the 11th International Conference*, pages 285–300. Springer, 2018.

[108] E. Nelson, M. Corah, and N. Michael. Environment model adaptation for mobile robot exploration. *Autonomous Robots*, 42:257–272, 2018.

[109] G. Neville, A. Messing, H. Ravichandar, S. Hutchinson, and S. Chernova. An interleaved approach to trait-based task allocation and scheduling. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1507–1514. IEEE, 2021.

[110] G. Neville, S. Chernova, and H. Ravichandar. D-ITAGS: A dynamic interleaved approach to resilient task allocation, scheduling, and motion planning. *IEEE Robotics and Automation Letters*, 8(2):1037–1044, 2023.

[111] G. Neville, J. Liu, S. Chernova, and H. Ravichandar. Q-ITAGS: Quality-optimized spatio-temporal heterogeneous task allocation with a time budget. *arXiv*, 2024.

[112] E. Nikolova and D. R. Karger. Route planning under uncertainty: The Canadian traveller problem. In *AAAI Conference on Artificial Intelligence*, pages 969–974, 2008.

[113] S. C. Ong, S. W. Png, D. Hsu, and W. S. Lee. Planning under uncertainty for robotic tasks with mixed observability. *The International Journal of Robotics Research*, 29(8): 1053–1068, 2010.

[114] C. H. Papadimitriou and M. Yannakakis. Shortest paths without a map. *Theoretical Computer Science*, 84(1):127–150, 1991.

[115] J. Park, A. Messing, H. Ravichandar, and S. Hutchinson. Risk-tolerant task allocation and scheduling in heterogeneous multi-robot teams. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5372–5379. IEEE, 2023.

[116] J. Pearl and J. H. Kim. Studies in semi-admissible heuristics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-4(4):392–399, 1982.

[117] M. Phillips, B. J. Cohen, S. Chitta, and M. Likhachev. E-graphs: Bootstrapping planning with experience graphs. In *Robotics: Science and Systems*, volume 5, page 110, 2012.

[118] M. Pivtoraiko, R. A. Knepper, and A. Kelly. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, 26(3):308–333, 2009.

[119] I. Pohl. Heuristic search viewed as path finding in a graph. *Artificial intelligence*, 1 (3-4):193–204, 1970.

[120] A. Prorok. Robust assignment using redundant robots on transport networks with uncertain travel time. *IEEE Transactions on Automation Science and Engineering*, 17 (4):2025–2037, 2020.

[121] G. Rajendran, V. Uma, and B. O'Brien. Unified robot task and motion planning with extended planner using ROS simulator. *Journal of King Saud University-Computer and Information Sciences*, 34(9):7468–7481, 2022.

[122] J. H. Reif. Complexity of the mover's problem and generalizations. In *Symposium on Foundations of Computer Science*, pages 421–427. IEEE Computer Society, 1979.

[123] P. Reiser, M. Neubert, A. Eberhard, L. Torresi, C. Zhou, C. Shao, H. Metni, C. van Hoesel, H. Schopmans, T. Sommer, et al. Graph neural networks for materials science and chemistry. *Communications Materials*, 3(1):93, 2022.

[124] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 4th edition, 2021.

[125] Y. Satsangi, S. Whiteson, F. A. Oliehoek, and M. T. Spaan. Exploiting submodular value functions for scaling up active perception. *Autonomous Robots*, 42(2):209–233, 2018.

[126] D. Schleich, T. Klamt, and S. Behnke. Value iteration networks on multiple levels of abstraction. *Robotics: Science and Systems*, 2019.

[127] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel. Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research*, 33(9):1251–1270, 2014.

[128] D. Shah and S. Levine. ViKiNG: Vision-based kilometer-scale navigation with geographic hints. In *Robotics: Science and Systems*, 2022.

[129] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66, 2015.

[130] A. Shkolnik and R. Tedrake. Sample-based planning with volumes in configuration space. *arXiv*, 2011.

[131] D. Silver and J. Veness. Monte-Carlo planning in large POMDPs. *Advances in Neural Information Processing Systems*, 23, 2010.

[132] A. Somani, N. Ye, D. Hsu, and W. S. Lee. DESPOT: Online POMDP planning with regularization. *Advances in Neural Information Processing Systems*, 26, 2013.

[133] M. T. Spaan, T. S. Veiga, and P. U. Lima. Decision-theoretic planning under uncertainty with information rewards for active cooperative perception. *Autonomous Agents and Multi-Agent Systems*, 29:1157–1185, 2015.

[134] M. Stadler. Learned functions for perceptually informed robot navigation. *Master's thesis, Massachusetts Institute of Technology*, 2020.

[135] M. Stadler, K. Liu, and N. Roy. Online high-level model estimation for efficient hierarchical robot navigation. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5568–5575. IEEE, 2021.

[136] M. Stadler, J. Banfi, and N. Roy. Approximating the value of collaborative team actions for efficient multiagent navigation in uncertain graphs. In *International Conference on Automated Planning and Scheduling*, volume 33, pages 677–685, 2023.

[137] G. J. Stein, C. Bradley, and N. Roy. Learning over subgoals for efficient navigation of structured, unknown environments. In *Conference on Robot Learning*, pages 213–222, 2018.

[138] R. Stern, N. Sturtevant, A. Felner, S. Koenig, H. Ma, T. Walker, J. Li, D. Atzmon, L. Cohen, T. Kumar, et al. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *International Symposium on Combinatorial Search*, volume 10, pages 151–158, 2019.

[139] Z. N. Sunberg and M. J. Kochenderfer. Online algorithms for POMDPs with continuous state, action, and observation spaces. In *International Conference on Automated Planning and Scheduling*, 2018.

[140] A. Tamar, Y. Wu, G. Thomas, S. Levine, and P. Abbeel. Value iteration networks. *Advances in Neural Information Processing Systems*, 29, 2016.

[141] G. Theocharous and L. Kaelbling. Approximate planning in POMDPs with macro-actions. *Advances in Neural Information Processing Systems*, 16, 2003.

[142] A. Thomas, F. Mastrogiovanni, and M. Baglietto. MPTP: Motion-planning-aware task planning for navigation in belief space. *Robotics and Autonomous Systems*, 141, 2021.

[143] C. Thorpe and L. Matthies. Path relaxation: Path planning for a mobile robot. In *OCEANS 1984*, pages 576–581, 1984.

[144] Y. Tian, Y. Chang, F. Herrera Arias, C. Nieto-Granda, J. P. How, and L. Carlone. Kimera-Multi: Robust, distributed, dense metric-semantic SLAM for multi-robot systems. *IEEE Transactions on Robotics*, 38(4):2022–2038, 2022.

[145] A. J. B. Trevor, J. G. Rogers, and H. I. Christensen. OmniMapper: A modular multimodal mapping framework. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1983–1990. IEEE, 2014.

[146] P. Tsiotras and E. Bakolas. A hierarchical on-line path planning scheme using wavelets. In *2007 European Control Conference (ECC)*, pages 2806–2812, 2007.

[147] P. Tsiotras, D. Jung, and E. Bakolas. Multiresolution hierarchical path-planning for small uavs using wavelet decompositions. *Journal of Intelligent & Robotic Systems*, 66 (4):505–522, 2012.

[148] W. Vega-Brown and N. Roy. Admissible abstractions for near-optimal task and motion planning. In *International Joint Conference on Artificial Intelligence*, 2018.

[149] Y. Veys, M. S. Kurtz, and N. Roy. Generating sparse probabilistic graphs for efficient planning in uncertain environments. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024.

[150] J. Wang, W. Chi, C. Li, C. Wang, and M. Q.-H. Meng. Neural RRT*: Learning-based optimal path planning. *IEEE Transactions on Automation Science and Engineering*, 17(4):1748–1758, 2020.

[151] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou. Aggressive driving with model predictive path integral control. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1433–1440. IEEE, 2016.

[152] J. Wöhlke, F. Schmitt, and H. van Hoof. Hierarchies of planning and reinforcement learning for robot navigation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10682–10688. IEEE, 2021.

[153] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2020.

[154] K. Xu, J. Li, M. Zhang, S. S. Du, K. Kawarabayashi, and S. Jegelka. What can neural networks reason about? *International Conference on Learning Representations*, 2020.

[155] A. Yershova, L. Jaillet, T. Siméon, and S. M. LaValle. Dynamic-domain RRTs: Efficient exploration by controlling the sampling domain. In *2005 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3856–3861. IEEE, 2005.

[156] H. Yu, R. W. Beard, and J. Byrne. Vision-based local multi-resolution mapping and path planning for miniature air vehicles. In *2009 American Control Conference*, pages 5247–5252, 2009.

[157] W. Zhao, Q. Meng, and P. W. Chung. A heuristic distributed task allocation method for multivehicle multitask problems and its application to search and rescue scenario. *IEEE Transactions on Cybernetics*, 46(4):902–915, 2015.

[158] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa. CHOMP: Covariant hamiltonian optimization for motion planning. *The International Journal of Robotics Research*, 32(9-10):1164–1193, 2013.