Adaptive Motion Planning

Sanjiban Choudhury
February 14, 2018

CMU-RI-TR-18-04
The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Sebastian Scherer, CMU RI (Chair)
Siddhartha Srinivasa, UW CSE
Martial Hebert, CMU RI
Ashish Kapoor, Microsoft Research

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy.

Copyright © Sanjiban Choudhury

Abstract

Mobile robots are increasingly being deployed in the real world in response to a heightened demand for applications such as transportation, delivery and inspection. The motion planning systems for these robots are expected to have consistent performance across the wide range of scenarios that they encounter. While state-of-the-art planners, with provable worst-case guarantees, can be employed to solve these planning problems, their finite time performance varies across scenarios. This thesis proposes that the planning module for a robot must adapt its search strategy to the distribution of planning problems encountered to achieve real-time performance. We address three principal challenges of this problem.

Firstly, we show that even when the planning problem distribution is fixed, designing a non-adaptive planner can be challenging as the performance of planning strategies fluctuates with small changes in the environment. We characterize the existence of complementary strategies and propose to hedge our bets by executing a diverse ensemble of planners.

Secondly, when the distribution is varying, we require a meta-planner that can automatically select such an ensemble from a library of black-box planners. We show that greedily training a list of predictors to focus on failure cases leads to an effective meta-planner. For situations where we have no training data, we show that we can learn an ensemble on-the-fly by adopting algorithms from online paging theory.

Thirdly, in the interest of efficiency, we require a white-box planner that directly adapts its search strategy during a planning cycle. We propose an efficient procedure for training adaptive search heuristics in a data-driven imitation learning framework. We also draw a novel connection to Bayesian active learning, and propose algorithms to adaptively evaluate edges of a graph.

Our approach leads to the synthesis of a robust real-time planning module that allows a UAV to navigate seamlessly across environments and speed-regimes. We evaluate our framework on a spectrum of planning problems and show closed-loop results on 3 UAV platforms - a full-scale autonomous helicopter, a large scale hexarotor and a small quadrotor. While the thesis was motivated by mobile robots, we have shown that the individual algorithms are broadly applicable to other problem domains such as informative path planning and manipulation planning. We also establish novel connections between the disparate fields of motion planning and active learning, imitation learning and online paging which opens doors to several new research problems.

Acknowledgements

I would like first to thank my advisor, Sebastian, whose constant support, patience and guidance have helped me navigate through the fog of research. His fearless approach to robotics and encyclopaedic Linux knowledge continue to captivate and inspire me. Thank you for giving me the freedom to explore collaborations, to voice criticisms and to evolve as a researcher.

I am grateful to my thesis committee members for their invaluable feedback. Sidd, thank you for teaching me everything I know about motion planning - I look forward to our academic adventure. Martial, thank you for your insightful and fundamental questions that always sends me running back to the basics. Ashish, thank you for the hours of technical discussions and for taking a chance with me at MSR and on your plane. I would also like to thank Drew Bagnell, whose profound impact on my research and philosophies is evident in this thesis.

Robotics is a team effort, and I could not have asked for a better team than the people at AirLab. Thanks especially to folks with whom I have had the opportunity to work with - Andrew, Hugh, Geetesh, Luke, Kristen, Shichao, Brian and Azaraksh. Thanks to Vishal and Mohak whose research was critical to this thesis. Thanks to Ratnesh and Rogerio for the amazing discussions. I am forever indebted to Silvio Maeta on whom I have relied for every single robot result (he truly is Superman). Thanks to Daniel M. for the sporadic rants that led me to Wes Anderson, Dennet and Neruda. Thanks to Daniel A. for being the best mentor and friend - your relentless sense of humour is sorely missed. Thank you, Sankalp, O Captain, for being a constant confidante, competitor, critic and companion - indeed our fearful trip is done.

I am thankful to the people who have supported me over the years, especially the folks at Near Earth Autonomy. Thanks to Sanjiv for bringing me to CMU and research. Thanks to Marcel and Lyle for encouragement and support. Thanks especially to Kyle, Adam, Brad, AJ and Puru. I am indebted to Mark Hardesty for lessons on life and aerodynamics. At CMU, I would like to thank Sanae Minick, Nora Kazour and Suzanne Lyons Muth for all your help.

Thank you, Abhijeet, for being a partner in crime in all intellectual endeavours and for being a patient audience to my flip-flopping narratives. I am grateful to Dey for his mentorship, his infectious excitement and for granting me a second wind at MSR. Thanks to Jonathan Gammel for showing me how to write papers, to do experiments and remain optimistic. Thanks to Wen for teaching me so much about machine learning and pork buns. Thanks to Anirudh for being

the best intern and being at the origin of this work. Thank you, Arun and Venkat, for the invigorating discussions on learning and planning. Thank you, Oren, for sharing your wisdom and your clarity of thought. Thanks to Humphrey, Vishnu, Venkat R., Jiaji and others for sharing their research over the years - I have learnt a lot through our interactions. I was fortunate to have collaborated with many talented researchers - Shervin, Gireeja, Guilherme, Steve, Nathan - thank you for teaching me so much.

Thanks to my friends - Sezal, Zania, Ananya, Senthil - for putting up with my nonsense, sharing their lives and steadying the ship. Thank you, Ravi, for the awesome journeys in Seattle. Thanks to Debangshu, Debanjan and Arko for being a thread back to simpler times.

Finally, words cannot express my feelings for my family. Thanks, Mom and Dad, for your unconditional love, encouragement and sacrifice - you have made us feel unbreakable. This achievement is as much yours as is mine. Thank you brother, “of course”, for being the ‘black-box’ who just fixes everything, Jeeves to my Wooster, omniscient and omnipotent! You have been an invariant through all adventures in life - thanks for seeing me through this one. And to my fianceé Paloma, I cannot imagine a closer partner, co-pilot and friend. Thank you for this beautiful entangled existence, free of solipsism, where we juggle roles and seesaw responsibilities. I feel very fortunate to have crossed this finish line with you.

Contents

1 Introduction **1**
- 1.1 Motivational examples and applications . . . 1
- 1.2 Why does a motion planning system need to adapt? . . . 4
- 1.3 Problem characterization . . . 6
- 1.4 Outline of approach . . . 9
- 1.5 Summary of contributions . . . 10
- 1.6 Review of experimental platforms . . . 11

2 Background **13**
- 2.1 The planning problem . . . 13
- 2.2 A distribution over planning problems . . . 14
- 2.3 What dominates planning time? . . . 15
- 2.4 Taxonomy . . . 16
- 2.5 Related work . . . 18

I Foundations **25**

3 Planning Algorithms that Exploit Structure **27**
- 3.1 A gentle start . . . 28
- 3.2 The optimal path planning problem . . . 33
- 3.3 A framework for systematically assembling planning algorithms . . . 33
- 3.4 Example algorithm 1: Hybrid local global search (RABIT*) . . . 39
- 3.5 Example algorithm 2: 3D sparse visibility graph (SPARTAN-Lite) . . . 42
- 3.6 Example algorithm 3: Reachability informed random graph (RRT*-AR) . . . 44
- 3.7 Theoretical limits of planning resolution . . . 45
- 3.8 Discussion . . . 49

4 A Diverse Ensemble of Expert Planners **51**

4.1 The planner design challenge . . . 52
4.2 What is an expert planner? . . . 52
4.3 General purpose planners versus precision planners . . . 54
4.4 Difficulty in performance prediction of precision planners . . . 57
4.5 Ensemble of diverse expert planners . . . 58
4.6 Case Study: Autonomous helicopter . . . 60
4.7 Conclusion and discussion . . . 62

II Black-box Adaptive Planners 65

5 Adaptive Ensembles of Expert Planners 67
5.1 Introduction . . . 68
5.2 Background . . . 71
5.3 Problem formulation . . . 72
5.4 Constructing a library of expert planners . . . 73
5.5 Predicting a static ensemble . . . 75
5.6 Predicting a dynamic ensemble . . . 78
5.7 Experimental evaluation . . . 82
5.8 Closed-loop evaluation of dynamic ensemble on different UAV platforms . . . 89
5.9 Discussion and future work . . . 94

6 Online Exception Planners 97
6.1 Introduction . . . 98
6.2 Background . . . 99
6.3 Problem formulation . . . 100
6.4 Online paging: The Least Recently Used algorithm . . . 102
6.5 Experiments . . . 103
6.6 Discussion and future directions . . . 104

III White-box Adaptive Planners 107

7 Data-driven Planning via Imitation Learning 109
7.1 Introduction . . . 109
7.2 Background . . . 113
7.3 Problem formulation . . . 121
7.4 Imitation of clairvoyant oracles . . . 124
7.5 Approach . . . 127
7.6 Experiments on informative path planning . . . 135
7.7 Experiments on search based planning . . . 142
7.8 Discussion and future work . . . 146

8 Bayesian Active Edge Evaluation 151

8.1 Introduction . . . 151
8.2 Background . . . 155
8.3 Problem formulation . . . 156
8.4 The Decision Region Edge Cutting algorithm (DIRECT) . . . 158
8.5 The Bernoulli Subregion Edge Cutting algorithm (BISECT) . . . 160
8.6 Approach . . . 164
8.7 Experiments . . . 166
8.8 Discussion and future work . . . 172

IV Bridging Theory and Practice **175**

9 A Unified Planning Architecture for UAVs **177**
9.1 Introduction . . . 177
9.2 Problem formulation . . . 178
9.3 Trajectory planning architecture . . . 181
9.4 Global planning via route guide generation . . . 186
9.5 Local planning via an adaptive ensemble . . . 188
9.6 Guaranteeing safety via trajectory executive . . . 189
9.7 Flight test summary . . . 190

10 Conclusion **195**
10.1 Summary and contributions . . . 195
10.2 Future directions . . . 198
10.3 Concluding remarks . . . 204

Appendices **207**

A Planning Problem for UAVs **209**

B Dynamics Projection Filter **213**

C Library of Expert Planners for UAVs **215**

D Feature Extraction for Black-box Planners **221**

References **225**

List of Figures

1.1 A spectrum of environments in which an autonomous helicopter operates 2
1.2 Re-using planning software across UAV platforms 3
1.3 Real-time planning for unconventional dynamical systems 4
1.4 Hardness of motion planning for mobile robots 4
1.5 What is an adaptive planner? 5
1.6 Challenge 1: Design a non-adaptive motion planner 7
1.7 Challenge 2: Design a black-box adaptive motion planner 7
1.8 Challenge 3: Design a white-box adaptive motion planner 8
1.9 The 3 UAV platforms on which we evaluate 11

2.1 Expensive edge evaluation on a state lattice 15
2.2 A taxonomy for adaptive motion planning 16
2.3 White-box vs black-box motion planner paradigm 17
2.4 Static vs dynamic motion planner paradigm 17
2.5 Overview of the related work 18

3.1 The narrow passage problem 28
3.2 Expansive space characterization 29
3.3 Utility of intermediate points in planning 31
3.4 Difficult to sample homotopy classes 39
3.5 An illustration of the RABIT* algorithm 39
3.6 Evaluation of RABIT* on random $\mathbb{R}^2$ world 40
3.7 Evaluation of RABIT* on random $\mathbb{R}^8$ world 41
3.8 Outdoor flight of a micro aerial vehicle 42
3.9 Illustration of SPARTAN-Lite algorithm 42
3.10 Evaluation of SPARTAN-Lite in simulation 43
3.11 Evaluation of SPARTAN-Lite on a real robot 43
3.12 Engine failure in an autonomous helicopter 44
3.13 Illustration of the `Near` $(\mathbf{v})$ of RRT*-AR 45

3.14 Evaluation of RRT*-AR . . . 45
3.15 Robot dynamics affect speed and planning resolution . . . 46
3.16 Equivalence of state lattice with a directed asymmetric hexagonal lattice . . . 47
3.17 Equivalence of a hexagonal lattice with a Markov chain . . . 48

4.1 Overview of an expert planner . . . 53
4.2 Designing expert planners for a planning problem distribution . . . 55
4.3 Score profiles of general purpose planners vs precision planners . . . 56
4.4 Mertis and pitfalls of precision planners . . . 57
4.5 Difficulty in predicting the performance of a precision planner . . . 58
4.6 Ensemble of expert planners . . . 59
4.7 Greedy design of ensemble . . . 60
4.8 Different planning problem distributions for an autonomous helicopter . . . 60
4.9 Different planning problems encountered during operation in Quantico . . . 61
4.10 Unsolved problems by `CHOMP`^{1} which are solved by `CHOMP`^{2} . . . 61
4.11 `CHOMP`^{1} enables avoidance of a mountain enroute . . . 62
4.12 `RRT*Tunnel`^{1} is able to find solutions where `CHOMP`^{1} fails . . . 63

5.1 The black-box adaptive planning framework . . . 68
5.2 Different planning problems in a cargo delivery application . . . 69
5.3 Training predictors to greedily focus on unsolved problems . . . 70
5.4 Framework for dynamic ensemble predictor . . . 79
5.5 Evaluation of dynamic ensemble on datasets . . . 83
5.6 Evaluation of dynamic ensemble on autonomous helicopter . . . 85
5.7 Evaluation of lazy greedy static ensemble on a representative dataset . . . 87
5.8 Evaluation of lazy greedy static ensemble on autonomous helicopter dataset . . . 88
5.9 Long term autonomy on a large hexarotor using an adaptive ensemble . . . 90
5.10 Training details of an adaptive ensemble on a large hexarotor . . . 91
5.11 Analysis of adaptation of the planner during the flight test . . . 92
5.12 Closed loop evaluation on a small quadrotor . . . 93
5.13 Switching behaviour in ensemble selection . . . 94

6.1 Illustration of online exception planners . . . 98
6.2 Receding horizon planning on two kinds of environments . . . 103
6.3 Comparison of different algorithms on two different datasets . . . 105

7.1 Sequential decision making in informative path planning and search . . . 110
7.2 The informative path planning problem . . . 114
7.3 The search based planning problem . . . 118
7.4 Overview of the two approaches for training policies . . . 127
7.5 An overview of QvalAgg in IPP . . . 130
7.6 An overview of SaIL in search based planning . . . 133
7.7 Results on a spectrum of 2D and 3D exploration problems . . . 135

7.8 Case study of Problem HIDDEN-UNC . . . 138
7.9 Case study of Problem HIDDEN-UNC . . . 139
7.10 Train on synthetic, test on real . . . 140
7.11 Comparison of REWARDAGG with CEM and FORWARDTRAINING . . . 141
7.12 Evaluation on a dataset of 2D search problems . . . 142
7.13 Evolution of search frontier of SAIL . . . 143
7.14 Adaptive behaviour of SAIL . . . 144
7.15 Evaluation of SAIL on an autonomous helicopter in a canyon environment . . . 145
7.16 Evaluation of SAIL on a real quadrotor . . . 147
7.17 The robot in a dark room problem . . . 148

8.1 Real world planning problems where edges are correlated . . . 152
8.2 Planning on Generalized Binomial Graphs . . . 153
8.3 Equivalence between the feasible path identification problem and DRD problem . 157
8.4 The DRD problem split into ECD problems . . . 159
8.5 The 'one region versus all' ECD problem . . . 160
8.6 Overall framework combining DIRECT and BISECT . . . 164
8.7 UAV planning and robot arm planning . . . 167
8.8 Comparison of algorithms on datasets . . . 168
8.9 Normalized cost of algorithms on datasets . . . 169
8.10 Analysis of DIRECT +BISECT . . . 169
8.11 Belief propogation in DIRECT . . . 170
8.12 Evaluation on real flight test data from an autonomous helicopter . . . 171

9.1 Overview of the input and output to the trajectory planner . . . 179
9.2 Illustrations of route constraints . . . 180
9.3 The 3 stage decomposition of the trajectory planning problem . . . 182
9.4 The trajectory planner block diagram . . . 183
9.5 Overview of the route guide generation algorithm . . . 186
9.6 The executive ensures trajectories are guaranteed safe . . . 190
9.7 Evaluation of planning architecture on 3 distinct UAVs . . . 191
9.8 Obstacle avoidance in different environment . . . 192
9.9 Execution of long missions . . . 193
9.10 Guaranteed safe architecture in operation . . . 194
9.11 Generalization to multiple vehicles . . . 194

10.1 A unified framework for adaptive motion planning . . . 196
10.2 Learning a good roadmap . . . 199
10.3 Learning anytime planning via incremental densification . . . 200
10.4 Planning problems are correlated temporally . . . 201
10.5 Learning to gather enough information to plan correctly . . . 202

A.1 Dynamics constraints from performance charts of a helicopter . . . 211

D.1 Illustration of different feature extractors 221
D.2 Comparison of different feature extractors 223

List of Tables

1.1 Mapping from challenges to chapters 9
1.2 Connections made in this thesis 10

3.1 Planning algorithms as sum of components 38

7.1 Mapping from problem and policy type to algorithm 132

C.1 Surrogate problem on curvature constrained state space 215
C.2 Library of expert sampling based planner for curvature constrained state space . 216
C.3 Surrogate problem on 3D state space 217
C.4 Library of expert sampling based planner for 3D holonomic state space 217
C.5 Surrogate problem on space of smooth trajectories 217
C.6 Surrogate problem on a graph of dynamically feasible motion primitives 217
C.7 Library of expert discrete search based planners 218

1

Introduction

Motion planning, the task of computing collision-free motions for a robotic system from a start to a goal configuration, has a rich and varied history [LaValle, 2006]. Up until now, the bulk of the prominent research has focused on the development of tractable planning algorithms with provable *worst-case performance guarantees* such as computational complexity [Canny, 1988a], probabilistic completeness [LaValle, 1998] or asymptotic optimality [Karaman and Frazzoli, 2011]. In contrast, analysis of the *finite time performance* of these algorithms has received considerably less attention.

While such questions are motivated primarily by practical robotic applications, the answers in fact point to something more fundamental. Hsu et al. [1999a] show that, even for the simplest instance of the path planning problem, the likelihood of finding a path in a fixed time budget depends intricately on the connectivity of the free space. Hence factors that affect the connectivity, such as obstacle configurations or robot dynamics, are likely to have a huge influence on the finite time planning performance. Up until now, the characterization of the *expected performance* of these algorithms on real-world planning problems has been limited due to the lack of standardized datasets or robotic platforms. However, recent advances in affordable sensors and actuators have enabled mass deployment of robots that navigate, interact and collect real data. This motivates us to examine new algorithmic questions such as: "How can we design planning algorithms that, subject to on-board computation constraints, maximize their expected performance on the actual distribution of problems that a robot encounters?"

1.1 Motivational examples and applications

The astounding advancement of robot autonomy has enabled mobile robots to break out of contrived laboratory settings and be deployed in the real world. While the field of self-driving autonomous cars has already breached the consumer market, research in the field of unmanned aerial vehicles (UAVs) that explore [Scaramuzza et al., 2014, Shen et al., 2012], inspect [Yoder and

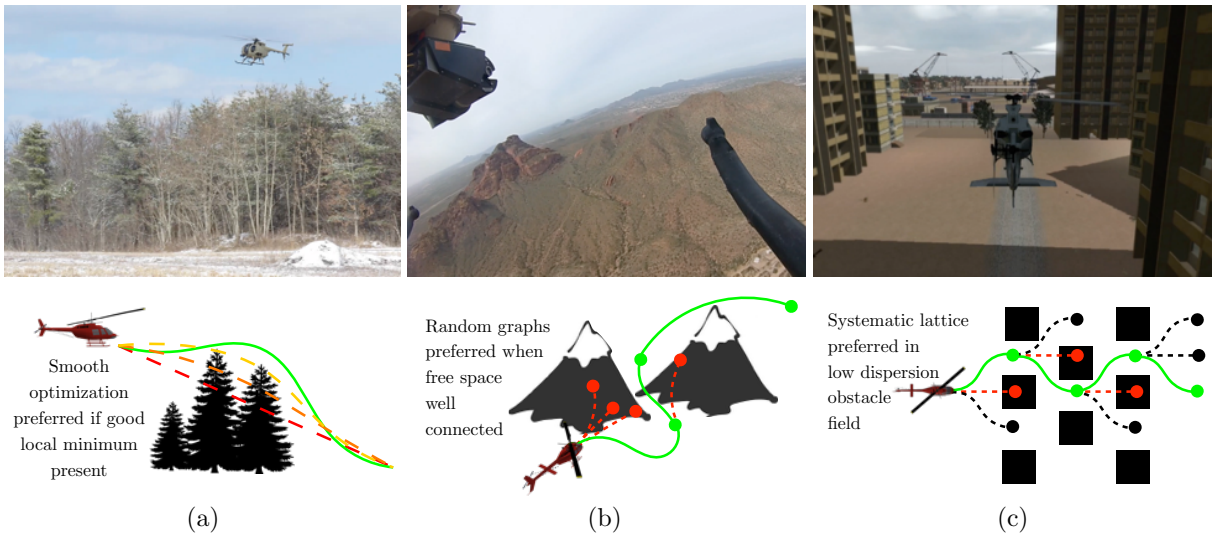


Figure 1.1: A spectrum of environments in which an autonomous helicopter has to operate. Each environment corresponds to a planning problem that favours particular search strategies. (a) Operations over tree lines favors a smooth trajectory optimization approach that is careful to respect glide slope limits while keeping the vehicle collision-free. (b) When avoiding mountains, random sampling methods that power through free space and focus search around the obstacles are effective. (c) When flying in Manhattan environments where obstacles spread out, search on a lattice with small edges is more likely to find collision-free paths.

Scherer, 2016] and deliver cargos [Choudhury et al., 2014, Whalley et al., 2014] shows that these will soon follow suit. A unifying theme is that as these mobile robots discover the world on-the-fly, they have to react in real-time to keep the system safe. Hence the finite time performance of such planning systems is of utmost importance. We show with supporting examples and applications that ensuring consistent planning performance is challenging. While we restrict our scope to mobile robots, the ideas discussed here are broadly applicable to other domains where real-time planning is critical.

1.1.1 Mobile robots navigating in varying scenarios

As mobile robot systems increasingly operate outdoors in the real world, the scenarios encountered by the robot vary widely and have a significant impact on its navigation performance. Consider the example of an autonomous helicopter designed for cargo delivery [Choudhury et al., 2014]. In order for it to complete its required tasks, it has to fly a wide range of speeds and stay close to the ground as it navigates through partially known environments. Fig. 1.1 shows a spectrum of different scenarios that it encounters - high speed GPS denied navigation in mountains or low speed maneuvering in dense urban canyons. Not all scenarios are always equally likely - the distribution over scenarios is dictated by the specifics of the mission that the robot has to execute. As the nature of the planning problems for these scenarios differ significantly, it is challenging for a planning algorithm to have consistent real-time performance during the mission. It is also impractical for a human designer to continue tuning algorithm parameters at the onset of every mission.

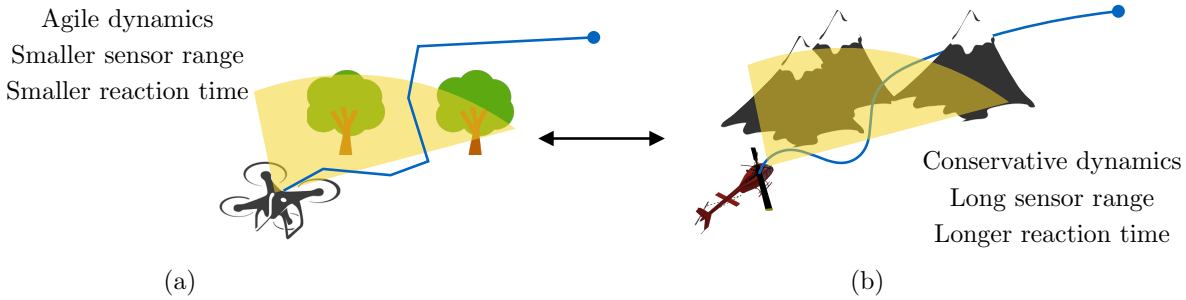


Figure 1.2: Comparison of the path planning problem in two different UAV (a) Small quadrotor: Given the attitude limits, thrust to weight ratio and low speed of operation (4 m/s), the controller for such systems can track arbitrary paths. The sensor range is small (20 m) so the UAV only senses smaller number of obstacles but has to react quickly (10 Hz). (b) Large helicopter: Due to bank and pitch constraints and high speed of operation (50 m/s), the controller can only track paths satisfying some curvature constraints. The sensor range is large (1000 m) so the UAV can sense more obstacles but has a longer reaction time (1 Hz)

1.1.2 Re-using planning software across robot platforms

Advancements in research on UAVs over the last few years have resulted in a range of platforms such as autonomous helicopters [Choudhury et al., 2014, Whalley et al., 2014], autonomous gliders [Otte et al., 2016, Warren et al., 2015] and UAVs of different sizes flying at varying speeds [Dubey et al., 2017, Mohta et al., 2018]. A standard approach to planning trajectories for such platforms is first to plan a path and then assign a speed profile such that a controller can track it. While these platforms have very different dynamics models, they share a lot of similarities from the perspective of the path planning module. The path planning problem always requires computing a 3D path with varying types of path constraints depending on the dynamics. Hence, this allows planning software to be re-used across platforms. This is a very attractive prospect from software design and verification perspective and potentially saves months of repeated development time. However, as illustrated in Fig. 1.2, the real-time performance of different planning algorithms vary as the path constraints and the complexity of the world varies. It is challenging and time-consuming to tune parameters or choose appropriate search strategies for every different platform.

1.1.3 Real-time planning for unconventional dynamical systems

Many applications such as the emergency landing of UAVs [Meuleau et al., 2009] or planning for hybrid VTOL [Ozdemir et al., 2014] require planning trajectories that satisfy rather unconventional dynamics constraints. Consider the situation in Fig. 1.3(a) where an UAV damages one of the ailerons mid-flight and has to land. However, the system can only make left turns. Interestingly, many planning algorithms are perfectly equipped to deal with such instances as long as the dynamics can be specified as a differential equation with constraints on state and action. The original RRT paper [LaValle, 1998] provides an example of planning with a left-turn-only car as shown in Fig. 1.3(b). Because RRTs are probabilistically complete, given enough time, they will find a path if it exists. However, real-time planning implies that there isn't enough time

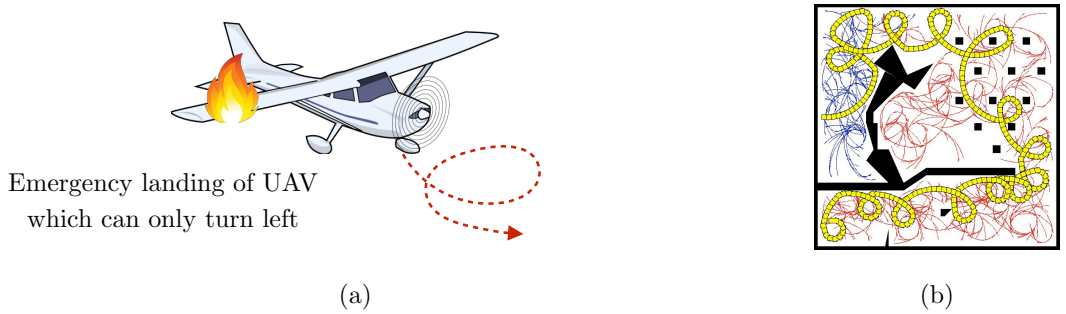


Figure 1.3: (a) The motion planner has to quickly plan a landing for a UAV whose right wing is damaged mid-flight. As a result, the UAV can only turn left. (b) Planning algorithms are equipped to deal with such unconventional dynamics as shown in this classic RRT example from LaValle [1998] where a car can only turn left. Because RRTs are probabilistically complete, given enough time they will be able to find a feasible plan if it exists. However, the tree explores a lot of configurations in this process. In the emergency landing setting, a planner does not have the time budget to do so.

for the system to explore all possible configurations. Moreover, it is difficult to apply any type of human intuition for these unconventional systems to improve real-time planning performance.

1.2 Why does a motion planning system need to adapt?

The need for a motion planning system to be cognizant of the nature of the planning problems can be attributed to the inherent hardness of the real-time planning problem. The factors contributing to the hardness can be distilled into three major categories

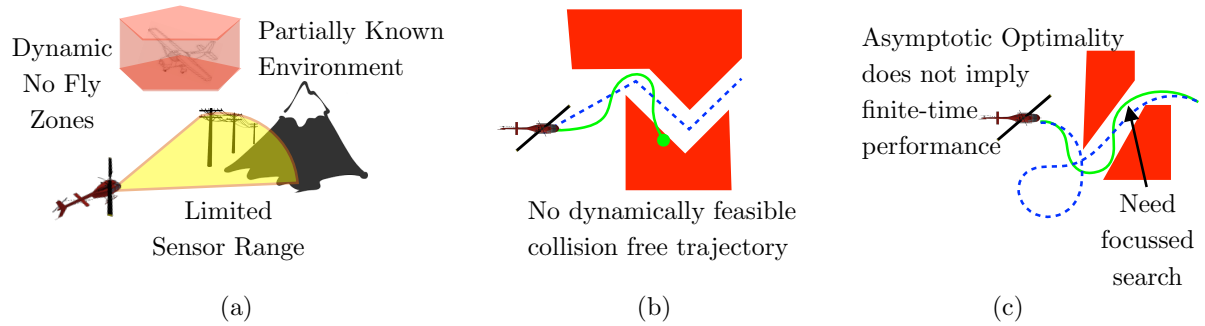


Figure 1.4: Factors contributing to the hardness of motion planning for mobile robots. (a) Limited sensing range and dynamic no-fly-zones (NFZs) requires fast re-planning (b) Dynamic constraints imply that even if there is a collision-free configuration space path, it does not mean a feasible trajectory exists in the same homotopy. (c) The real-time performance of the planning algorithm outweighs any asymptotic guarantees it might have.

1. **Partially known world:** The full environment through which the robot is navigating is not known apriori and must be perceived with on-board sensors. Fig. 1.4(a) depicts the environment from the perspective of an autonomous helicopter. The limited sensing range in conjunction with unforeseen no-fly-zones (NFZs) forces the system to react immediately

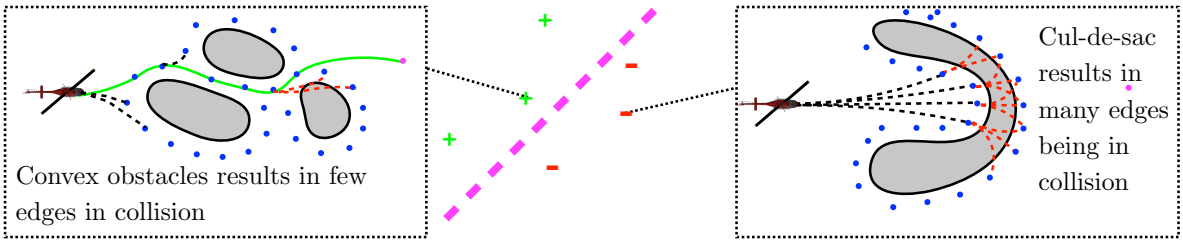


Figure 1.5: An adaptive planner, broadly speaking, classifies the environment into whether a search strategy would be effective or not. The search strategy shown here creates a search graph on the surface of obstacles. This is effective in environments with convex clusters of obstacles. However, in environments with highly non-convex obstacles, this search strategy would be ineffective.

and deviate significantly from its nominal plan. In addition, the system must continually re-plan to account for the ever-changing nature of its partial knowledge.

2. **Kinodynamic planning in high-dimensional state space:** Mobile robots are dynamical systems with actuator limits [Murray et al., 1995] and can even be underactuated [Koo et al., 2001]. Motion planning for such systems requires defining a *kinodynamic problem* in the *state space* of the robot which is high dimensional (Chapter 13.2.1 in LaValle [2006]). For many systems, the planning problem is *nonholonomic* [Laumond et al., 1998] and constrains the range of feasible motions the robot can execute [Kelly and Nagy, 2003]. Fig. 1.4(b) highlights the severity of the issue where a collision-free path might not imply the presence of a dynamically feasible trajectory around it.

3. **Real-time constraints with limited computation budget:** Since the robot is required to constantly re-plan, it is subjected to a hard time constraint to compute a feasible plan to follow using limited on-board computation. Since mobile robots have significant drift, i.e. cannot stop in place, the motion planner does not have the flexibility of elongating this time budget. However, as Fig. 1.4(c) illustrates, simply requiring a feasible plan to be produced in this time period is insufficient. A plan of sufficient solution quality is required to ensure mission success.

For the reasons stated above, it is very hard to engineer a "one size fits all" motion planner that has consistent real-time performance across all planning problems. On the other hand, if we were to make strong assumptions about the underlying structure of the planning problem, it would become easier to design a motion planner that meets performance requirements whenever the assumptions hold true. For example, Fig. 1.5 depicts a motion planner that focuses its search graph to lie only on the surface of obstacles. It assumes that the planning problem is such that such a graph would require minimal collision checking effort to arrive at a solution. For instance, this holds true in certain environments with convex obstacles. Such "precision planners" are brittle - they produce high quality solutions where such assumptions hold true but there are a large number of planning problems where they are unable to even find a feasible plan. In many cases, it might not even be possible for a human engineer to ascertain when these assumptions fail to hold. This implies that designs along the lines of a "rule of thumb" decision

tree will not scale and will eventually be rendered ineffective.

Hence, this thesis departs from the paradigm of a human engineered über motion planner and advocates the ability of a motion planning system to be *adaptive*. Borrowing the notion of adaptivity from Oreizy et al. [1999], we define an adaptive motion planner as a module that focuses its search strategy according to the likelihood of its success on a distribution of planning problems it expects to encounter. This broad definition is intended to subsume any motion planning system that reconfigures itself in response to a change in distribution without requiring a human to re-design it. This could mean using context about the planning problem to select planning strategies as shown in Fig. 1.5. This could also mean a planning system that performs inference during the search process and adapts its search accordingly.

We hypothesize that adaptation is key to bridging the gap between "brittle precision planners", that work well only in certain situations, and a robust planning system, that consistently produces high quality plans over a range of environments :

> *The thesis proposes that in order for a motion planner to consistently meet the real-time kinodynamic planning performance requirements of a mobile robot, it must adapt its search strategy to the distribution of planning problems the robot encounters.*

To address this problem in depth, this thesis focuses on the area of motion planning for UAVs. We do not explicitly worry about planning properties such as worst-case complexity and asymptotic optimality, but instead analyze the empirical real-time performance of planning modules. We focus on robotic systems that are likely to encounter a significant variation in the distribution of planning problems. In this regime, we operate under the assumption that a single static motion planning module is unlikely to have satisfactory performance. We will define adaptation as a process of making predictions using context extracted from the planning problem or using inference made by the performance of the algorithm itself during the search process.

1.3 Problem characterization

We break down the problem into 3 principle challenges arranged in order of increasing levels of adaptivity expected from the planning system

Challenge 1. Fixed distribution - Design a non-adaptive motion planner

We start off by considering the challenge of designing a motion planning system that meets performance requirements for a fixed distribution of planning problems known to a human engineer at design time, as illustrated in Fig. 1.6. Although this challenge is restrictive in terms of scope, it is still relevant as it cannot trivially be solved by an arbitrary candidate motion planner. This challenge has two aspects to it.

Firstly, we wish to investigate a framework for systematically designing planners that can exploit structure present in planning problems in order to have good real-time performance. In contrast to conventional motion planning research, this complementary line of inquiry seeks

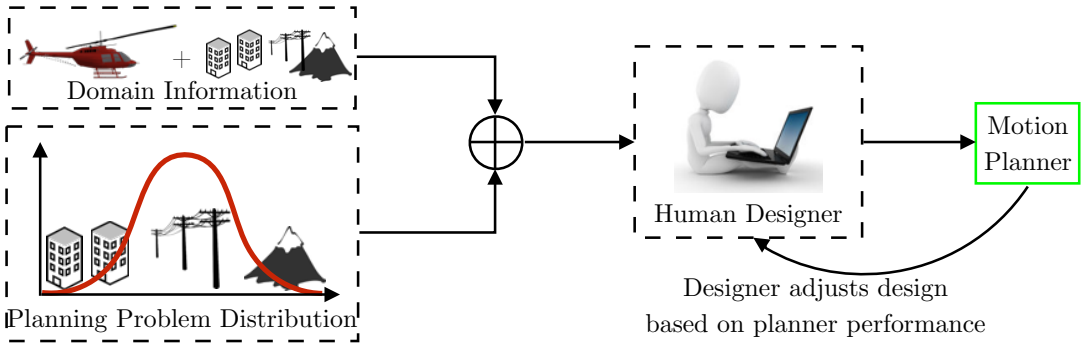


Figure 1.6: Overview of a non-adaptive motion planner. The human designer has access to both domain information - kinds of planning problems which in turn depend on vehicle dynamics and environment types - and distribution over planning problems. The objective is to design a motion planner that meets performance requirements for this known fixed distribution of planning problems.

to understand the inverse mapping from planning problems to suitable search strategies. This framework can be far more limited in scope than that proposed by LaValle [2006] which focuses more on a universal grammar for planning algorithms. On the other hand we wish to think about algorithms more seamlessly than the UAV planning taxonomies created by Kendoul [2012] and Goerzen et al. [2010].

Secondly, it also requires addressing the strategies one can use when expected performance is not met by the planning system.

Challenge 2. Varying distribution - Design a black-box adaptive motion planner

We next consider the challenge that the distribution of planning problems can vary with different missions. This happens because the robot might operate in a new environment, a new speed regime or with changed dynamics. Unlike Challenge 1, the system cannot be redesigned.

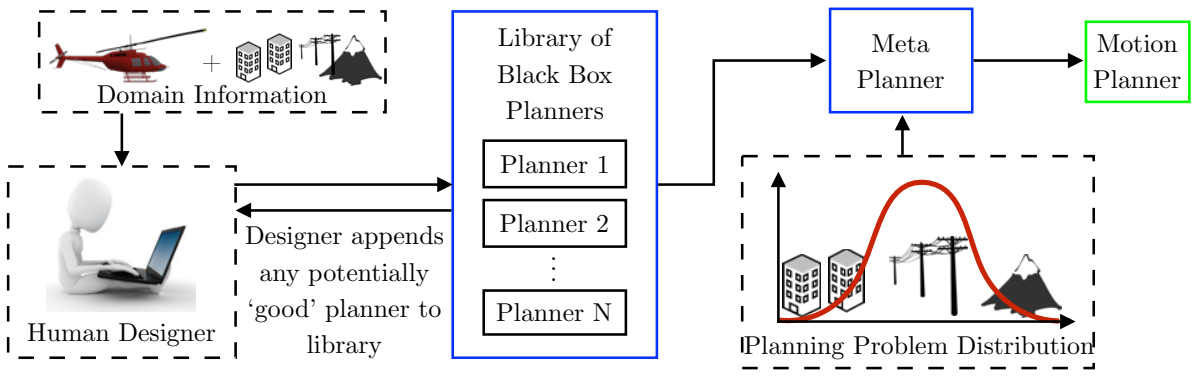


Figure 1.7: Overview of a black-box adaptive motion planner. The human designer has access only to domain information. This is used to create a library of black-box planners. The objective is to design a meta-planner that takes as input the planning problem distribution and selects a black-box planner from the library.

We first consider the paradigm in which a planner is treated as a *black-box atomic operation*

- no amount of intervention or inference is allowed during the search process. We assume, for simplicity, that we have a library of such black-box planners designed by the human using domain information. As Fig. 1.7 illustrates, black-box adaptation involves designing a *meta-planner* that effectively schedules these black-box planners. The meta-planner is free to use whatever information it needs to make such a decision such as context extracted from the planning problem or performance information of planners at previous time steps.

The black-box paradigm is attractive in how it decouples the problem of adaptation from planner design. The planner design question becomes - "What makes a good library of black-box planners?". The adaptation question becomes - "Given a set of black-box planners, how can we train a meta-planner to maximize performance on a given distribution of planning problems?"

Challenge 3. Varying distribution + computational efficiency - Design a white-box adaptive motion planner

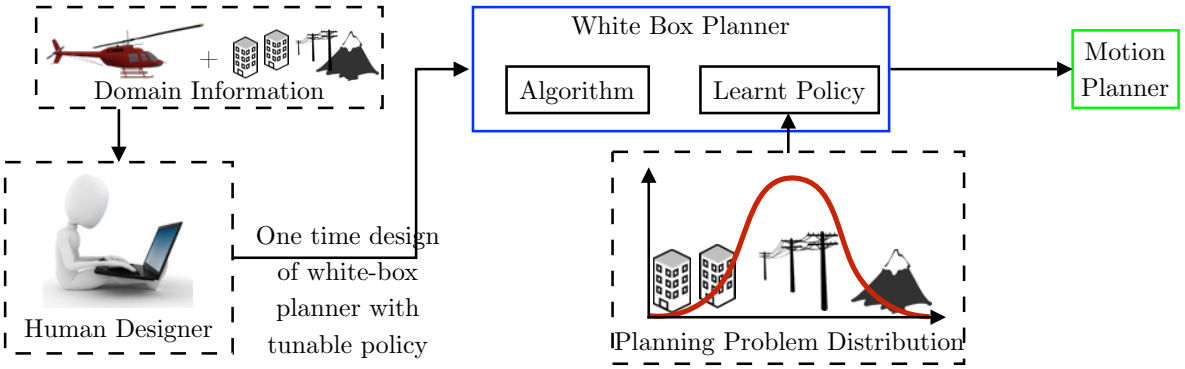


Figure 1.8: Overview of a white-box adaptive motion planner. The human designer has access only to domain information - the support of planning problems. He creates a white-box adaptive planner - a customized search strategy with one adaptive element. The adaptive element is a policy that makes certain decisions during search - such as collision check or act as a heuristic or sample states. The objective is to train this policy on the target planning problem distribution to maximize performance of the planner.

There are a couple of issues with the black-box paradigm. Firstly, it is restrictive. It does not allow any sort of intervention *during* a planning cycle. Often a lot of inference can be made about the planning problem in the intermediate stages of search. Based on this information, the planning algorithm itself can choose to adapt its search strategy.

This leads to the second issue - it is computationally inefficient. It invests time in extracting context from the planning problem before making a decision. In contrast, a *white-box* paradigm would allow it to make the most out of its planning effort.

White-box adaptation involves designing a policy to *directly alter the search strategy* used by the planning algorithm as shown in Fig. 1.8. The role of the human designer is to design the framework of the planning algorithm and the legal set of decisions the policy can make. Having created this framework, the adaptation question becomes - "How can we train such a policy to maximize performance on a given distribution of planning problems?"

Table 1.1: Mapping from challenges to chapters

Challenges	Chapters
Challenge 1: Designing effective planning modules	Chapter 3: Planning Algorithms that Exploit Structure Chapter 4: A Diverse Ensemble of Expert Planners
Challenge 2: Learning for black-box meta-planners	Chapter 5: Adaptive Ensembles of Expert Planners Chapter 6: Online Exception Planners
Challenge 3: Learning for white-box adaptive planners	Chapter 7: Data-driven Planning via Imitation Learning Chapter 8: Bayesian Active Edge Evaluation

1.4 Outline of approach

This dissertation develops an approach to motion planning for mobile robots that answers each of these three challenges. Table 1.1 shows a mapping of challenges to chapters. Chapter 2 presents some relevant notation, taxonomies and briefly reviews related work on this topic. The following chapters then present the core component of the thesis and is grouped into four parts:

1. Part I: We begin by laying the foundation for discussions on the need for planning algorithm to exploit the underlying structure of the environment in Chapter 3. We propose a framework for assembling planning algorithms to do so and show 3 concrete real-world examples - RABIT* [Choudhury et al., 2016a], SPARTAN-Lite [Nuske et al., 2015] and RRT*-AR [Choudhury et al., 2013]. We also theoretically analyze the special case of planning in a Poisson forest [Choudhury et al., 2015b].
   In Chapter 4, we address Challenge 1 - designing a high performance module for a fixed problem distribution. We show the inherent difficulty in designing a single threaded planning module and instead advocate for using an ensemble of expert planners [Choudhury et al., 2014]. We provide evidence for the efficacy of this framework with real world experiments on an autonomous helicopter.

2. Part II: We carry over the insights from the previous chapter to address Challenge 2 - adaptively selecting planners from a library of black-box planners. In Chapter 5, we present training procedures for meta-planners that predict a list of planners [Choudhury et al., 2015a, Tallavajhula et al., 2016]. We present results on various planning datasets and show closed loop evaluation of planning onboard two distinct UAV platforms.
   In Chapter 6, we address problems that may arise due to train / test mismatch. We propose an online exception planning framework and draw novel connections to the setting of online paging.

3. Part III: The black-box paradigm does not harness the full potential of planning since planners are treated as atomic operations. We address Challenge 3 by showing two distinct ways to interleave planning and inference. In Chapter 7, we formulate this as a problem of sequential decision making under uncertainty where at a given iteration a planning policy must map the state of the search to a planning action. We present a novel data-driven imitation learning framework to efficiently train planning policies by imitating a

Table 1.2: Connections made in this thesis between sub-problems in planning and machine learning

Motion Planning	Machine Learning
Selecting an ensemble of planners	List prediction for cost sensitive classification
Online library of exception planners	Online paging
Learning a search policy	Sequential decision making
Learning an edge evaluation policy	Bayesian active learning

clairvoyant oracle - an oracle that at train time has full knowledge about the world map and can compute optimal decisions [Choudhury et al., 2017a]. We present results on problems from two important domains that rely on partial information based policies - informative path planning [Choudhury et al., 2017d,e] and search based motion planning [Bhardwaj et al., 2017].
In Chapter 8, we examine the orthogonal problem of deciding which edge to evaluate in a graph. We show that this can be framed as a Bayesian active learning problem where edges are actively chosen to reduce uncertainty about the validity of paths [Choudhury et al., 2017b,f]. We present a novel framework to compute near-optimal policies and evaluate it on various datasets of motion planning problems for mobile robot, manipulators and autonomous helicopters.

4. Part IV: In this final part, we shed light on how these adaptive planning algorithms are used in practice. We present a unified planning architecture in Chapter 9 that we use across UAV platforms. We present results showing how adaptation not only helps a particular UAV solve different types of problems, but also aids in using the same underlying planning software across UAV platforms.

We conclude in Chapter 10 with a summary of the presented work, vital lessons learned and potential future directions.

1.5 Summary of contributions

We summarize the primary contributions of this dissertation. Table 1.2 shows the novel connections we draw between problems in planning and well studied problems in machine learning.

1. *Adaptation to different planning problems:* Training procedure for a meta-planner to select an ensemble of planners from a library to maximize expected performance over a distribution of planning problems.

2. *Online adaptation to handle failures:* The LRU algorithm for creating a dynamic library of exception planners online.

3. *Synthesizing planning policies with good finite time performance:* Training procedures for informative path planning (QvalAgg) and heuristic search policies (SaIL) using imitation learning.

4. *Synthesizing collision checking policies with good finite time performance:* The BISECT and DIRECT algorithm for evaluating edges on an expensive graph.

5. *Generalization across platforms:* Experimental evaluation of the above algorithms on various datasets with a focus on planning problems encountered by UAVs.

1.6 Review of experimental platforms

(a) (b) (c)

Figure 1.9: We present results from evaluation of planning onboard 3 UAV platforms (a) A full-scale helicopter (b) A large hexarotor (c) A small quadrotor

We consider a number of motion planning instances across three UAV platforms. Each of these platforms vary in scale, sensing capabilities and present unique challenges. We show that an adaptive planning module improves performance in each of these systems. Each chapter evaluates the algorithms on datasets from these applications, along with results from closed loop evaluation. Chapter 9 describes the planning architecture deployed on these systems in more detail and summarizes statistics of flight tests.

1. Full-scale helicopter(Fig. 1.9(a)): The helicopter can travel at speeds of upto $60m/s$. The dynamics of the system is heavily constrained. It is equipped with a Riegl laser scanner with a range of 1200m.

2. Large-scale hexarotor(Fig. 1.9(b)): The UAV is a DJI-M600. It can fly at speeds of upto $10m/s$. It is equipped with 2 VLP-16 lasers with a range of 100 m.

3. Small-scale quadrotor(Fig. 1.9(c)): The UAV is a DJI-M100. It can fly at speeds of upto $5m/s$. It is equipped with a spinning Hokyuo laser with a range of 15 m.

2

Background

In this chapter, we introduce the preliminary definitions and terminology that will setup the infrastructure for remaining discussions in this thesis. In Section 2.1, we define formally what we mean by a planning problem for a mobile robot. Section 2.2 talks about a distribution over planning problem distribution that a mobile robot encounters. Section 2.4 presents a taxonomy of terms that are used throughout this work and disambiguates amongst them. Finally, Section 2.5 presents a broad collection of related work.

Prior to discussing the planning problem in the upcoming section, we provide a terse description for the motion planning pipeline for a mobile robot. The mobile robot, equipped with a range limited sensor, is moving in an environment. At a given planning cycle, the motion planning module is invoked to plan a dynamically feasible collision free path to a goal point. This path is then sent to the control system of the robot which takes a step along the path. The robot receives measurements from the sensor, which it uses to update its belief about the world. This cycle is repeated. We will discuss this in more detail later in Chapter 9.

2.1 The planning problem

We formally define the kinodynamic motion planning problem that we wish to solve. Let $\mathcal{X} \subset \mathbb{R}^d$ be the state space and $\mathcal{U} \subset \mathbb{R}^m$ be the control space of the system. Let the dynamics of the system be $\dot{x}(t) = f\left(x(t), u(t)\right)$, where $x(t) \in \mathcal{X}$ and $u(t) \in \mathcal{U}$.

The objective of a motion planning system is to compute a *trajectory* - a function mapping from time to state and control values. Let the domain of the trajectory be $[0,\ t_f]$. Let $x : [0,\ t_f] \rightarrow \mathcal{X}$ be a state trajectory and $u : [0,\ t_f] \rightarrow \mathcal{U}$ be a control trajectory. We define a trajectory $\sigma \in \Sigma$ as a concatenation of state and control trajectory $\sigma(t) = \{x(t), u(t)\}$, where Σ is the set of all non-trivial trajectories.

We now define a *feasible trajectory*. The trajectory is constrained to be dynamically feasible, to lie in free space and to lie in valid regimes of operation. We categorize these constraints into

3 classes - a set of equality constraints $\mathcal{F}(\sigma(t)) = 0$, a set of inequality constraints $\mathcal{H}(\sigma(t)) \leq 0$ and a valid regime $\sigma(t) \in \Sigma_{\text{valid}}$. In addition to this the trajectory also has to satisfy boundary constraints, i.e., $\sigma(0) \in \Sigma_{\text{start}}$ and $\sigma(t_f) \in \Sigma_{\text{goal}}$.

Let $J : \Sigma \to [0,\ J_{\max}]$ be a cost functional that measures solution quality by penalizing traversal time and proximity to unsafe regions. The cost function is capped at a user specified threshold $J_{\max}$. We also follow the convention that an empty solution has a cost of $J(\emptyset) = J_{\max}$.

The *planning problem* is then defined as the following constrained optimization problem

Problem 1 (Kinodynamic Planning)**.** The kinodynamic planning problem is then formally defined as the search for the trajectory, σ^*, that minimizes a given cost function, while satisfying boundary and trajectory wide constraints

$$\begin{aligned} \min_{\sigma \in \Sigma} \quad & J(\sigma) \\ \text{s.t} \quad & \sigma(0) \in \Sigma_{\text{start}} \\ & \sigma(t_f) \in \Sigma_{\text{goal}} \\ & \mathcal{F}(\sigma(t)) = 0 \\ & \mathcal{H}(\sigma(t)) \leq 0 \\ & \sigma(t) \in \Sigma_{\text{valid}} \\ & \forall t \in [0,\ t_f] \end{aligned} \tag{2.1}$$

Appendix A specifies the details for an autonomous helicopter.

We will now define what we mean by *real-time kinodynamic planning*. Let Γ be a planning problem. We define a motion planner, $\mathcal{P}$ as an operator that takes a planning problem as input and produces a trajectory as output, i.e., $\sigma = \mathcal{P}(\Gamma)$.

Definition 2.1 (Real-time Kinodynamic Planning)**.** A real-time kinodynamic planning algorithm is an operator that can run for a time budget of at most T. If a planner is unable to find a feasible, collision free trajectory within this time budget, it returns an empty trajectory $\sigma = \emptyset$. Hence the cost of the output of the planner operator is always defined, i.e. $J(\mathcal{P}(\Gamma)) \in [0,\ J_{\max}]$ for all planning problems Γ.

2.2 A distribution over planning problems

We represent the distribution over planning problems $P(\Gamma)$ using a database of finite samples. This can be done in one of two ways.

In the first paradigm, we assume we have a complex generative distribution that we can sample worlds from. The robot's start and goal is kept invariant, while worlds are sampled from. This process is repeated N times to get a database of $\{\Gamma\}_{i=1}^{N}$ problems.

In the second paradigm, we assume we have a database of saved maps on which we want the robot to perform missions. The start and goal locations are sampled from a user designed distribution to create a database of $\{\Gamma\}_{i=1}^{N}$ problems.

It is important to note that in theory $P(\Gamma)$ depends on the sequence of planners used by the robot to navigate the map. Hence this distribution is non i.i.d. However, for the scope of this

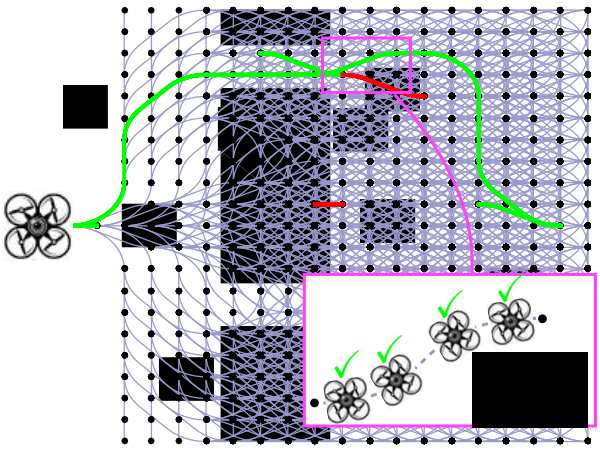


Figure 2.1: Expensive edge evaluation on a state lattice. Evaluating an edge requires stepping through states on the edge and evaluating if the vehicle is clear of obstacles.

work, we circumvent this issue by collecting data using an oracle planner that always produces high solution quality trajectories.

We define *performance* of a motion planner on a distribution of planning problems as the expected cost of trajectories returned by the planner. Given a planner $\mathcal{P}$ and a planning problem distribution $P(\Gamma)$, the performance of the planner is $\mathbb{E}_{\Gamma\sim P(\Gamma)}\left[J\left(\mathcal{P}\left(\Gamma\right)\right)\right] \approx \sum_{i=1}^{N} J\left(\mathcal{P}\left(\Gamma_i\right)\right)$.

2.3 What dominates planning time?

For many planning applications, the main computational bottleneck in motion planning is collision checking an edge. For manipulators, collision checking typically involves computing intersections between a triangular mesh model of the manipulator as well as models of the objects in the environment. The more articulated the robot, the more complex the mesh model, and the more computationally expensive the collision check. This has been independently observed in many papers [Dellin and Srinivasa, 2016, Pan et al., 2012].

In UAVs, we observe this to be particularly true for nonholonomic path planning on a state lattice. Examples of such problems are shown in Section 7.7 and Section 8.7.1. A simple illustration is shown in Fig. 2.1. For such problems, the number of vertices are not very large. However, on account of dynamic constraints, the edges of such graphs are long sequences of states. To evaluate if an edge is valid, the collision checker has to iterate through every state in the sequence. To evaluate if a state is valid, it has to collision check the geometry of the robot with the representation of the world (or ensure the robot is sufficiently clear of obstacles). Since this representation is being constructed onboard the robot as it senses the world, it cannot be pre-processed to make this process efficient. While there are techniques such as occupancy maps to speed up this step, computing a more sophisticated obstacle proximity metric such as time to collision (refer to Appendix A) requires iterating through the grid several times for an edge. Hence, the edge evaluation time dominates time consumed by other operations in the search.

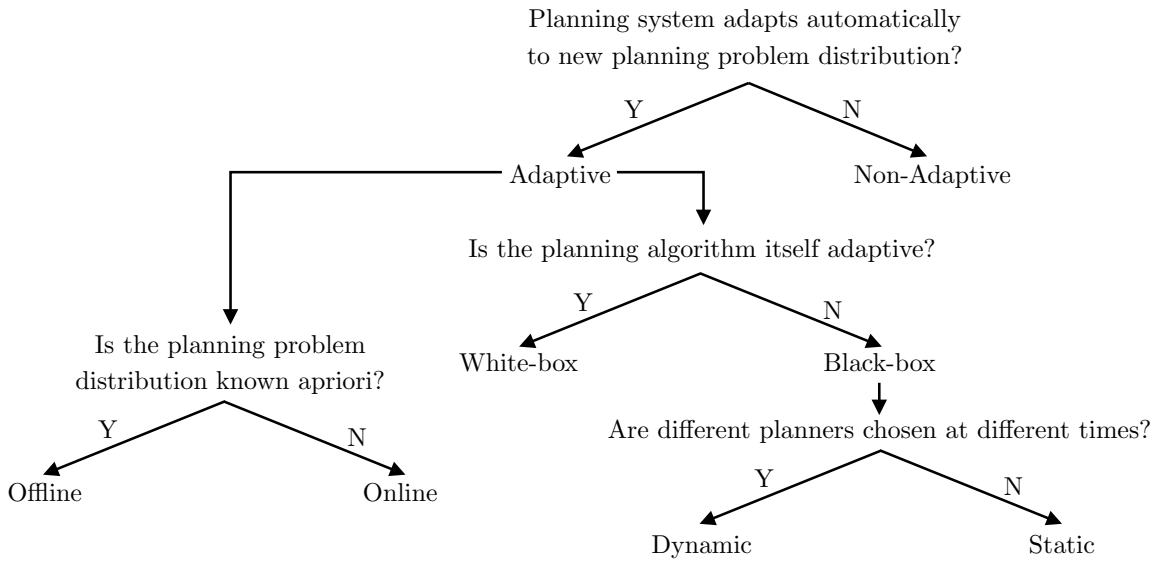


Figure 2.2: A taxonomy for adaptive motion planning

This motivates us to formulate the problem of minimizing edge evaluation in Chapter 7 and Chapter 8.

2.4 Taxonomy

In this section, we differentiate between a set of terms that we will use through out the document. These terms refer to different kinds of adaptivity of motion planners. Fig. 2.2 presents a taxonomy of these terms.

Definition 2.2 (Adaptive vs Non-adaptive Motion Planner)**.** Any motion planning system that automatically adapts to a new planning problem distribution $P(\Gamma)$ without the need of intervention by a human designer is defined to be adaptive. In the absence of this property, the system is non-adaptive.

Definition 2.3 (Black-box vs White-box Adaptive Motion Planner)**.** A black-box paradigm treats the motion planning operation as an atomic unit where intervention is not possible. Hence, in this paradigm, once a decision by a meta-planner is made to invoke a planning algorithm, that algorithm must be executed to completion. A white-box adaptive planner adapts its search strategy during the search process itself. Fig. 2.3 shows an overview of both paradigms.

Note that the black-box planner has a dedicated two step approach of extracting context and then making a decision. Contrast this to a white-box planner where the policy keeps updating itself with the outcomes of decisions made by the search strategy. Hence, although such a policy might not be as straightforward to train, it is computationally efficient.

Definition 2.4 (Static vs Dynamic Black-box Motion Planner)**.** In the static paradigm, the meta-planner makes a single decision given a planning problem distribution. In the dynamic paradigm,

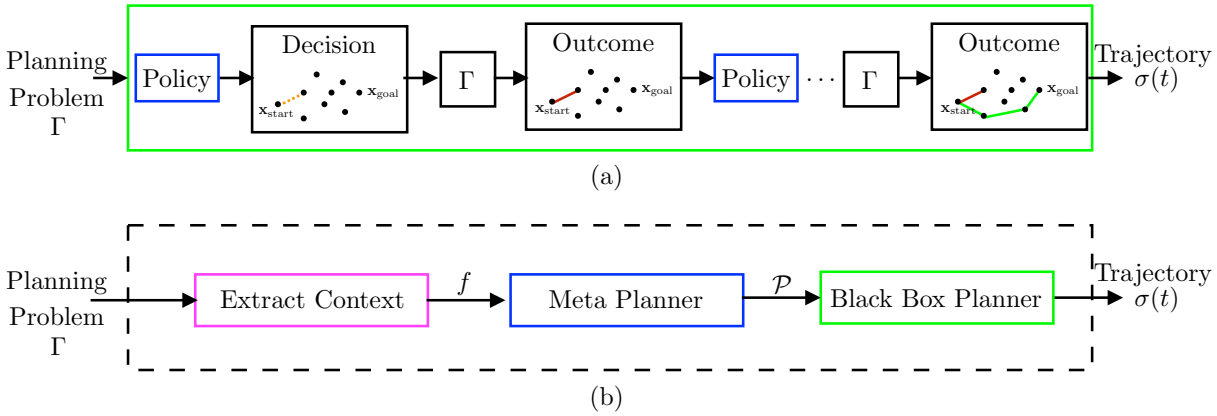


Figure 2.3: White-box vs black-box motion planner paradigm (a) In the white-box paradigm, the adaptation happens inside the planner. The planner alternates between making decisions and observing outcomes to update its decision making policy. The adaptive element here is the policy that looks at the history of decisions made and outcomes received in order to make the next decision. (b) In the black-box paradigm, the motion planner dedicates an amount of its resource to extract context. This information is then used by a meta-planner to select a black-box planner which is then executed.

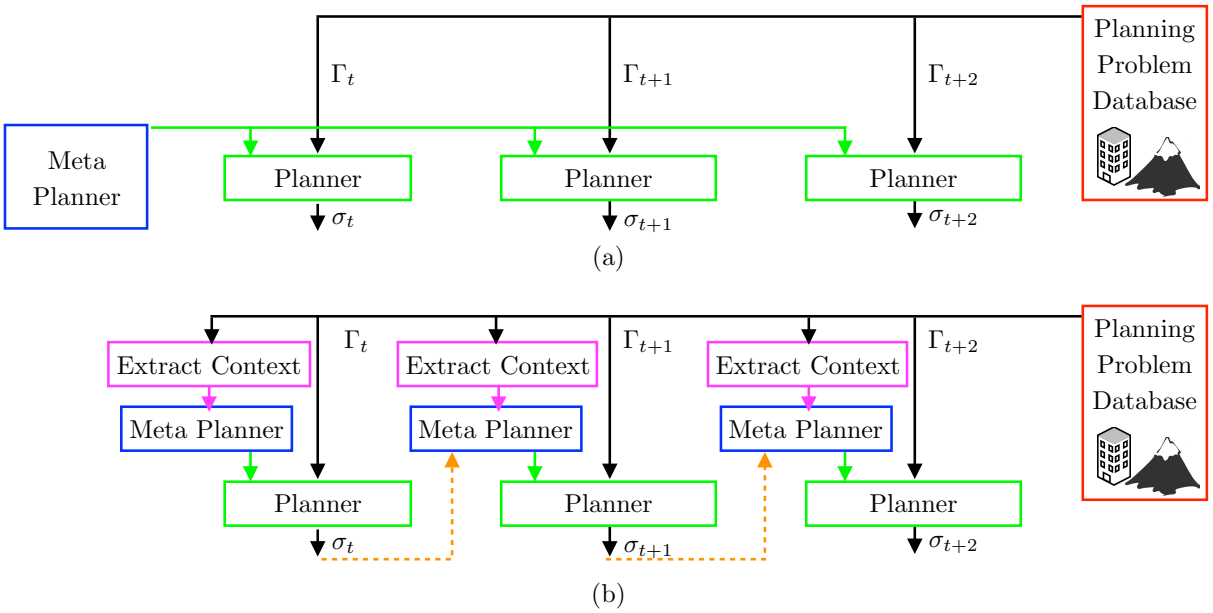


Figure 2.4: Static vs dynamic motion planner paradigm. (a) In the static paradigm, the meta-planner selects a motion planner that remains fixed for all time steps (b) In the dynamic paradigm, the meta-planner may select a different motion planner at every time step. This decision maybe based on the context extracted from the planning problem at a given time step. Optionally, the results from previous time-steps may also be used to make the decision.

the meta-planner may change its decision with time steps as shown in Fig. 2.4. Note that the dynamic paradigm does not just imply that a meta-planner decision changes with context. The decisions maybe influenced by the outcomes of previous decisions for planning problems at

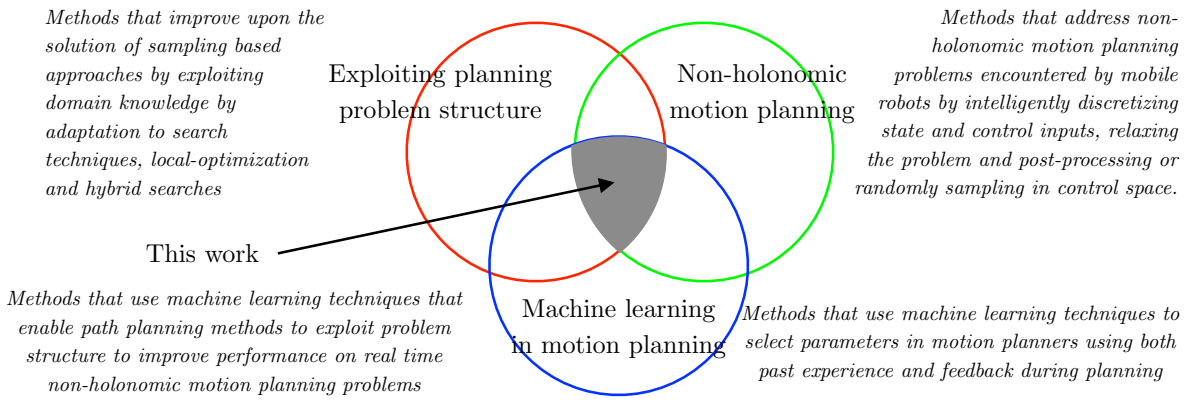


Figure 2.5: Overview of the related work covered in this section and the intersection to which the thesis belongs.

previous time-steps.

Definition 2.5 (Offline vs Online Adaptive Motion Planner)**.** In the offline paradigm, the distribution over planning problem is known apriori. Hence policies can be trained on this distribution and simply be executed at test time. In the online paradigm, the distribution is not known and may not even be i.i.d.

Definition 2.6 (General Purpose Planners vs Precision Planners)**.** General Purpose Planners are expert planners that do reasonably well on a large number of planning problems but do not necessarily have the best performance (lowest solution cost) on a bulk of these problems. Examples are conventional uniform sampling based planners, low-dispersion grids, etc. Precision planners are expert planners that have high performance on a small number of problems, but fail catastrophically on most. Examples are local trrajectory optimizer, shooting methods, etc.

2.5 Related work

Since this thesis delves into various topics in motion planning and machine learning, the relevant work spans many areas. In this section, we briefly discuss related work belonging to three most relevant themes as shown in Fig. 2.5. Firstly, we look at some diverse attempts to adapt planning algorithms to exploit problem structure and improve performance. Secondly, we delve into nonholonomic planning, its inherent hardness and efforts to solve it approximately in real-time. Finally, we look at relevant work on applying machine learning techniques to improve planning performance.

We also provide additional background material in Chapter 5, 6, 7 and 8 which are useful for understanding the proposed problem formulations and algorithms.

2.5.1 Exploiting planning problem structure

Substantial work exists on improving the solution quality of sampling-based planners by exploiting domain knowledge about the planning problem. This include adaptations to search

techniques, local optimization methods, and hybrid searches.

Many adaptations exist to the Rapidly exploring Random Tree (RRT) search procedure. Urmson and Simmons [2003] use a heuristic to bias sample generation in an RRT while Ferguson and Stentz [2006] use a series of independent RRTs in their Anytime RRTs algorithm. Jaillet et al. [2010] combine RRT with stochastic optimization techniques in their Transition-based RRT (T-RRT) algorithm, while Rickert et al. [2008] attempt to balance exploration and exploitation through gradient information in their Exploring/Exploiting Tree (EET) algorithm. Though these techniques are effective, their asymptotic optimality is limited by the underlying RRT. Karaman and Frazzoli [2011] incrementally rewire the RRT graph using random geometric graph (RGG) theory to achieve asymptotic optimality in their algorithm, Asymptotically optimal RRT (RRT*). Recent work has focused on improving the convergence rate of asymptotically optimal planners.

Alterovitz et al. [2011], Akgun and Stilman [2011], and Nasir et al. [2013] all use path-biased sampling in their algorithms. This increases the likelihood of sampling near the current solution and the convergence to a local optimum, but results in a nonuniform distribution that can decrease the likelihood of finding solutions in *other* homotopy classes. Gammell et al. [2014] improve convergence for problems seeking to minimize path length by directly sampling a (upper-bound) heuristic in their Informed RRT* algorithm. Although this has linear convergence to the optimum in the absence of obstacles, the presence of obstacles in practical problems prevents the subproblem from shrinking indefinitely. Arslan and Tsiotras [2013] use dynamic programming Bellman [1954] and Lifelong Planning A* (LPA*) techniques Koenig et al. [2004] in their RRT$^{\#}$ algorithm to improve RRT* rewiring. This improves convergence but does not directly focus the search.

Janson et al. [2015b] use a marching method on a set of samples in their Fast Marching Tree (FMT*) algorithm. The search expands outward from the start in order of increasing cost-to-come; however, it is not anytime and must be restarted if more samples are needed to find a solution. Salzman and Halperin [2015] extend FMT* to quasi-anytime performance with their Motion Planning Using Lower Bounds (MPLB) algorithm. Denser sets of samples are searched using lower-bounding estimates of the solution cost through states, with improved solutions being found only when a search finishes. They state that this can be done efficiently by reusing information, but they do not provide specific methods to do so. Gammell et al. [2015] combine incremental graph-search techniques with RGG theory in their Batch Informed Trees (BIT*) algorithm to create an anytime asymptotically optimal search that checks potential solutions in order of estimated cost. This is done efficiently by using heuristics to search batches of samples.

Local optimization methods focus on improving an initial suboptimal path towards a local optimum. All these methods can be used to post-process results from global searches and some can be used to solve a problem directly. While optimization can occasionally switch between topologically close classes, these methods are generally limited to the homotopy class of the initial path.

Basic techniques seek to simplify the initial path by removing redundant states through path pruning or path shortcutting [Berchtold and Glavina, 1994, Geraerts and Overmars, 2007, Hauser and Ng-Thow-Hing, 2010, Hsu et al., 1999b, Sekhavat et al., 1998]. Path pruning iteratively

improves a discrete path by considering new connections between existing vertices, while path shortcutting performs a similar procedure but also considers interpolating edges to create new vertices. In both, when a feasible connection is found the path is simplified by using it and removing the intermediate vertices.

More advanced techniques seek to optimize an initial path, independent of its feasibility, by exploiting additional information about the problem domain (e.g., cost gradients). These can be used independent of a global search; however, they may become stuck in the local optima of nonconvex cost functions.

Zucker [2009] use gradient methods to optimize an initial solution in their Covariant Hamiltonian Optimization for Motion Planning (CHOMP) algorithm. Kalakrishnan et al. [2011] use stochastic methods to replace analytical gradients in their Stochastic Trajectory Optimization for Motion Planning (STOMP) algorithm. These techniques associate a cost to obstacles and perform optimization of an unconstrained cost function. This allows for the rapid discovery of local optima, but does not guarantee that the path will be feasible (i.e., collision-free). Schulman et al. [2013] perform optimization constrained by obstacle avoidance in their trajectory optimization (TrajOpt) algorithm; however, it is still susceptible to local optima.

Hybrid search techniques combine the results of multiple, possibly different, search algorithms into a solution that is better than that of the individual inputs. Basic techniques combine the results of completed searches [Luna et al., 2013, Raveh et al., 2011]. This allows for a wide variety of methods as inputs, including both global and local techniques, but keeps each search independent. This means information discovered by one search is not shared with others, a limitation that is especially problematic in domains with difficult-to-sample features such as narrow passages.

Otte and Correll [2013] use a parallel hybrid method in their Coupled Forest of Random Engrafting Search Trees (C-FOREST) algorithm. Demonstrated with RRT*, the algorithm shares information between multiple sampling-based planners using heuristics and rejection sampling. Compared to other parallel algorithms [Ichnowski and Alterovitz, 2012], this results in a superlinear speedup in computation time; however, they only use global searches and it is unclear how to incorporate local searches into the algorithm.

Hence we conclude that the efficacy of these improvements is dependent on the planning problem structure and that mapping is not straightforward to predict.

2.5.2 nonholonomic motion planning

A substantial body of work has looked at tractable approximate algorithms to solving nonholonomic motion planning problems encountered by mobile robots. Solutions can be broadly categorized as creating lattices by intelligently discretizing state and control inputs, relaxing the problem and post-processing or randomly sampling in control space.

The term nonholonomic path planning was introduced by Laumond [1986] to describe the problem of motion planning for wheeled mobile robots (refer to Laumond [1998], Li and Canny [1993] for an overview). Curvature bounded path planning in presence of obstacles was proved to be NP-Hard [Reif and Wang, 1997]. Approximation algorithms for finding such paths were proposed by Jacobs and Canny [1989] with exact results obtained only for obstacles with bounded

curvature [Boissonnat and Lazard, 1996]. These early works provide a glimpse into the inherent hardness of the problem and the need for good approximations to get high quality real-time solutions.

There has been substantial work that look at the problem in absence of obstacles. This is known as the boundary value problem, i.e., finding a steering input to drive the system from one configuration to another. For curvature bounded systems, analytical solutions were obtained by Dubins [Dubins, 1957] and Reeds and Shepp [1990]. Continuous curvature systems were analyzed by Fraichard and Ahuactzin [2001], Fraichard and Scheuer [2004], Scheuer and Laugier [1998]. In general however, the boundary value problem can be expensive to solve. Kelly and Nagy [2003] proposed using curvature polynomials to parameterize the trajectory which they solve for numerically.

One class of approach have been the use of a state lattice as proposed by Pivtoraiko et al. [2009]. This is a lattice where edges between vertices are solved for offline using BVP solvers. These edges are called motion primitives. This acts as a wall of separation between the vehicle dynamics that create the graph and algorithms that can search the graph. This has been used to great success by Dolgov et al. [2010], Heng et al. [2011], Hwangbo et al. [2007], Likhachev and Ferguson [2009], Lindemann and LaValle [2006], MacAllister et al. [2013]. However, this process does require a fair bit of engineering in terms of lattice design, selecting a resolution, selecting a search method. These methods do not naturally adapt to changing dynamics or changing environment although progress has been made in this area by Howard [2009b].

A general solution to such problems would ultimately require planning with the dynamics as opposed to a contrived decoupled search. While the original RRT algorithm required a local planner to connect states, the Expansive Space Tree (EST) method introduced shortly after by Hsu et al. [1999a] planned by directly forward integrating random control inputs. However, this method does require careful parameter selection and quality cannot be assured. Recently, the Stable Sparse RRT (SST) method Li et al. [2015] was developed providing almost sure asymptotic convergence to an approximately optimal solution without requiring a local planner. Another method, Generalized Label Correcting [Paden and Frazzoli, 2016] was developed recently with further improvements. While this remains an active area of research, these methods are still quite a bit away from solving nonholonomic problems in real-time.

Hence we conclude that nonholonomic problems can only be approximately solved, the quality of the solution heavily dependent on the type of approximation applied. Moreover, the approximation is influenced by the dynamics of the robot and the environment in which its operating in.

2.5.3 Machine learning in motion planning

There is no clear consensus on when, where and how machine learning methods should be integrated into motion planning pipelines. We refer to Otte [2009] for a brief discussion of this dilemma. A plethora of work exists on learning models to process raw sensor measurements and predict control actions, but we do not focus on such techniques. A lot of works exists also on learning from demonstrations, but we work in a paradigm where we assume that the planning problem is well defined. Hence we focus on related work that has applied machine learning

techniques in prediction of parameters used by motion planning methods.

Jetchev and Toussaint [2013] was an early work on predicting seeds for trajectory planning. Cost regression and classification were implemented, without the formalism of loss-sensitive classification. Their results show that cost regression is a more difficult task. Dragan et al. [2011] predicted the usefulness of end-effector goals for trajectory planning on a manipulator. Some heuristics in their procedures are explained when a surrogate loss is used to derive algorithms. Both Jetchev and Toussaint [2013] and Dragan et al. [2011] predicted a single element. While a stronger predictor class Π was suggested to increase performance, a list was not. ConseqOpt for list prediction appeared in Dey et al. [2013]. The algorithm was presented along with a theoretical guarantee. A follow up to ConseqOpt by Ross et al. [2013] considered list prediction in a different setting. It was shown that when trained online, with data streaming in, a single predictor for predicting a list works well. Zucker Zucker [2009] generates a 'behavior library' of optimized trajectories and predicts the best trajectory given a query. Berenson et al. [2012] generates a library of past plans and uses a heuristic to select one that can be repaired easily to solve a new environment. Pan et al. [2014] predicts if a seed trajectory will be successful for local optimization. Poffald et al. [2014] uses a library of motion primitives and predicts the best primitive that can be adapated for a new environment. Wzorek et al. [2010] predicts a motion planning strategy, from a library, that can be applied to repair a plan. Palmieri and Arras [2015] learns a distance metric for an RRT to predict the nearest neighbour. In conclusion, these work fall under the category of a black-box adaptive motion planner (Definition 2.3) where a decision is made and explicit feedback about its success is used to refine the decision.

There is a body of work in using machine learning techniques to divide the C-space into regions and select different planning strategies to apply to these regions. One of the earliest approaches in this area was presented by Morales et al. [2005]. Rodriguez et al. [2008] classifies regions based on the entropy of samples and uses this to refine sampling. Kurniawati and Hsu [2008] and Hsu et al. [2005] present adaptive hybrid sampling strategies for PRM methods. Tang et al. [2006] guide RRT methods based on an ensemble of metrics. Zucker et al. [2008] formulated an adaptive workspace biased sampling technique as a policy optimization problem and use the REINFORCE algorithm to solve it. Baldwin and Newman [2010] learn how to bias RRTs from experience. Burns and Brock [2005b] formulate a utility criteria for guiding sampling. Burns [2007] surveys approaches that guide sampling by exploiting structure. Diankov and Kuffner [2007] uses statistical techniques to decide which node of a tree to sample around. Dalibard and Laumond [2011] use PCA techniques to analyze the nature of free space to guide sampling. Pan et al. [2012] use instance based learning to model free-space. Choudhury et al. [2016b] also model free-space and use this to reason about the pareto optimal surface of configuration space beliefs. A recent work by Ichter et al. [2017] learns a generative model to sample from states likely to belong to the optimal path. They learn this using a conditional variational auto-encoder. In conclusion, this work falls under the category of an adaptive white-box planner (Definition 2.3) where a learning method models the nature of C-space to guide the search.

Howard [2009a] has extensively analyzed the role of adaptation in model predictive trajectory generation for mobile robots. An adaptive search space is presented that exploits environmental information to maintain feasibility and locally optimize the mapping between nodes and states.

Napoli et al. [2018] builds on this approach by learning models that predict possible improvement on optimizing maneuvers in state lattices. Knepper and Mason [2012] shows how structure in the environment can be exploited to adapt path sampling in search.

There is a large scope for applying machine learning techniques in heuristics for motion planning. This topic falls under the umbrella of machine learning for general purpose planning as reviewed by Jiménez et al. [2012]. Yoon et al. [2006, 2008] proposed using ordinary least squares regression to learn the difference between actual distance to go and estimate given by FF-Heuristic [Hoffmann and Nebel, 2001]. Xu et al. [2007, 2009, 2010] improve upon this in a beam-search framework. They learn a discriminative model to rank top successors to include in the beam search. Arfaee et al. [2011] learned heuristics by iteratively improving on prior heuristics. ús Virseda et al. [2013] learn combination of heuristic values that most accurately predicts the cost-to-go. Wilt and Ruml [2015] build heuristics for greedy search using a Kendall rank coefficient. Thayer et al. [2011] address the problem of online heuristic learning during search. Garrett et al. [2016] frame heuristic learning as a ranking problem.

A relevant class of work falls under the framework of multi-heuristic A* by Aine et al. [2016] where multiple heuristics are combined in a round robin fashion sharing information while preserving optimality guarantees. Narayanan et al. [2015] improve this framework with uncalibrated heuristics. Islam et al. [2015] uses this framework to dynamically add heuristics to aid planning. Phillips et al. [2015] investigates efficient approaches of scheduling heuristics in this framework. Aine et al. [2015] show how this framework allows one to leverage experience. Aine and Likhachev [2016] generalizes this framework further to show how a portfolio of planning methods can be share information. In conclusion, this body of work falls under the category of an adaptive white-box planner (Definition 2.3) where multiple policies are acting in conjunction to guide the search. The advantage of this framework is two-fold: the retention of guarantees of A* and the ability for multiple strategies to share information. However, learning such policies is complicated by a super-modular sharing effect and is a topic for further research.

We also highlight a couple of interesting relevant approaches. Learning from experience [Ratliff et al., 2009b]) while primarily used for imitating human demonstrations, is an interesting tool for functional optimization. It would be interesting to adapt such methods for learning relaxations of the original motion planning problem that can be used as heuristics. Reinforcement planning [Zucker and Bagnell, 2012]) is similar in spirit where a low dimensional cost function which is used to plan a path that is followed by a controller. Policy gradient techniques are used to solve this problem. Learning dimensional descent [Vernaza and Lee, 2011] is a technique to learn a descent direction that causes the greatest variation in cost. Finally, a rather simple way of incorporating experience is dumping previous paths in a pool that heuristics can use to speed up search [Phillips et al., 2012]. Learning which paths to keep and how to adapt such paths is an interesting direction for research.

Part I

Foundations

3

Planning Algorithms that Exploit Structure

This chapter provides a good entry point into the problem of adaptive motion planning. Before we can ask how or why planning systems must adapt, we need to better understand how the performance of a planning algorithm depends on the "structure of the problem". We define this structure to mean the configuration of obstacles in the environment as well as the reachability of the robot. By picking apart the mechanics of a planning algorithm, we see that the underlying implicit graph plays a major role in determining the finite time performance of these algorithms. Since Challenge 1 requires us to design good planners for different problem distributions, we develop some insight on how one may go about "exploiting this structure" by designing strategies to encode domain knowledge and prior experience about where the solution might lie.

Instead of working with the kinodynamic motion planning problem in Problem 1, we focus on a simple subproblem - the *optimal path planning* problem. This allows us to focus our thoughts on the geometry and the connectivity of the free configuration space, without having to worry about dynamic feasibility or speed. We will return to the original problem in Chapter 4 and show that the insights we develop here easily carry over.

We begin with a gentle start to the problem in Section 3.1 where we review the seminal analysis of expansive spaces first presented by Hsu et al. [1999a]. We then define the optimal path planning problem in Section 3.2. We describe a framework in Section 3.3 that aids in the design process of path planning algorithms by exploiting the particular structure of different problems. We consolidate this framework with 3 different novel planning algorithms. Section 3.4 presents a hybrid local global search, RABIT* [Choudhury et al., 2016a], that exploits the presence of difficult to sample homotopy classes. Section 3.5 presents a 3D visibility graph search, SPARTAN-Lite [Nuske et al., 2015], that exploits the geodesic properties of a path in an environment with sparse obstacle clusters. Section 3.6 presents a reachability informed graph search, RRT*-AR [Choudhury et al., 2013], exploits the reachability volume of a system to prune away infeasible edges to accelerate search. Finally, Section 3.7 presents a theoretical perspective on the relation between the distribution of obstacles in an environment and the

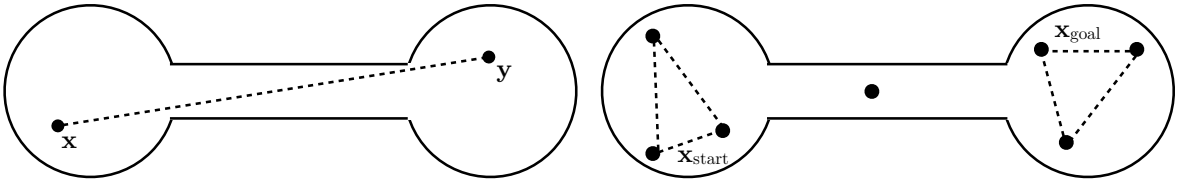


Figure 3.1: "The narrow passage": a classic example from Hsu et al. [1999a]. A configuration space whose connectivity is difficult to capture via random sampling due to the presence of a long and narrow passage. Randomly sampled roadmaps end up having multiple connected components instead of one.

planning resolution that can guarantee the existence of a solution.

3.1 A gentle start

Path planning is a fundamental problem in robotics [Latombe, 1991]. Given the geometry of a robot and obstacles, a path planner is required to generate a collision-free path between an initial and a goal configuration. Reif [1979] showed that a complete planner, i.e. a planner that finds a path if one exists and indicates none otherwise, will take time exponential in the number of degrees of freedom. Since then, a plethora of approximation based strategies have arisen that have shown good empirical performance.

However, the *finite time* performance of these strategies have been found to vary significantly on varying the configuration of obstacles. This has led to a situation where there is no clear consensus on what is the best planning strategy to employ. In this section, we wish to shed some light on the question:

> How is the finite time performance of a planning approach affected by the configuration of obstacles?

We examine this question in the context of sampling based planning. We review the seminal analysis of Hsu et al. [1999a] that answers this question by crisply characterizing the nature of the configuration space. We then use the framework to examine the role statistical methods can play in planning.

3.1.1 The shape of free space

We now introduce a few simple notations that will help us characterize the shape of the space in which we plan. We follow Hsu et al. [1999a] and analyze the geometric path planning problem. The configuration (or state) of the robot specifies its position and orientation with respect to a fixed world frame. Let X be the configuration space, i.e. the set of all such configurations. A subset of these configurations, $\mathcal{O} \subset X$ are in collision with obstacles, and $X_{\text{free}} = X \setminus \mathcal{O}$ is the resulting free space.

We are examining a class of sampling based roadmap planners. They typically construct a roadmap graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ to approximate the connectivity of X_{free}. Configurations are sampled uniformly at random from X, checked if they are free and retained in $\mathcal{V}$ as *milestones*. An edge

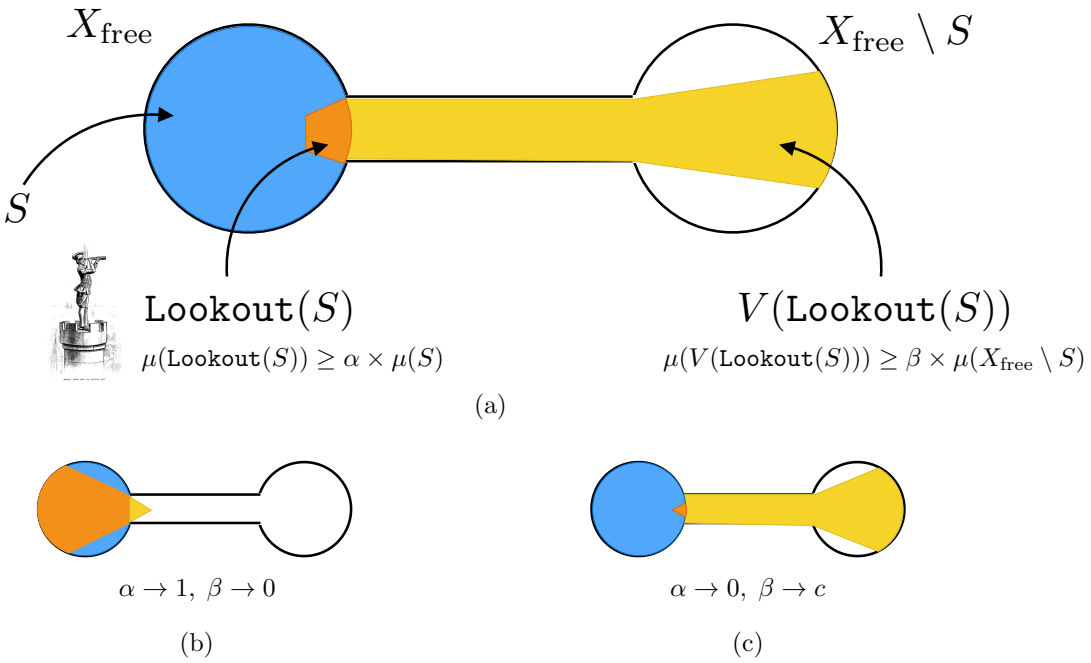


Figure 3.2: (a) Illustration of expansive space characterization with α, β. The narrow passage must have one of these parameters to be small. (b) If we push α to 1, β drops to 0. (c) We cannot drive β to 1. We can at most drive it to $c < 1$ at which points α becomes 0

is created between two milestones if the straight line joining them lies entirely in X_{free}. The roadmap graph is then queried with a start $\mathbf{x}_{\text{start}}$ and goal $\mathbf{x}_{\text{goal}}$ configuration. The hope is that since the roadmap is approximating the connectivity of the free space, if a path exists between $(\mathbf{x}_{\text{start}}, \mathbf{x}_{\text{goal}})$ that lies in X_{free}, then there also exists a path in $\mathcal{G}$.

Narrow passages in X_{free} (illustrated in Fig. 3.1 pose a significant difficulty to such planners because the probability of picking random milestones that are connected by straight lines through such passages is small. For the example in Fig. 3.1, if we fail to sample a pair of such milestones, the roadmap $\mathcal{G}$ will contain two connected components, one in each globe, and will not reflect the connectivity of X_{free} which has one connected component.

We now try to mathematically capture this phenomenon by characterizing the free space X_{free} using 3 numbers α, β, ϵ. For any subset $S \subseteq X_{\text{free}}$, let $\mu(S)$ denote the volume of the set. For convenience assume $\mu(X_{\text{free}}) = 1$. Two configurations are visible if they are connected by a straight line in X_{free}. We will also for simplicity assume X_{free} has only one connected component (the analysis holds for multiple).

Let $V(\mathbf{x}) \subseteq X_{\text{free}}$ denote the region visible from a configuration $\mathbf{x} \in X_{\text{free}}$. Given a set $S \subseteq X_{\text{free}}$ and $\beta \in (0, 1)$, we define the $\texttt{Lookout}(S)$ to be the set of points in S that can see at least β fraction of the complement of S.

$$\texttt{Lookout}(S) = \{\mathbf{x} \in S \mid \mu(V(\mathbf{x}) \setminus S) \geq \beta \times \mu(X_{\text{free}} \setminus S)\} \tag{3.1}$$

We are now ready to characterize a space.

Definition 3.1 (Expansive Space)**.** Let α, β, ϵ be constants in $(0, 1)$. The free space X_{free} is $(\alpha, \beta, \epsilon)$-expansive if

1. Sufficient visibility: For every point $\mathbf{x} \in X_{\text{free}}$, $\mu(V(\mathbf{x})) \geq \epsilon$
2. Sufficient lookout: For any subset $S \subseteq X_{\text{free}}$, $\mu(\texttt{Lookout}(S)) \geq \alpha \times \mu(S)$

The intuition is that α and β correspond to volume of points that can contribute to new visibility regions. Say we sample a set of points M which have a combined visibility region S. A large α, β implies that it will be easy to find points (because of large α) that will expand the visibility significantly (because of large β). This will "expand" S till it covers the whole space, and we have enough information to solve the problem.

We now examine the narrow passage once again and try to characterize its α and β as shown in Fig. 3.2. We can take S to be one globe of the free space X_{free}. Then there is a very small portion of S that can see a large portion of its compliment. This implies either one of α or β has to be small. If we take β to be large enough, α will eventually have to be 0 because of the narrow passage.

3.1.2 What makes a narrow passage difficult?

We now ask the question - "How does a small α, β increase planning difficulty?" We will answer this question by linking the connectivity of the free space to the connectivity of the roadmap.

We begin by defining the linking sequence of a point $\mathbf{x} \in X_{\text{free}}$. The linking sequence is a sequence that monotonically increase the cumulative visibility region of the previous points in the sequence

Definition 3.2 (Linking Sequence)**.** The linking sequence of a point $\mathbf{x} \in X_{\text{free}}$ is a sequence of points $\mathbf{x}_0 = \mathbf{x}, \mathbf{x}_1, \mathbf{x}_2, \ldots$ and a sequence of sets $V_0 = V(\mathbf{x}_0), V_1, V_2, \cdots \subseteq X_{\text{free}}$ such that for all $i \geq 1$, $\mathbf{x}_i \in \texttt{Lookout}(V_{i-1})$ and $V_i = V_{i-1} \cup V(\mathbf{x}_i)$. The sets are the cumulative visibility volume. The points are cascaded lookouts.

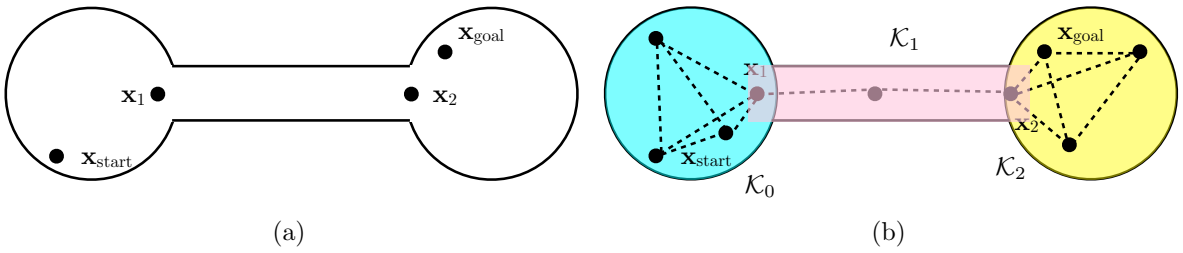
We now will state two lemmas. Lemma 3.1 states that any set of randomly sampled configurations is likely to have a linking sequence of a given length for any point. Lemma 3.2 states that the cumulative visibility volume monotonically increases.

Lemma 3.1 (Linking Sequences are Likely)**.** Suppose that a set M of n milestones is chosen independently and uniformly at random from the free space X_{free}. Let $s = \frac{1}{\alpha\epsilon}$. Given any milestone $\mathbf{x} \in M$, there exists a linking sequence in M of length t for p with probability at least $1 - se^{-\frac{n-t-1}{s}}$

Proof. Refer to Hsu et al. [1999a] □

Lemma 3.2 (Cumulative Visibility Volume Monotonically Increases)**.** Let $v_t = \mu(V_t)$ be the volume of the t^{th} set V_t determined by the linking sequence $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots$ for a point $\mathbf{x} \in X_{\text{free}}$. Then $v_t \geq 1 - e^{-\beta t}$

Proof. Refer to Hsu et al. [1999a] □



(a) (b)

Figure 3.3: (a) Addition of intermediate points (or bottlenecks) (b) These implicitly results in decomposition into expansive components which explains why a connected roadmap becomes more likely.

We now state the main theorem which is a consequence of the two lemmas. As milestones are sampled, any two milestones are likely to have increasingly longer linking sequence with increasingly larger cumulative visibility volume. Eventually the volumes will intersect and the two milestones will be connected.

Theorem 3.3 (Roadmap Connectivity)**.** Let $\gamma \in (0,1)$ be a constant. Suppose a set S of $2n$ milestones, for $n = \left\lceil 8\frac{\log(\frac{8}{\epsilon\alpha\gamma})}{\epsilon\alpha} + \frac{3}{\beta} + 2 \right\rceil$, is chosen independently and uniformly at random from the free space X_{free}. Then with probability at least $1-\gamma$, the roadmap is a connected graph.

Proof. Refer to Hsu et al. [1999a] □

Note that as α, β, ϵ get larger, the space becomes more expansive and the number of milestones required decreases in inverse proportion. Conversely, as α or β become small, the number of samples required increase significantly. This implies that when run for a finite time budget, uniform sampling based approaches would not be able to find the path passing through the passage.

3.1.3 How can learning play a role?

Fortunately, the analysis does not stop at this negative result. Hsu et al. [1999a] point out that for some problems the location of narrow passages is obvious to the human user. If he were to insert some interim points $\mathbf{x}_1, \ldots, \mathbf{x}_n$, the planner simply has to find paths connecting consecutive points. The notion of expansive spaces is useful in explaining what points would be useful and why.

We re-examine the narrow passage problem in Fig. 3.3. By specifying the intermediate points, the user increases the likelihood that sampled points in the passage will likely be link to the connected component of the roadmap belonging to at least one of the milestones. This will enable all connected components to eventually link up to one another resulting in a fully connected roadmap.

The underlying explanation is that by specifying the points, we effectively decompose X_{free} into a number of expansive components $\mathcal{K}_0, \mathcal{K}_1, \ldots, \mathcal{K}_m$ which can be possibly overlapping. Each of this expansive components corresponds to a $\alpha_i, \beta_i, \epsilon_i$. These values will be much larger than the original space because none of these components themselves contain a narrow passage. Note

that we are not *explicitly* decomposing the X_{free}. Also note that expansive space decomposition is much more relaxed than decomposing into convex cells.

Hsu et al. [1999a] end on a note that is very pertinent to this discussion

> "An open problem is, of course, to generate these intermediate points automatically. It would not only relieve the user of the burden of specifying intermediate points, but also help in situations where narrow passages are not obvious to the user. This problem may not have an efficient general solution, but maybe solvable in some specific planning environments"

We expand on this point and look to answer the question of whether we need statistical approaches, and if so what should we look to learn. We build up our insight through a series of question answers

Q 1. How can we generalize the notion of these "intermediate points"?

Let us begin by examining the necessary properties of the intermediate points. They appear to be at the junction of two expansive components $\mathcal{K}_i$ and $\mathcal{K}_{i+1}$. However, the union of these components is not an expansive component. Hence these points are actually *bottlenecks* - locations through which paths connecting two components have to pass. The necessary criteria can be stated as

Definition 3.3 (Bottleneck points)**.** A bottleneck point $\mathbf{x}_b$ must belong to 2 components $\mathbf{x}_b \in \mathcal{K}_1$ and $\mathbf{x}_b \in \mathcal{K}_2$ which are $(\alpha_1, \beta_1, \epsilon_1)$-expansive and $(\alpha_2, \beta_2, \epsilon_2)$-expansive respectively such that

1. Both components are expansive i.e. $\alpha_1, \beta_1, \alpha_2, \beta_2 > \tau$
2. The union of the components $\mathcal{K}_{12} = \mathcal{K}_1 \cup \mathcal{K}_2$ is less expansive, i.e. $\alpha_{12}, \beta_{12} \leq \tau$.

Q 2. Can we algorithmically compute bottleneck points? Given a candidate, can we judge its utility?

It is indeed very computationally expensive to ascertain whether a given point is a bottleneck. This involves determining the two expansive components, which in theory could be as difficult as finding a path joining start and goal.

Even if we were given a candidate point, it might be the case that path joining the start and goal query does not pass through the bottleneck. There would be no way of knowing this apriori to planning. The only way to judge the utility would be to plan with the point and evaluate if indeed it helped reduce planning time.

Q 3. Can we use statistical techniques to obtain these bottleneck points?

Since we know that bottleneck points are locations through which paths joining start and goal queries must pass through always, this certainly points to the fact that we can use clustering techniques to determine suitable bottleneck points. We can collect a database of problems, solve these problems offline, collect the solutions and cluster the configurations. To generalize to different problems, we can consider extracting some local context about the point and learning a model. This can then be used in test time to sweep over the space and generate these points.

We can also use domain knowledge to create a library of planning strategies, each using different bottleneck points. We can then look to learn a mapping from planning problem to strategy.

3.2 The optimal path planning problem

We now define the *optimal path planning problem* that we wish to solve as an interim step to addressing the kinodynamic motion planning problem (Problem 1). The overall aim is to compute a path from start to goal that minimizes an objective while remaining in collision free space and satisfying bounds on its derivatives.

Let $X \subseteq \mathbb{R}^n$ be the state space of the planning problem, $\mathcal{O} \subset X$ be the states in collision with obstacles, and $X_{\text{free}} = X \setminus \mathcal{O}$ be the resulting set of permissible states. Let $\mathbf{x}_{\text{start}} \in X_{\text{free}}$ be the initial state and $X_{\text{goal}} \subset X_{\text{free}}$ be the set of desired goal states. Let $\xi : [0,1] \to X$ be a path which is defined as a function that maps an index $\tau \in [0,1]$ to a state $\mathbf{x} \in X$. A set of inequality constraints is imposed on the path and its derivatives $q(\xi) \leq 0$. Hence, we are restricted to search in the space of feasible paths denoted by Ξ.

Problem 2 (Optimal Path Planning)**.** The optimal planning problem is then formally defined as the search for the path, ξ^*, that minimizes a given cost function, $c : \Xi \to \mathbb{R}_{\geq 0}$, while connecting $\mathbf{x}_{\text{start}}$ to $\mathbf{x}_{\text{goal}} \in X_{\text{goal}}$ through free space,

$$\xi^* = \underset{\xi \in \Xi}{\arg\min} \{ c(\xi) \mid \xi(0) = \mathbf{x}_{\text{start}}, \xi(1) \in X_{\text{goal}}, \forall \tau \in [0,1], \xi(\tau) \in X_{\text{free}} \}, \tag{3.2}$$

where $\mathbb{R}_{\geq 0}$ is the set of non-negative real numbers.

Note that the efficacy of the design decision to reduce the original kinodynamic motion planning problem in Problem 1 to the path planning problem in Problem 2 is dependent on the post-processing module that will assign a time profile to the path followed by possible filtering / smoothing steps. Moreover, there maybe several such reductions that work well. In the interest of clarity, we defer reasoning about such choices to a later time in Chapter 4 and shift our focus on techniques to effectively solve optimal path planning problems once they have been defined.

3.3 A framework for systematically assembling planning algorithms

Having defined the path planning problem in Problem 2, we wish to systematically reason about choosing a planning algorithm that is likely to meet performance requirements. The space of candidate algorithms is very large and intractable to search over. Even if one were to choose between two candidate planning algorithms, there isn't a clear one to one mapping to allow a fair comparison. The lack of such a mapping stems from the inherent hybrid nature of planning algorithms - they consist of algorithmic operations interleaved with domain dependent decision making.

A comprehensive framework for reasoning about a vast number of incremental planning algorithms was proposed by LaValle [2006] (Section 14.3.4). While such a framework is indeed

very expressive, we follow a simpler framework prescribed by Pearl [1984] (Section 2.3.1). Broadly speaking, a planning algorithm is an operation that converts an *implicit state-space graph* to an *explicit search tree*. We define these two components as follows:

Definition 3.4 (Implicit State-Space Graph)**.** Let $\mathcal{G} = \langle X_{\text{samples}}, \texttt{Near}, \texttt{Steer} \rangle$ be an implicit state-space graph which is represented as a tuple with the following elements:

> *Candidate States*: A set of candidate states $X_{\text{samples}} \subset X$ which serve as the vertices of the graph.
>
> *Near Function*: A function $\texttt{Near}(\mathbf{v})$ that returns the set of near states $X_{\text{succ}} \subseteq X_{\text{samples}}$ given a vertex $\mathbf{v}$. Note that this function can be an algorithmic operation that can take the state of the search as input to allow more flexibility. This function can also specify if successor or predecessor states thus expressing a directed nature of the graph.
>
> *Steer Function*: A steering function $\texttt{Steer}(\mathbf{x}, \mathbf{y})$ returns a dynamically feasible path ξ joining $\mathbf{x}$ and $\mathbf{y}$. For instances where such a boundary value problem (BVP) cannot be efficiently solved, this framework is not suitable and the reader is referred to approaches such as Hsu et al. [2002], Li et al. [2015], Paden and Frazzoli [2016].

Definition 3.5 (Explicit Search Tree)**.** We define $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ to be an *explicit* tree with a set of vertices, $\mathcal{V} \subset X_{\text{free}}$, and edges, $\mathcal{E} = \{(\mathbf{v}, \mathbf{w})\}$ for some $\mathbf{v}$, $\mathbf{w} \in \mathcal{V}$. Given a tree $\mathcal{T}$, the path from start state $\mathbf{x}_{\text{start}}$ to a vertex $\mathbf{v}$ is given by $\xi_{\mathcal{T}}(\mathbf{v})$. Hence the cost of a solution computed by the search procedure is $c(\xi_{\mathcal{T}}(\mathbf{x}_{\text{goal}}))$.

We now have the required components to define an incremental planning algorithms as follows:

Definition 3.6 (Incremental Planning Algorithms)**.** An incremental planning algorithm at iteration k interleaves the following two steps

1. **Incremental Search**
$$\mathcal{T}^{k+1} = \texttt{Search}\left(\mathcal{G}^k, \mathcal{T}^k\right)$$
2. **Incremental Graph Update**
$$\mathcal{G}^{k+1} = \texttt{Update}\left(\mathcal{G}^k, \mathcal{T}^{k+1}\right)$$

Thus an incremental planning algorithm traces out the following sequence over iterations

$$\left(\mathcal{T}^0, \mathcal{G}^0\right) \rightarrow \mathcal{T}^1 \rightarrow \mathcal{G}^1 \rightarrow \ldots \mathcal{G}^{b-1} \rightarrow \mathcal{T}^b \tag{3.3}$$

Having defined these components, we can now begin to have a systematic approach to assembling planning algorithms by selection of an implicit state-space graph $\mathcal{G}$ and a search procedure `Search`. We present examples from planning literature for both options, highlighting their applicability, and show how several state of the art planning algorithms can be viewed from this perspective.

3.3.1 Examples of implicit state-space graph

Example 3.1 (Random Geometric Graphs (RGG))**.** Randomly sampling states $\mathbf{x}$ uniformly from the state space X, first popularized in motion planning by LaValle [1998], has retained its importance over the years primarily due to its implementation ease. A randomly drawn sequence enjoys the property of being *probably dense*, which allowed sampling based approaches such as RRT to eventual discover feasible solutions to hard high dimensional problems. Its recent surge in popularity, however, can be attributed to the connections made to random geometric graphs (RGG) (Penrose [2003]) in seminal work by Karaman and Frazzoli [2010, 2011]. We first define what we mean by an RGG in this context

X_{samples} : States are sampled uniformly randomly $\mathbf{x} \sim \mathcal{U}(x)$

`Near` : The `Near` $(\mathbf{v})$ returns states from X_{samples} depending on their *geometric* location with respect to $\mathbf{v}$. There are two categories - either the functions returns its nearest neighbours (a k-nearest graph [Xue and Kumar, 2004]), or all neighbours within a specified distance (a r-disc graph [Gilbert, 1961]).

`Steer` : The steer function `Steer` is required to solve a BVP online (due to states being randomly drawn).

RGG theory provides probabilistic relations between the number and distribution of samples X_{samples}, the parameters of `Near` and asymptotic connectivity properties [Muthukrishnan and Pandurangan, 2005, Penrose, 2003]). Karaman and Frazzoli [2011] showed conditions under which such graph asymptotically capture the optimal solution.

While the asymptotic properties are attractive, one may wonder if these graphs contain acceptable solutions if executed for a finite time. This is indeed the case when the free space X_{free} is well connected. In such situations, the connectivity nature of RGG results in graphs that are able to successfully "shortcut" through free space. Even in cases where only certain sections of X_{free} enjoy connectivity, biasing the sampling distribution to contain vertices in such areas allow RGGs to be very effective in practice and express paths relatively free of discretization artifacts. In general, finite time performance of such graphs is a topic of active research [Dobson and Bekris, 2013]. ■

Example 3.2 (Low Dispersion / Discrepancy Graphs)**.** An alternative to random sampling is to optimize a criteria called *dispersion* [Niederreiter, 1978]. A low dispersion sampling implies a guarantee on the coverage of the state space X. Dispersion generalizes the notion of grid resolution. An incrementally updating low dispersion graph can be interpreted as an incrementally densifying grid. The components are :

X_{samples} : States are created by discretizing each dimension of X.

`Near` : The `Near` $(\mathbf{v})$ returns adjacent states to $\mathbf{v}$ which is well defined given the regular nature of the graph. The connectivity can be manually chosen to define the expressiveness of the graph.

`Steer` : Since the graph has spatial symmetry, the BVPs to compute dynamically feasible edges can be computed as offline primitives.

A Sukharev Grid [Sukharev, 1971] or a state lattice [Barraquand and Latombe, 1991] are examples of low dispersion graphs. Such graphs work well in environments where obstacles are spread out in the space. Such configurations reduce the usefulness of RGG which can no longer "shortcut" through free space effectively. However, low discrepancy methods have well spread out points making local connections, thus allowing them to have better connectivity.

In some situations, having axis aligned grids can hurt - for instance, a gap in a wall of obstacle might not have an edge present in it. Discrepancy becomes a metric of choice in such situations [Niederreiter, 1978]. Dispersion serves as a lower bound for discrepancy. The Halton sequence [Halton, 1960] is an example of a low discrepancy sampling. Janson et al. [2015a] showed properties that the `Near` must satisfy to provide optimality guarantees in finite time.

While low dispersion sampling methods are superior to sampling from uniform distributions, they make an assumption that the planning space is an unit cube. Achieving this effect over arbitrary volumes is hard, hence making these techniques difficult to use as is. ■

The previous two examples are fairly problem agnostic. We now take a look at a more domain specific family of implicit state-space graphs.

Example 3.3 (Geodesic Approximating Graphs)**.** The ideal implicit graph would be one that captures the optimal path and very little else such that a search procedure might efficiently uncover it. For certain problems, such as 2D holonomic path planning with polygonal obstacles, the visibility graph would be such a graph. In general, we would like the implicit graph to approximate the geodesic, i.e., the optimal path between our query points is likely to lie on it. This is akin to "biasing" sampling to pick good states and edges that have some likelihood of advancing search. The components of such a graph can be described in a general sense as follows

X_{samples} : A set of states that are likely candidates for being part of the optimal path (such as lying tangential to the surface of obstacles)

`Near` : This function returns candidates that are likely to be connected if they are to be part of an optimal path. For example, in the case of visibility graphs, edges which are not tangential cannot be part of the optimal path and hence will not be contained in the graph.

`Steer` : This can remain standard such as using a BVP solver.

Approaches that bias sampling in "promising areas" [Diankov and Kuffner, 2007, Kiesel et al., 2012, Palmieri et al., 2016] fall into this category. Hence this is a useful tool to exploit problem structure. Such implicit graph tend to have a high variance in their performance - on problems where the assumptions are met, they produce high quality solutions in real-time, however may fail catastrophically in situations where they are not met.

■

3.3.2 Examples of `Search` procedures

Example 3.4 (Incremental Informed Search)**.** Pearl [1984] (Section 2.3.1) describes a basic structure for informed search. An informed search is a systematic procedure for visiting vertices in the implicit graph in order of promising nature, and assigning them a parent each. A slight

modification to this framework allows this procedure to be incremental as described in Algorithm 1.

Algorithm 1: `InformedSearch` $(\mathcal{G}, \mathcal{T})$

```
1  Q ← T;
2  while Q ≠ ∅ do
3      v_m ← BestInQueue(Q), Q ←⁻ v_m;
4      if v_m = x_goal then
5          break;
6      for w_m ∈ Near(v_m, T ∪ G) do
7          if (v_m, w_m) ∉ T then
8              if g_T(v_m) + c((v_m, w_m)) < g_T(w_m) then
9                  T ←⁻ {(v, w_m) ∈ T};
10                 T ←⁺ (v_m, w_m);
11                 if w_m ∉ Q then
12                     Q ←⁺ w_m;
13 return T;
```

The algorithm proceeds as follows. The current search tree is entered in a priority queue (Line 1). Then the following process loops till the queue is empty. The most promising vertex in the queue is popped (Line 3). As long as this is not the goal vertex (Line 4), potential outgoing connections are considered (Line 6) and optimal parent assigned (Line 10). The algorithm enjoys nice guarantees such as optimality as long as the queue sorting condition satisfies certain properties. Incremental informed search can be implemented efficiently under the framework of life long planning [Koenig et al., 2004]. Lazy edge evaluations also improve performance [Cohen et al., 2015, Dellin, 2016, Gammell et al., 2015]).

This search method can be used to exploit problem structure by changing the heuristic that drives the search in Line 3. For a full exposition on heuristics, their nature and how they can be designed for a problem, refer to Pearl [1984]. ■

Example 3.5 (Local Policy Iteration)**.** While the systematic search in Algorithm 1 enjoys properties such as optimality, it requires re-evaluating all vertices that are likely to have a better path from the start at each iteration. This can scale as the number of vertices increase. A thing to note is that since the `Search` procedure isn't required to have the optimal solution with respect to the graph it has seen at an interim stage. A procedure that asymptotically approaches this would suffice.

Local policy iteration is one such procedure. Whenever this incremental `Search` procedure is invoked, it examines new samples added to the implicit graph and updates the search tree directly affected by these samples. The observation that this procedure is a local policy iteration was made by Arslan and Tsiotras [2016].

Algorithm 2 describes the procedure. The new sample is obtained (Line 1) and assigned the locally best parent (Line 2-4). The local vertices in the search tree are re-wired subsequently (Line 5-8).

Algorithm 2: `LocalPolicyIteration` $(\mathcal{G}, \mathcal{T})$

$\mathbf{x}_{\mathrm{m}} \leftarrow$ `GetSample` $(\mathcal{G})$;
$X_{\mathrm{near}} \leftarrow$ `Near` $(\mathbf{x}_{\mathrm{m}}, \mathcal{T})$;
$\mathbf{v}_{\mathrm{m}} \leftarrow \underset{\mathbf{v} \in X_{\mathrm{near}}}{\arg\min}\, g_{\mathcal{T}}(\mathbf{v}) + c((\mathbf{v}, \mathbf{x}_{\mathrm{m}}))$;
$\mathcal{T} \xleftarrow{+} (\mathbf{v}_{\mathrm{m}}, \mathbf{x}_{\mathrm{m}})$;
for $\mathbf{v} \in X_{\mathrm{near}}$ **do**
  if $g_{\mathcal{T}}(\mathbf{x}_{\mathrm{m}}) + c((\mathbf{x}_{\mathrm{m}}, \mathbf{v})) < g_{\mathcal{T}}(\mathbf{v})$ **then**
    $\mathcal{T} \xleftarrow{-} \{(\mathbf{x}, \mathbf{v}) \in \mathcal{T}\}$;
    $\mathcal{T} \xleftarrow{+} (\mathbf{x}_{\mathrm{m}}, \mathbf{v})$;
return $\mathcal{T}$;

A variant of the above procedure is to update the values of the entire search tree. This would be performing value iteration. Arslan and Tsiotras [2013] compute this via Gauss-Seidel step. This search procedure also allows the flexibility of choosing if a sample is "promising" enough to perform a local update [Arslan and Tsiotras, 2015]. ■

3.3.3 Assembling planning algorithms

We will now show how various implicit state graphs $\mathcal{G}$, and incremental search functions `Search` can be combined to produce planning algorithms. Table 3.1 shows a handful of planning algorithms from recent literature as a sum of such components.

We present 3 specific algorithms in Section 3.4, 3.5 and 3.6 that assemble planning algorithms to exploit a particular structure.

Table 3.1: Planning algorithms as sum of components

Planning Algorithm	**Implicit Graph ($\mathcal{G}$)**	**Incremental Search (`Search`)**
RRT* [Karaman and Frazzoli, 2010]	Random geometric graphs	Local policy iteration
RRT# [Arslan and Tsiotras, 2013]	Random geometric graphs	Local policy iteration with Gauss-Seidel relaxation
InformedRRT* [Gammell et al., 2014]	Random geometric graphs with informed samples	Local policy iteration
FMT* [Janson et al., 2015b]	Random geometric graphs	Single-pass informed search
BIT* [Gammell et al., 2015]	Random geometric graphs with informed samples	Incremental informed search with lazy edge evaluation
gPRM [Janson et al., 2015a]	Low discrepancy sampling using Halton sequences	Single-pass informed search
hRRT [Urmson and Simmons, 2003]	Random graph biased around promising leafs of search tree	Local policy iteration
RA* [Diankov and Kuffner, 2007] / SBA* [Persson and Sharf, 2014]	Random graph biased around promising leafs of search tree	Incremental informed search
LBT-RRT [Salzman and Halperin, 2015]	Random geometric graph	Local policy iteration with edges filtering criteria
State Lattice [Pivtoraiko et al., 2009]	Multi-resolution lattice with feasible motion primitives	Incremental informed search with down-sizing heuristic

3.4 Example algorithm 1: Hybrid local global search (RABIT*)

Consider the application where a helicopter is required to fly through a mountainous terrain as shown in Fig. 3.4(a). In addition to this, there are no fly zones that may dynamically appear to designate the presence of radio towers or power-lines that the system must also stay clear of. On examining the corresponding path planning problem, shown in Fig. 3.4(b), we see the presence of "difficult-to-sample" homotopy classes. On the other hand, the solution to such a problem may not be uncovered by a pure local search approach.

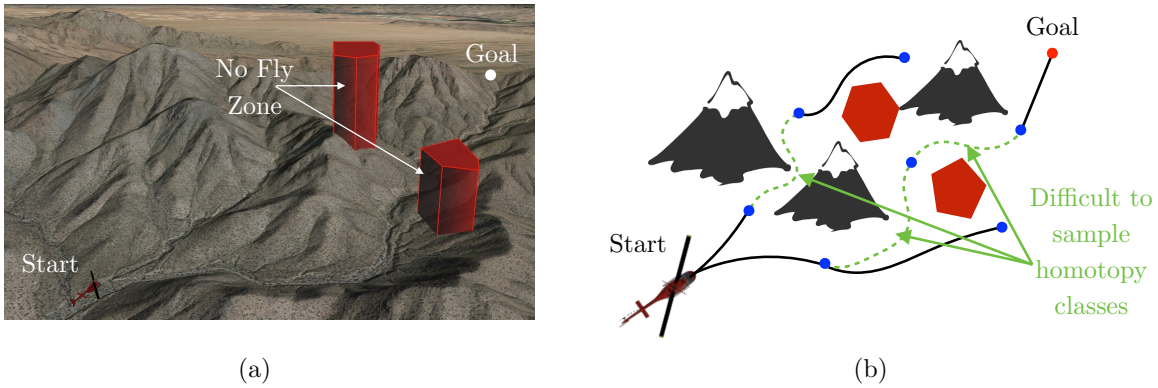


Figure 3.4: An instance of planning problems containing difficult to sample homotopy classes (a) A helicopter flying in the mountains with no fly zones appearing dynamically (b) The corresponding planning problem shows that an effective implicit graph for this problem requires edges to "bend" around obstacles. This is because these regions are difficult to sample to discover a collision free edge.

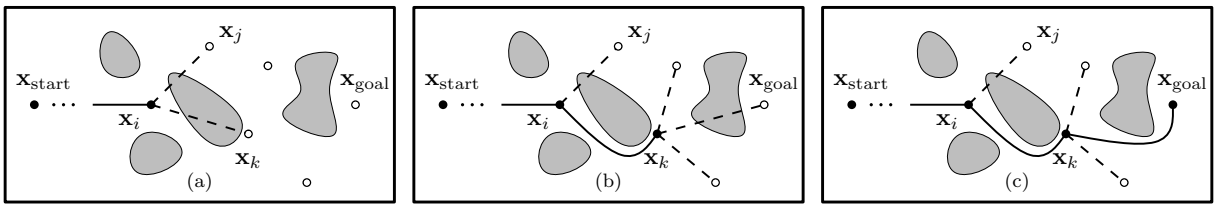


Figure 3.5: An illustration of how the RABIT* algorithm uses a local optimizer to exploit obstacle information and improve a global search. (a) The informed search evaluates the potential edge from $\mathbf{x}_i$ to $\mathbf{x}_k$ first as it could provide a better solution than an edge from $\mathbf{x}_i$ to $\mathbf{x}_j$. (b) The search optionally invokes the local optimizer in the `Steer` $(\mathbf{x}_i, \mathbf{x}_k)$. (c) This process repeats to create a hybrid search tree where some of the edges are locally optimized.

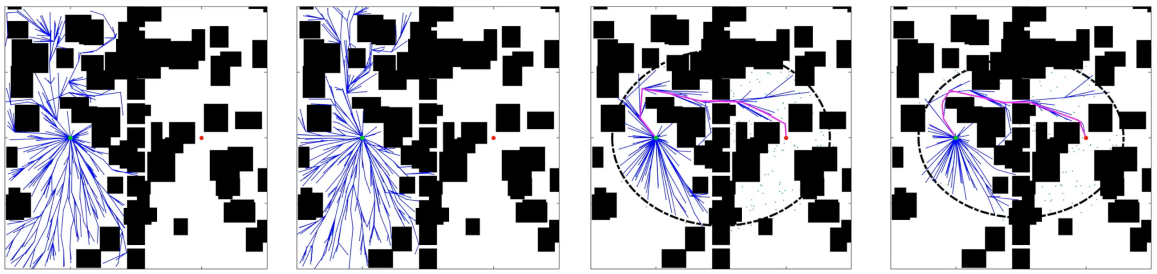
We present a hybrid planning algorithm, Regionally Accelerated BIT* (RABIT*) [Choudhury et al., 2016a], that integrates the benefits of both methods into a single search. A key insight is that applying local optimization to a subset of edges likely to improve the solution avoids the prohibitive cost of optimizing every edge in a global search. This is made possible by Batch Informed Trees (BIT*) [Gammell et al., 2015], an informed global technique that orders its search by potential solution quality. In our algorithm, we extend BIT* by using optimization to exploit local domain information and find alternative connections for edges in collision and accelerate the search. This improves search performance in problems with difficult-to-sample

homotopy classes (e.g., narrow passages) while maintaining almost-sure asymptotic convergence to the global optimum.

RABIT* can be understood from the viewpoint of our framework as follows.

Implicit Graph ($\mathcal{G}$): The underlying implicit graph is an informed random geometric graph (as used in BIT* [Gammell et al., 2015]) with one critical modification. The steer function, `Steer`$(\mathbf{x}, \mathbf{y})$ can optionally invoke a local optimization method - in this case CHOMP [Ratliff et al., 2009a].

Incremental Search (`Search`): The search method is an informed incremental search using lazy edge evaluation (as used in BIT*). The algorithm uses a heuristic rule to detect if an edge is in a potential bad local minima to reason about whether to invoke the optimization option in the steer function as shown in Fig. 3.5.



RRT* Cost: Inf (a) | Informed RRT* Cost: Inf (b) | BIT* Cost: 1.67 (c) | RABIT* Cost: 1.57 (d)

Figure 3.6: The results after 0.2 seconds of RRT*, Informed RRT*, BIT*, and RABIT* on a random $\mathbb{R}^2$ world. Note how RABIT* uses local information to find a path through the optimum narrow gap with fewer samples than BIT*, as well as how the path bends around obstacles.

To systemically evaluate RABIT*, it was run on randomly generated worlds in $\mathbb{R}^2$ (shown in Fig. 3.6) and $\mathbb{R}^8$ (shown in Fig. 3.7(a),(b)). The worlds consisted of a (hyper)cube of width 2 divided in half by a wall with 10 narrow gaps. The world also contained random axis-aligned (hyper)rectangular obstacles such that at most one third of the environment was obstructed. This allowed us to randomly generate challenging planning problems that had an optimal solution passing through a difficult-to-sample narrow passage. RABIT* was compared to publicly available Open Motion Planning Library (OMPL) implementations of RRT, RRT-Connect, RRT*, Informed RRT*, and BIT*. To quantify the results, we calculate the time for each algorithm to reach 90% of its final value. In $\mathbb{R}^2$, RABIT* takes 0.215s compared to BIT*'s 0.179s. In $\mathbb{R}^8$, RABIT* takes 0.262s compared to BIT*'s 0.471s. These results show that in $\mathbb{R}^2$ RABIT* performs similarly to BIT*; however, that in $\mathbb{R}^8$ RABIT* finds better solutions faster on these problems with difficult-to-sample homotopy classes.

To evaluate RABIT* on real planning problems, it was run on a recorded flight mission of an autonomous helicopter. The autonomous helicopter operates at speeds of up to 50 m/s in challenging environments that may contain difficult-to-sample features such as valleys. Plans

must obey the dynamic and power constraints of the vehicle (including climb-rate limits) and completely avoid obstacles, a planning problem that is difficult to solve in *real-time*.

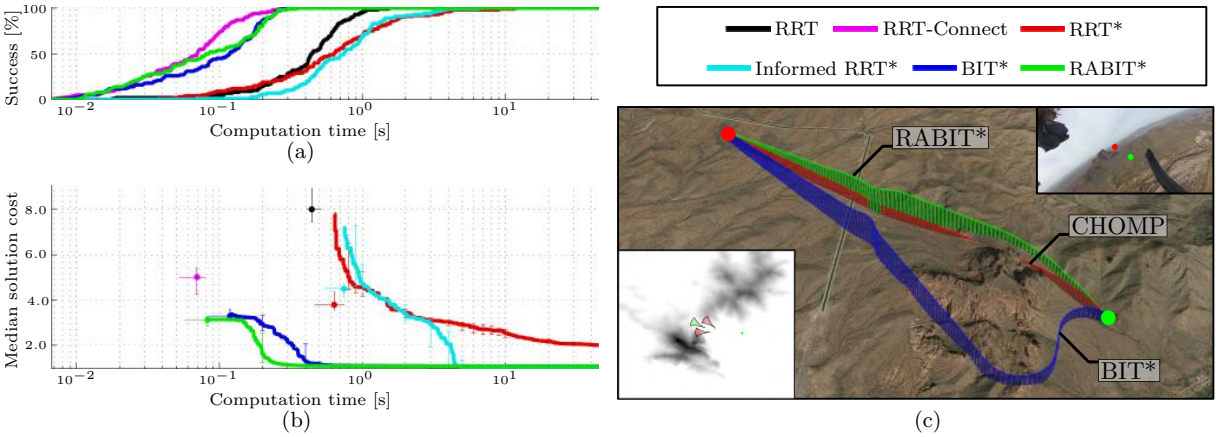


Figure 3.7: The results from representative worlds in $\mathbb{R}^8$ and on real data from autonomous helicopter experiments show the efficacy of RABIT* in converging quickly to a high quality solution. For the chosen random worlds, (a) shows the percentage of the 100 trials solved versus run time, while (b) shows the median solution cost versus run time. (c) The results of a planning problem encountered during flight tests of an autonomous helicopter in Mesa, Arizona from a start (green dot) to a goal located 4.91 kilometres away (red dot). Examples of inevitable collision states (red cones) and useful states (green cones) are shown on evidence map to illustrate the difficulty of navigating the through the valley. RABIT* is able to use edges proposed by a local optimizer to find a path through the valley (green, 5.0 kilometres), which BIT* is unable to navigate the valley (blue, 6.17 kilometres) and CHOMP finds an infeasible local optima.

Sensor data collected from test flights over Mesa, Arizona were used to propose a planning problem around mountains (Fig. 3.7(c)). This problem is challenging because the helicopter's constraints create a large number of states from which no collision-free path can be found (inevitable collision states). These states reduce the connectivity of the free space and increase the inherently difficult problem of sampling a valid path through the narrow valley.

This problem was used to compare the ability of BIT*, CHOMP, and RABIT* to plan for a vehicle with restrictive constraints given limited computation time (2 seconds). These results demonstrate how RABIT* combines the benefits of global and local techniques. CHOMP uses cost-gradient information but can become stuck in poor local minima when optimizing long paths, failing to find a feasible solution. BIT* almost-surely converges asymptotically to the global optimum but has difficulty sampling the valley in the available time, finding a path that goes around the mountain (6.17 kilometres). RABIT* uses the local optimizer on short paths to help find narrow passages and a global search to avoid infeasible local minima, finding a path through the valley (5.0 kilometres).

Hybrid search can have much greater benefit for problems other than geometric planning, such as planning on vector fields [Pereira et al., 2016].

3.5 Example algorithm 2: 3D sparse visibility graph (SPARTAN-Lite)

Consider the scenario of a micro aerial vehicle (MAV) flying in outdoor unstructured environment as shown in Fig. 3.8. Such environments have obstacle clusters, such as trees or buildings that the robot is required to fly around without significant deviations. It is desirable for the optimal path to "bend" smoothly around obstacles as it joins start and goal states.

Figure 3.8: Planning problems encountered by a micro aerial vehicle (MAV) flying outdoors (a) The MAV is expected to fly around trees that occur in clusters (b) A close up of the MAV avoiding a tree (c) The robot's perspective shows a cluster of obstacles. It creates a sparse graph (blue) that bends around obstacles and contains a high quality solution (green) that can be uncovered easily

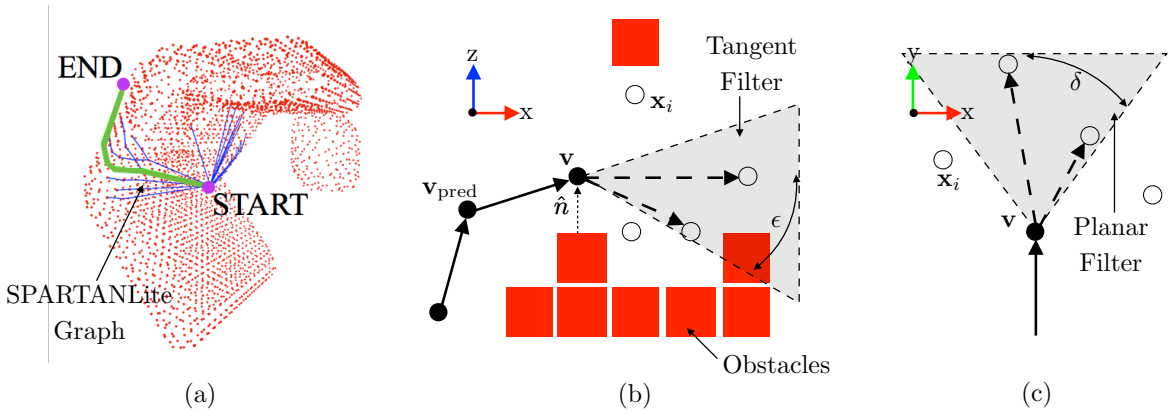
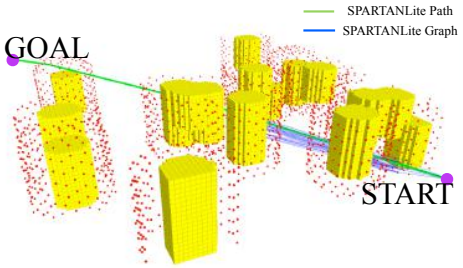


Figure 3.9: Illustration of SPARTAN-Lite algorithm (a) An example of the sparse tangential network created by SPARTAN-Lite. Only the incoming edges for expanded vertices are shown in blue. (b) The tangent filter projects edges on the plane containing the normal to the obstacle surface. It only allows outgoing edges that bend towards obstacle upto a threshold. (c) The planar filter projects edges on the plane perpendicular to the normal and allows outgoing edges that have a bounded deviation

We present an approach, SPARTAN-Lite [Nuske et al., 2015], to approximate 3D visibility graphs which are NP-Hard in general [Canny, 1988b]. Leveraging incremental approaches to calculate distance fields from occupancy grids [Lau et al., 2010], we spawn vertices on a manifold on the surface of obstacles [Cover et al., 2013]. Exploiting properties of geodesics on smooth manifolds, only edges that are likely to be part of the optimal path are allowed creating a sparse

tangential network as shown in Fig. 3.9(a).



(a)

Algorithm	Success	Average Cost	Average Time (s)
SPARTANLite	100%	2878(±950)	0.071(±0.087)
RRT*	76.72%	3472(±1240)	0.102(±0.003)

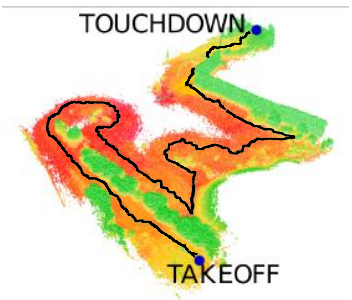
(b)

Figure 3.10: SPARTAN-Lite outperforms RRT* (a) A simulated forest with random trees where the robot has to plan around obstacles. The SPARTAN-Lite graph is shown (b) Computation times and solution costs for both algorithms

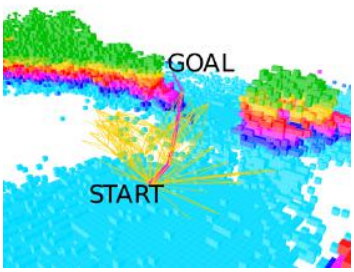
SPARTAN-Lite can be understood from our framework as follows:

Implicit Graph ($\mathcal{G}$): The underlying implicit graph is an approximate visibility graph. The set of vertices are spawned on the surface of obstacles during the expansion step of incremental distance transforms operating on the occupancy grid. The successor function `Near` ($\mathbf{v}$) returns vertices such that the joining edge is likely to be part of the optimal path (i.e. be a geodesic). Hence a set of filters are applied to remove edges from the otherwise fully connected graph. The first such filter is a tangent filter that only allows edges that bend around obstacles while being tangential as shown in Fig. 3.9(b). The second filter is a spatial filter that restricts the deviation between incoming and outgoing edges passing through a vertex.

Incremental Search (`Search`): The search method is a single pass informed search.



(a)



(b)

Statistic	Value
Planning Time (s)	0.175(±0.199)
Vertex Computation (s)	0.785(±0.661)
Total Distance (m)	1744.8
Speed (m/s)	1.395(±0.431)

(c)

Figure 3.11: (a)An autonomous outdoor flight stress testing the motion planner. The robot explored an area of 1.7 km from takeoff to touchdown. SPARTAN-Lite was able to produce collision free paths while the system flew at an average speed of 1.4 m/s (b) A particular planning instance during the autonomous stress test run. SPARTAN-Lite plans through a clearing in the trees. The search graph is relatively sparse compared to the number of potential edges. (c) Statistics from the run

Proofs and implementation details can be found in Nuske et al. [2015]. We evaluated SPARTAN-Lite in a random world to emulate a forest like environment as shown in Fig. 3.10. SPARTAN-Lite outperforms RRT* due to its sparse graphs. We also stress tested SPARTAN-Lite in real outdoor experiments where it kept the robot collision free the entire time as shown in Fig. 3.11.

3.6 Example algorithm 3: Reachability informed random graph (RRT*-AR)

Consider the scenario of motion planning for an autonomous helicopter whose engine has failed as shown in Fig. 3.12. The helicopter enters a phase known as autorotation where it has to fly along a profile that preserves its kinetic energy which can be traded off near the ground to cushion its landing. In this phase, the reachability of the vehicle is severely restrictive and filters out the set of allowable edges that can exist. A search approach that is agnostic to this structure would waste a considerable amount of times considering edges that are not likely to be dynamically feasible.

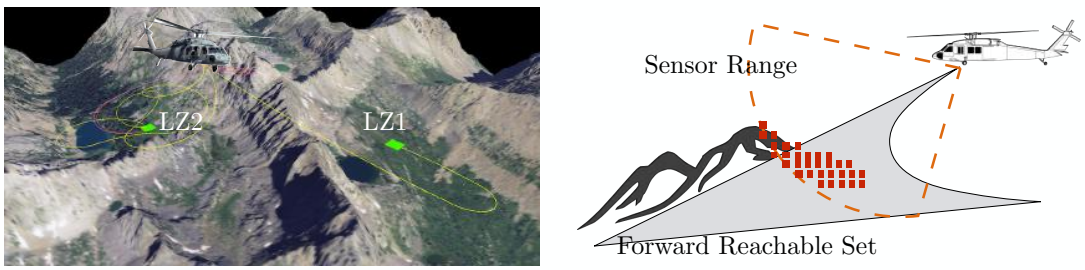


Figure 3.12: Planning problems encountered by an autonomous helicopter whose engine has failed and must land safely (a) Since the helicopter cannot hover in place, it must immediately plan to land. As the world is only partially known, it must plan for contingencies simultaneously (b) The problem is hard given the limited range of the sensor and the restriction imposed by the reachability volume of the vehicle

We present an algorithm, RRT*-AR [Choudhury et al., 2013, 2014] that conducts a random graph search with modifications to ensure plans are found quickly to multiple goal points. We wish to highlight one key aspect of this algorithm - the modification to the nearest neighbour function. The function is "informed" by the reachability of the vehicle to return states that are likely to be connected by the BVP. Note that this is merely empirical - for a theoretical treatment of the issue, refer to work done by Karaman and Frazzoli [2013].

RRT*-AR can be understood from our framework as follows

Implicit Graph ($\mathcal{G}$): The underlying implicit graph is a random geometric graph. The `Near` ($\mathbf{v}$) is modified to return a different set for predecessors or successors as illustrated in Fig. 3.13(b),(c). For predecessors, it "shifts" the NN ball in the direction of the backward reachability set X_{BRS} to return the set $X_{\text{near}}^{\text{back}}$. Similarly, for successors, it "shifts" the NN ball in the direction of the forward reachability set X_{FRS} to return the set $X_{\text{near}}^{\text{front}}$.

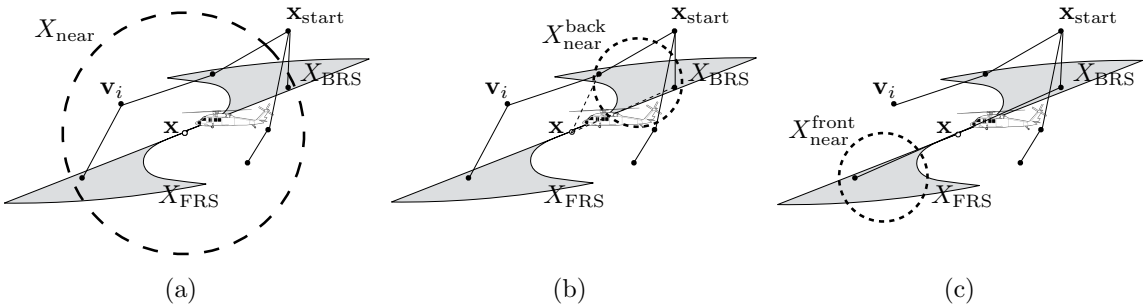
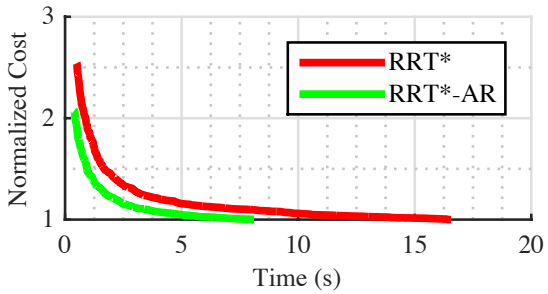


Figure 3.13: Illustration of the `Near` $(\mathbf{v})$ of RRT*-AR. (a) The `Near` $(\mathbf{v})$ for a nominal RGG performs a symmetric lookup (b) In RRT*-AR, while computing the set of predecessors, the NN ball is shifted to cover a larger area of the backward reachable set (c) Similarly, while computing successors, the NN ball is shifted towards the forward reachable set.

Incremental Search (`Search`): The search method is an informed incremental search using lazy edge evaluation (as used in BIT* [Gammell et al., 2015]). This is the local policy iteration, with speedups enabled by using approximations.

For proofs and implementation details, refer to Choudhury et al. [2013]. RRT*-AR was evaluated on a synthetic dataset of planning problems by simulating engine failure in a mountainous region to create 3318 problems. It was compared to RRT* algorithm which does not modify the `Near` $(\mathbf{v})$ function. Fig. 3.14(a) shows RRT*-AR is able to compute the initial path faster and have faster convergence. Fig. 3.14(b) compares the solution quality of the two algorithms at different times during the search, thus showing the efficiency of RRT*-AR.



Algorithm	Cost $t = 0.6s$	Cost $t = 1.0s$	Time to **cost** $= 1.1$
RRT*	1.97($\pm$1.28)	1.60($\pm$0.86)	4.40($\pm$1.28)
RRT*-AR	1.65($\pm$1.03)	1.37($\pm$0.61)	2.62($\pm$1.07)

(b)

Figure 3.14: Comparison of RRT*-AR with RRT* on a dataset of 3318 challenging problems. (a) Plot of the mean normalized solution cost of the two algorithms (b) A table of the mean and standard deviation of normalized cost at key instances during the search

3.7 Theoretical limits of planning resolution

We briefly summarize our work on understanding the theoretical Limits of speed and resolution for kinodynamic planning in a Poisson forest originally presented in Choudhury et al. [2015b] with further derivations in Choudhury [2015].

There has been an extensive amount of work in developing planning algorithms that perform a discrete search on a state lattice [Dolgov et al., 2010, Heng et al., 2011, Hwangbo et al., 2007, Likhachev and Ferguson, 2009, Lindemann and LaValle, 2006, MacAllister et al., 2013, Pivtoraiko et al., 2009]. The success of planning algorithms in practice depends on two key factors. Firstly, what is the *maximum speed* at which a collision free trajectory is guaranteed to exist with a high probability? If the robot violates this speed limit, there is a non-zero probability that it will encounter situations where a collision free trajectory does not exist, irrespective of the planning algorithm being used. Secondly, if the robot is operating below the speed limit, what is the *minimum planning resolution* of an algorithm such that it can compute a collision free trajectory? This controls the trade-off between time complexity and the success of the planning algorithm. The answer to both of these questions is fundamentally linked to the dynamics of the robot and the density of obstacles in the environment.

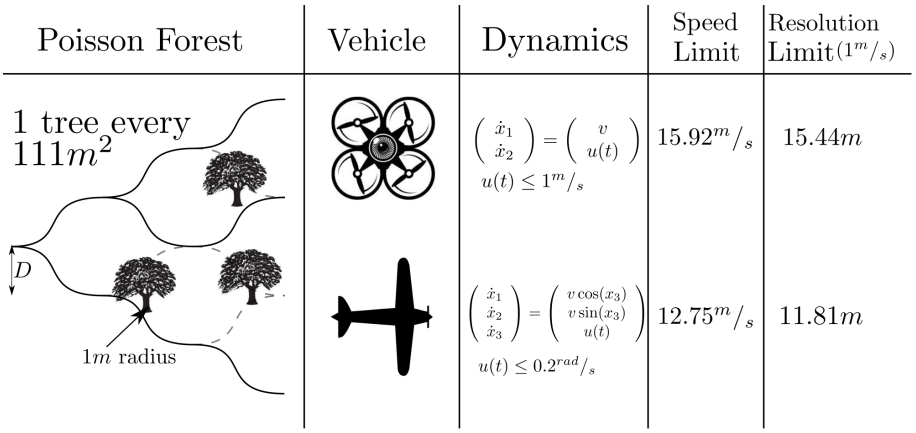


Figure 3.15: Robot dynamics affect maximum speed and minimum planning resolution. The less conservative dynamics of the quadrotor allow for a higher speed as well as planning with a coarser resolution while still ensuring collision free motion.

Karaman and Frazzoli [2012a] solved the problem of the theoretical speed limit of a system with single integrator dynamics flying in a Poisson forest. They show that the system undergoes a *phase transition* - that above a critical speed there exists no infinite collision-free trajectory with probability 1, while below this speed a path exists almost surely. By mapping the lattice obtained from the single integrator dynamics to a regular lattice, they use known results from discrete percolation theory [Bollobas and Riordan, 2006, Grimmett, 1999, Sahini and Sahimi, 1994] to compute bounds on the speed. They also show that the success of a planning algorithm for a system with nonlinear dynamics undergoes a phase transition using arguments based on k-independent bond percolation [Karaman and Frazzoli, 2012b]. However, they were unable to show how one may *compute explicit bounds* for such problems because of the lack of usable results in k-independent percolation theory.

In Choudhury et al. [2015b], we analyze both problems under a single framework and provide expressions for the speed limit and resolution limit as a function of robot dynamics and obstacle density. The major milestones of our approach are

1. We show that solutions to both problems are equivalent to solving bond percolation on a *directed asymmetric hexagonal lattice*. Any variation in the problem, robot dynamics or obstacle density corresponds to different edge probabilities on the hexagonal lattice as shown in Fig. 3.16.

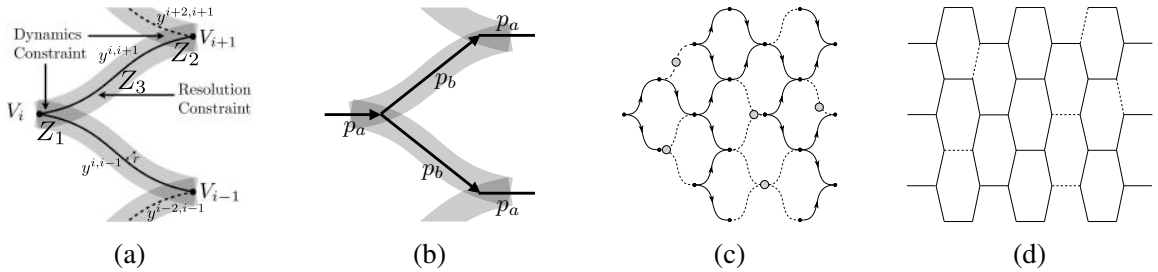


Figure 3.16: Equivalence of state lattice with a directed asymmetric hexagonal lattice. (a) Adjacent vertices in a state lattice along with the swaths of edges. Different zones correspond to regions of overlap of the swath. (b) An equivalent hexagonal lattice which maps zones of the swath to edges on the lattice (c) The open cluster of a state lattice in the presence of a Poisson obstacle field (d) The open cluster of the hexagonal lattice that is equivalent to the state lattice.

Under the assumption of an ergodic distribution of obstacles, the edge probabilities of p_a and p_b are independent and depend on areas Z_1, Z_2, Z_3 as follows

$$\begin{aligned} p_a &= e^{-\rho(Z_1+Z_2)} \\ p_b &= e^{-\rho Z_3} \end{aligned} \tag{3.4}$$

2. Since standard percolation theory lacks analysis of such a lattice, we map the hexagonal lattice to a discrete time Markov chain A_n on the collection of finite subsets of integers. This is illustrated in Fig. 3.17(a). The two parameters in the chain are $0 \leq p \leq q \leq 1$. Given A_n, the events $x \in A_{n+1}$ are conditionally independent and

$$P(x \in A_{n+1}|A_n) = \begin{cases} q & \text{if } |A_n \cap \{x, x+1\}| = 2 \\ p & \text{if } |A_n \cap \{x, x+1\}| = 1 \\ 0 & \text{if } |A_n \cap \{x, x+1\}| = 0 \end{cases} \tag{3.5}$$

We apply the following theorem of the survival of the Markov chain from Liggett [1995].

Theorem 3.4. If the parameters p and q satisfy the inequalities

$$\frac{1}{2} < p \leq 1, \quad 4p(1-p) \leq q \leq 1$$

then the Markov chain A_n satisfies $\mathbb{P}(\forall n : A_n \neq \emptyset) > 0$

3. The result can be applied to derive an upper bound for the critical probability for the directed asymmetric hexagonal lattice bond percolation. Fig. 3.17(b) illustrates the mapping

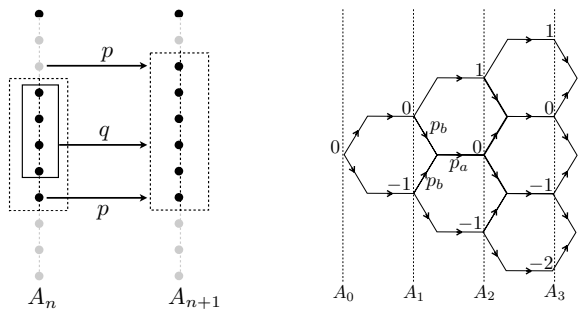


Figure 3.17: (a) Illustration of the Markov chain defined in (3.5) as a collection of subsets of integers on the number line. At time slice n, the set A_n is shown by elements in the rectangle. The transition probabilities of elements with a neighbour in A_n is shown as q while elements with one neighbour is shown as p. (b) The hexagonal lattice is represented as the Markov chain. Each vertex is assigned an integer to map to the integer line. A vertex is contained in A_n if there exists an open path from the origin to it.

of the hexagonal lattice to a Markov chain.

$$\begin{aligned} p &= p_a p_b \\ q &= 2p_a p_b - p_a p_b^2 \end{aligned} \tag{3.6}$$

According to Theorem 3.4, the conditions on p_a and p_b for percolation, $C(p_a, p_b) \leq 0$, are as follows

$$\begin{aligned} 0 \leq p_a &\leq 1 \\ 0 \leq p_b &\leq 1 \\ \frac{1}{2} < p_a p_b &\leq 1 \\ p_a p_b(2 - p_b) &\leq 1 \\ p_b - 4p_a p_b + 2 &\leq 0 \end{aligned} \tag{3.7}$$

As long as the inequalities $C(p_a, p_b) \leq 0$ in (3.7) are true, the lattice will almost surely have an infinite open cluster.

4. As a result, we provide a solution framework that takes as input the robot dynamics and the obstacle density and returns the speed and resolution limit. Our framework is not only able to recover the results in Karaman and Frazzoli [2012a], but provide results for problems which Karaman and Frazzoli [2012b] could not provide explicit answers to.

3.8 Discussion

In this section, we made a step towards addressing Challenge 1 by asking the question - "how can we go about designing path planning algorithms that exploit problem structure?". To answer this question, we began by reviewing analysis of a classic problem in motion planning - the narrow passage. We presented a framework where a planning algorithm is assembled by an implicit graph $\mathcal{G}$ and an incremental search procedure `Search`. We provided examples for each of these components, demonstrating how to systematically incorporate knowledge about the problem structure into this design to boost performance. We examined 3 diverse applications where this technique was applied to design a planning algorithm that met performance standards. We also discussed a theoretical perspective of how planning resolution is linked to obstacle density in the environment.

The framework of decomposing a planning algorithm to $\mathcal{G}$ and `Search` allows us to have better clarity on what it means to "exploit structure". In the 3 examples algorithms we discussed, we interpreted it to have some knowledge about the optimal path ξ^* to Problem!2. This knowledge is in the form of - what are the likely vertices which might belong to ξ^*, what are the likely edges that might belong to ξ^* or even more generally - what lower dimensional manifold might ξ^* belong to.

If this were the case, one may wonder - can we learn a $\mathcal{G}$ automatically to exploit structure? We feel, given the number of classes to which $\mathcal{G}$, can belong - this would be a very hard space to search over. Ichter et al. [2017] takes a significant stride in this direction by training a variational auto-encoder to learn a generative model to approximate the stationary distribution that solutions belong to. However, it is unclear how much information such a model needs to extract from the world at test time. Vernaza and Lee [2011] are able to learn a manifold where ξ^* might lie - however the class of problems belongs to a restrictive set of holonomic planning problems. Hence, as we will see in Chapter 5, a tractable alternative is to allow a human designer to create a library of $\mathcal{G}$ and pose the question as *selecting* a $\mathcal{G}$ from the library.

The `Search` function plays a complementary role to $\mathcal{G}$. In many cases it is agnostic to the problem, and decisions are made purely on tradeoff between desired solution quality and computation budget. However, there are situations where this function needs careful consideration. These are cases where $\mathcal{G}$ is large in size and / or may have a high degree. Such situations allow `Search` to be guided by heuristics. The problem of learning heuristics is much more well defined that learning $\mathcal{G}$. The heuristics inherently act as a *selector function* - selecting from a discrete set of decisions that the `Search` function reasons about at a given iteration. As we will see in Chapter 7, one can learn this selector function to allow the search to explore different parts of $\mathcal{G}$ to find ξ^*.

We close the discussion by talking about planning with *explicit graphs* as an alternate to implicit graphs. An explicit graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ explicitly stores the vertices and edges before planning commences, even though the graph is unevaluated. It has also been known as planning with roadmaps in literature [Kavraki et al., 1996]. In some ways, this paradigm is more restrictive (we can always convert an implicit graph to explicit graph first and then start planning). This is because the graph itself *remains fixed* from one planning problem to the next - the query

and the valid status of edges change. However, explicit graphs offer a powerful capability - we can *explicitly model* the likelihood of edges in the graph being valid without having to use any feature extraction methods. Each edge has an unique integer id - hence we can simply model the configuration space in the space of these ids. This paradigm is attractive as its is completely domain agnostic. As we will see in Chapter 8, we can use this information to compute policies that judiciously decides which edge to evaluate.

4

A Diverse Ensemble of Expert Planners

We return to Challenge 1 which requires us to design a motion planning system that solves real-time kinodynamic planning problems for a fixed distribution of planning problems. In Chapter 3, we presented a framework to design planning algorithms that can exploit the structure of the *optimal path planning problem*. In this section, we will show how these path planners can be used to address the *kinodynamic planning problem*. We introduce in Section 4.2 the term *expert planner* to describe a framework that takes as input the original kinodynamic planning problem, creates a surrogate path planning problem, invokes a path planning algorithm and then projects the solution to a dynamically feasible trajectory.

When designing an expert planner for a problem distribution, we encounter a fork in the road. On one hand, we can design *general purpose planners* that have acceptable performance on a large set of problems, but achieve near-optimal quality on a small fraction of those problems. On the other hand, we have *precision planners* that are primed to work really well on a small set of problems, but completely fail on problems outside this set. We characterize this in Section 4.3. We argue in Section 4.4 that even though we would ideally like to design a strategy to pick an appropriate precision planner for a problem, predicting the performance of such planners is hard as the performance fluctuates with small changes in the environment. Hence we advocate to hedge our bets with an *ensemble* of expert planners. We offer justification behind the decision to run diverse expert planners in parallel in Section 4.5. We state an algorithm to greedily construct this ensemble to maximize performance on a distribution of planning problems. Section 4.6 presents a case study of experiments performed with the ensemble on a full scale autonomous helicopter.

4.1 The planner design challenge

We return to the original planning problem formulation described in Problem 1 and re-state it here for convenience

$$\begin{aligned} \min_{\sigma} \quad & J(\sigma) \\ \text{s.t} \quad & \sigma(0) \in \Sigma_{\text{start}} \\ & \sigma(t_f) \in \Sigma_{\text{goal}} \\ & \mathcal{F}(\sigma(t)) = 0 \\ & \mathcal{H}(\sigma(t)) \leq 0 \\ & \sigma(t) \in \Sigma_{\text{valid}} \\ & \forall t \in [0,\ t_f] \end{aligned} \tag{4.1}$$

Note that this differs from the optimal path planning problem Problem 2) stated in Section 3.2 due to the added presence of temporal dimension.

Let $\Gamma \in \Omega$ be a planning problem and $P(\Gamma)$ be a distribution over planning problem. We now state the problem formulation for Challenge 1.

Problem 3 (Non-adaptive Planner Design)**.** Design a motion planning system, $\mathcal{P}$, that takes as input a planning problem Γ and outputs a trajectory σ, to minimize its expected cost on a distribution of planning problems $P(\Gamma)$, i.e.

$$\mathbb{E}_{\Gamma \sim P(\Gamma)} \left[J\left(\mathcal{P}\left(\Gamma\right)\right) \right] \tag{4.2}$$

4.2 What is an expert planner?

The planning problem in Problem 1 is a high dimensional kinodynamic planning problem which is required to be solved in real-time (≈ 1 Hz). Although this is an active area of research with methods such as Kinodynamic RRT* [Webb and van den Berg, 2013] and SST [Li et al., 2015] pushing the state of the art, it is impractical to apply such methods to the full problem.

We follow guidelines presented by Laumond et al. [1998] and LaValle [2006] and decouple the approach into path planning and trajectory planning. However, we observe that there are *multiple ways* to achieve this decoupling - the success and failures of which depends on not just the dynamics of the system, but *varies* with planning problems. For example, Laumond et al. [1998] describe a curvature rate constrained system being treated as a Dubin's car [Dubins, 1957] to plan a path, following by a "smoothing" method such that a feasible time indexing can be applied. However, planning with the Dubin's car model might not always be a suitable relaxation. In some cases, planning with a smoothness constraint, or even unconstrained planning - followed by the "smoothing" method and time profile assignment can provide acceptable performance. We first present a framework to allow this flexibility and give examples of different relaxations being suitable in different context.

In this paper, we propose the framework of an *expert planner* to solve the planning problem in Problem 1. An expert planner defines a mapping from the original problem to a surrogate low

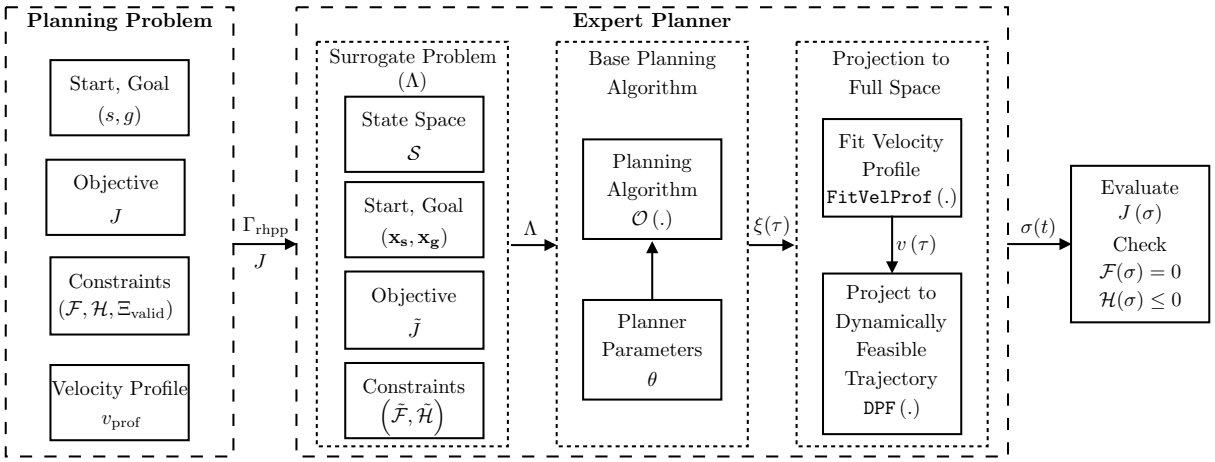


Figure 4.1: Overview of an expert planner.

dimension path planning problem, solves this efficiently using a base planning algorithm, and projects the solution back into the full space to form a trajectory. The term "expert" is borrowed from Blum [1998] to denote that the different modules in the expert planner framework are not independently chosen at random but instead carefully assembled by an expert.

Fig. 4.1 shows an overview of an expert planner. It consists of 3 main modules.

1. *Surrogate Problem* Λ: The expert planner first maps the high dimensional problem Γ_{pp}, J to a low dimensional surrogate path planning problem Λ (Problem 2) with the expectation that a solution to the surrogate problem can be processed to produce a candidate solution to the original problem. The surrogate problem is a tuple $\Lambda = \left(\mathcal{S}, \mathbf{x_s}, \mathbf{x_g}, \tilde{J}, \tilde{\mathcal{F}}, \tilde{\mathcal{H}}, \tilde{\Xi}_{\text{valid}}\right)$ where $\mathcal{S}$ is the search space, $(\mathbf{x_s}, \mathbf{x_g})$ is the pair of start and goal states, $\tilde{J}$ is the objective and $\left(\tilde{\mathcal{F}}, \tilde{\mathcal{H}}, \tilde{\Xi}_{\text{valid}}\right)$ are the constraints.

2. *Base Planning Algorithm* $\mathcal{O}(.)$: The expert planner invokes a base planning algorithm to solve the surrogate problem as discussed in Section 3.3. This base planner is an operator $\mathcal{O}(\Lambda, \theta)$ that maps the surrogate problem Λ and base planner parameters θ to a path $\xi(\tau)$. A path is defined as a functional mapping $\xi : [0, 1] \to \mathcal{S}$.

3. *Projection to Original Space*: The expert planner finally projects the path $\xi(\tau)$ to a trajectory $\sigma(t)$ in the original space. This is done in a two step process. Firstly, a velocity profile is assigned to the path using a prior velocity profile v_{prof}. The operator $\texttt{FitVelProf}(\xi(\tau), v_{\text{prof}})$ maps the path $\xi(\tau)$ and the route guide v_{prof} to a velocity profile $v(\tau)$. Secondly, a dynamics projection filter operator $\texttt{DPF}(s, \xi(\tau), v(\tau))$ is used to map an infeasible path to a dynamically feasible trajectory. Description of this step is provided in Appendix B.

An expert planner satisfies the following contract - it takes as input the planning problem Γ_{pp} and the objective function J, and produces as output either a feasible trajectory σ (that

satisfies $\mathcal{F}(\sigma) = 0, \mathcal{H}(\sigma) \leq 0)$ or an empty solution $\emptyset$. Hence the output can directly be used without the need for any kind of post-processing.

Appendix C shows how a spectrum of expert planners can be created for motion planning for general UAVs. The key thing to note is that the elements of the surrogate problem Λ have to be chosen jointly to create sensible problems that can be solved efficiently. This is why these choices are hard to automate and require an expert. However, each class does have several free parameters which dictate the effectiveness of the approach. This hybrid nature of the expert planner space makes it difficult to apply black-box methods naively to obtain good parameters.

4.3 General purpose planners versus precision planners

We are now ready to address Challenge 1 in its entirety. It is now realized as follows - given a distribution of planning problems $P(\Gamma)$, design an expert planner $\mathcal{P}$ that has high performance, i.e., that sufficiently minimizes expected cost $\mathbb{E}_{\Gamma \sim P(\Gamma)} \left[J \left(\mathcal{P} \left(\Gamma \right) \right) \right]$.

Let us start by considering the example application shown in Fig. 4.2(a). A 2D helicopter is flying in an environment with 4 convex obstacles of different width and height. It always has to navigate to a goal point in front of it. The system has dynamics constraint in terms of how much the robot can roll. For simplicity, assume that the system is flying at a constant speed. For such a problem, a reasonable choice for any expert planner would be to plan with the state space $\mathcal{S} : (x, y, \psi)$, $\tilde{J}$ being the path length and the constraints corresponding to a Dubins car model [Dubins, 1957].

We now come to the choice of base planning algorithm $\mathcal{O}(.)$. Since the obstacles are convex and spaced apart, the free space is likely to have good connectivity. This incentivizes selecting sampling based approaches. Choosing a conventional sampling based planner that does not require many tuning parameters (such as Informed RRT* [Gammell et al., 2014]) should suffice. Since such planners make little assumptions about the problem (other than the connectivity assumption), they are indeed very general purpose.

However, we may encounter a problem occasionally that has a clump of obstacles in the middle as shown in Fig. 4.2(b). We see that our general purpose planner samples the whole space and finds a path that circumvents the cluster. It is unable to find a good path since it lies in a difficult to sample homotopy class. Given the dynamics constraints of the system, the measure of paths that pass through the gap is indeed very small.

We can solve this problem by designing an alternate base planning algorithm. We can keep all other components same, but simply change the sampling scheme to focus on a tunnel around the start and goal. Note that we are injecting domain knowledge that the path must lie in this volume. The algorithm, by focusing the samples is eventually able to power through the gap and reach the goal. Note that such a base planning algorithm is precisely designed to solve such problems and will be unable to return any solution at all to problems where no feasible path exists in this volume.

We now broaden this observation to define two categories of expert planners: *general purpose planners* and *precision planners*. General purpose planners are those that work reasonably well on a large number of problems, but are generally unable to find the optimal path on any problem

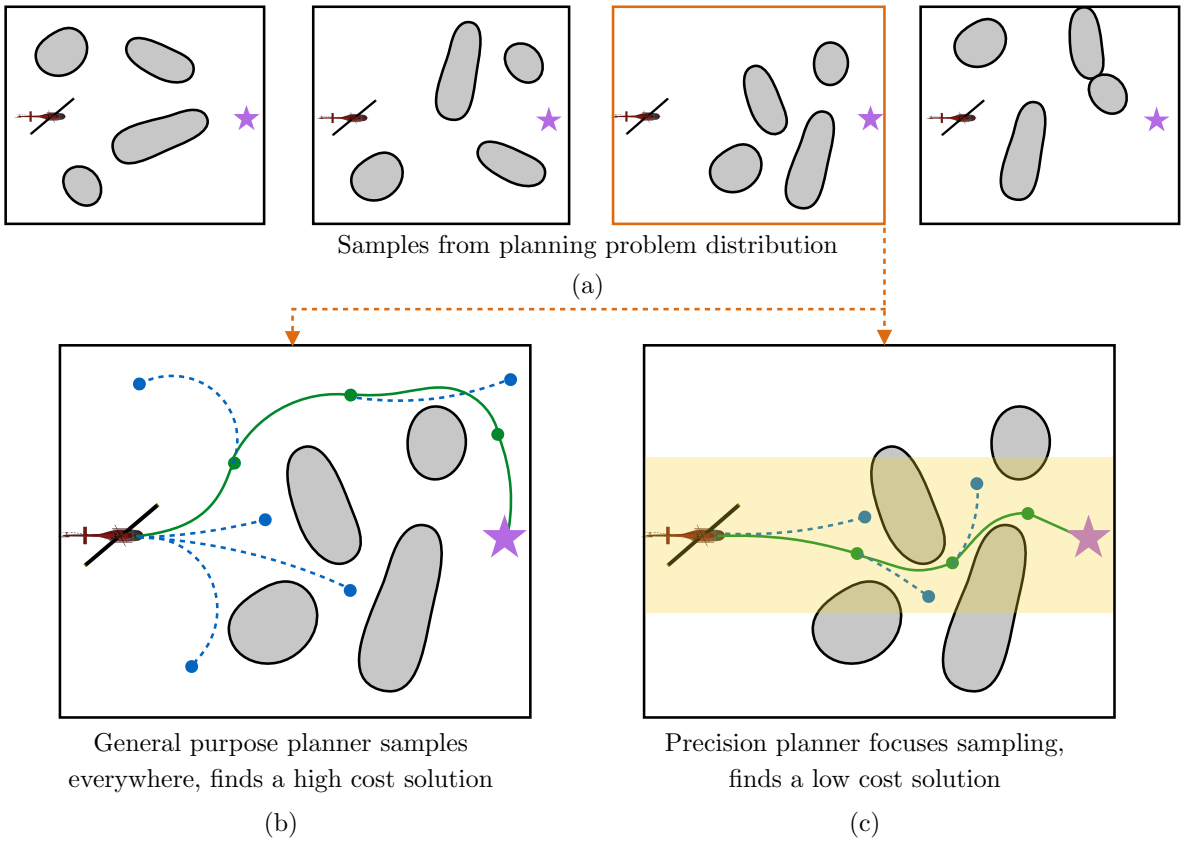


Figure 4.2: Designing expert planners for a planning problem distribution. (a) An example planning problem distribution encountered by a UAV. There are 4 convex objects occurring in various configurations. Since the space appears to have good connectivity, one expects general purpose sampling based planners to do well. (b) The general purpose planner samples everywhere and is not able to sample in the right homotopy class within the time budget. (c) One can design a precision planner that focuses its sampling in the yellow volume and hence is able to eventually find an edge through the gap and reach the goal.

(within the prescribed time budget). Examples are algorithms such as RRT* [Karaman and Frazzoli, 2011], BIT* [Gammell et al., 2015], RRT-Connect [Kuffner and LaValle, 2000], A* on state lattice [Likhachev and Ferguson, 2009], etc. Precision planners are those that finds near-optimal solutions on a small number of problems but fail to produce any solution on most problems. Examples are approaches using local trajectory optimization [Ratliff et al., 2009a, Schulman et al., 2013], shooting methods [Bryson and Ho, 1975] or custom samplers as illustrated just now.

We can try to formalize this. Let Ω be the support of the planning problem distribution. We define the normalized score of a planner as

$$V(\mathcal{P}, \Gamma) = \frac{J_{\max} - J\left(\mathcal{P}\left(\Gamma\right)\right)}{J_{\max} - J_{\min}}$$

Every planner $\mathcal{P}$ corresponds to a score function defined on the domain Ω, where the range of the function is bounded $[0, 1]$. For randomized planners, this function will be noisy - in such

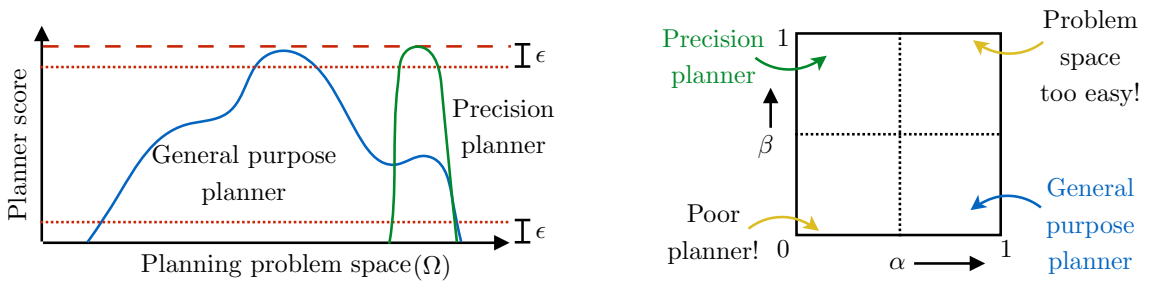


Figure 4.3: The score profiles of general purpose planners in comparison to precision planners. (a) General purpose planners have a larger support but a small fraction of that is withing the $1-\epsilon$ band. Precision planners have a smaller support but almost all of it is in the band (b) The different planners visualized in the α, β space.

cases, consider the mean of the function. Let $\mu(.)$ be the volume.

Given a tolerance parameter $\epsilon \in [0, 1]$, we can choose to represent a planner $\mathcal{P}$ with two normalized terms α and β.

1. Let $\alpha \in [0, 1]$ be the fraction of problems where the score is $> \epsilon$, i.e.

$$\alpha = \frac{\mu(\{\Gamma \in \Omega \mid V(\mathcal{P}, \Gamma) > \epsilon\})}{\mu(\Omega)} \tag{4.3}$$

   This represents the fraction of problems where the planner has an acceptable solution.

2. Amongst the problems where the score is $> \epsilon$, let $\beta \in [0, 1]$ be the fraction of problems where the score is $\geq 1 - \epsilon$, i.e.

$$\beta = \frac{\mu(\{\Gamma \in \Omega \mid V(\mathcal{P}, \Gamma) \geq 1 - \epsilon\})}{\mu(\{\Gamma \in \Omega \mid V(\mathcal{P}, \Gamma) > \epsilon\})} \tag{4.4}$$

   This represents the fraction of problems where the planner is near-optimal.

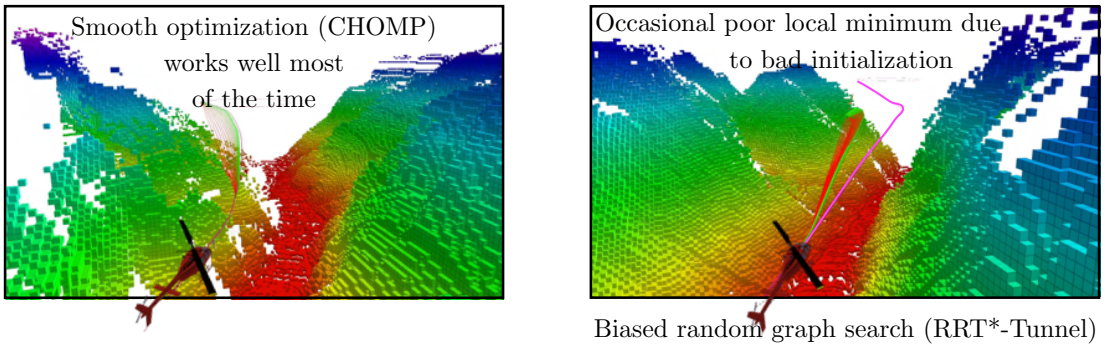
We can now define planners based on these numbers.

Definition 4.1 (General Purpose Planners)**.** Planners that have high α but low β, i.e. solve most problems but have low solution quality on all but a small fraction.

Definition 4.2 (Precision Planners)**.** Planners that have low α but high β, i.e. solves some problems but has high solution quality on most problems it can solve.

This is illustrated in Fig. 4.3. Note that if a planner has high α and β, it implies that the planning problem support is too small for one planner to solve most problems well. On the other hand,, if both α and β is low, the planner has poor performance through out and should be abandoned.

We show a more real-world example in the context of an autonomous helicopter operating in a canyon as shown in Fig. 4.4. Given the topology of the environment present in the distribution of planning problems, one might conclude that an expert planner using a smooth optimization method, `CHOMP`[1] (Appendix C), to be a good candidate. This is motivated by presence of good



(a) (b)

Figure 4.4: Mertis and pitfalls of precision planners. (a) A smooth optimization approach about an initial guess, `CHOMP`[1] (Appendix C) acts as a precision planner by dedicating its planning effort to search around the initial guess. This works remarkably well in many cases and outperforms other general purpose planners. (b) However, there are cases when this planner fails catastrophically. In such a scenario, a relatively more general purpose planner, `RRT*Tunnel`[1] (Appendix C), is still able to find an acceptable solution.

local minima around the nominal path between start and goal as shown in Fig. 4.4(a). However, the planning problem contains examples such as Fig. 4.4(b) where this expert planner fails to find any feasible path. This is due to poor initialization of the algorithm. Here the `CHOMP`[1] planner plays the role of a precision planner that makes strong assumptions about the environment to boost performance but fails disastrously when those assumptions do not hold.

Contrast this to an expert planner using a biased random graph search approach, `RRT*Tunnel`[1] (Appendix C), which focuses sampling in a volume around the nominal path. The general performance of this algorithm is not comparable to `CHOMP`[1] owing to it spending its planning effort collision-checking areas that are almost always occupied by obstacles. However, RRT*-Tunnel is more likely to find a solution as shown in Fig. 4.4(b).

4.4 Difficulty in performance prediction of precision planners

Since we have established that there is no universally effective planner, how do we go about designing a planning system to solve Challenge 1. One approach is along the lines of "examine the problem, select an approach to execute, feed the output to another process and continue". This would be a single-threaded sequential planning strategy. This is a common convention followed by planning systems. It is resource efficient in the sense that the problem is processed to identify a good precision planner to execute and then all the resources are fed to that module.

However, such approaches usually run into a problem. Any single threaded planning module must be able to reasonably predict the performance of precision planners given a problem. Consider the scenario shown in Fig. 4.5. The slightest change in the environment dramatically alters the behaviour of a precision planner. This is due to the fact that, in the presence of dynamic constraints, small changes in the workspace can invalidate an entire homotopy of paths. Precision planners that focus on that homotopy go from always finding the best path to not finding a path at all. Under such uncertainty, allowing for redundancies at the expense of potentially additional

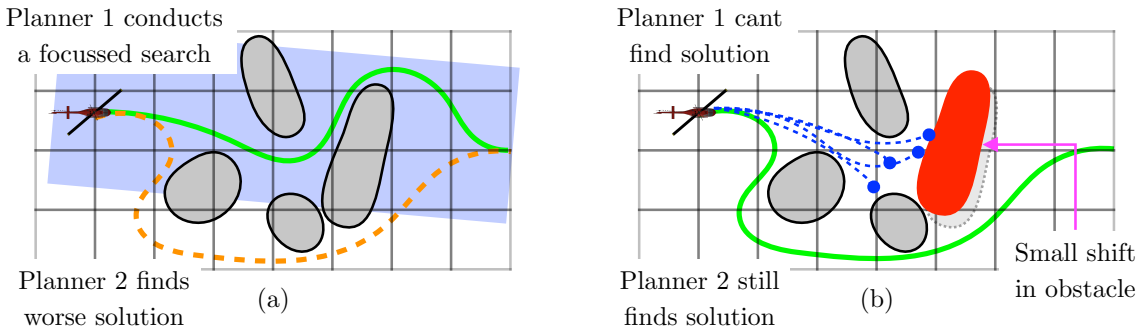


Figure 4.5: Difficulty in predicting the performance of a precision planner. (a) Planner 1 focuses its search in a volume and is able to compute a high quality solution. Planner 2, which does not focus its search, finds a solution of much lower quality. (b) The slightest shift in obstacles now makes it impossible for Planner 1 to find a solution given the dynamics constraints. Planner 2 still finds reasonable solutions. The difference between these two environments is too small for a context to pick up.

planning effort has significant payoff.

There are some auxiliary reasons as well for not pursuing a single-threaded strategy

1. Non-monotonic effects of composing planners: There is no guaranteed monotonicity in adding modules to an expert planner. E.g - adding a complex initialization procedure for a trajectory optimizer could degrade performance in scenarios where a straight-line initialization would have worked had the entire effort been dedicated to it.

2. Difficulty in automation: Hybridization is a difficult process to automate. Even if one were to accept the non-monotonic nature of adding modules, composition of different strategies might not always be straight forward.

4.5 Ensemble of diverse expert planners

We advocate the use of an *ensemble of expert planners*. This is illustrated in Fig. 4.6. At a given decision step, the planning problem is provided to an ensemble of planners running in parallel. Each planner is *independent* of another. The plans are provided to a trajectory selector that takes the best of the solutions and sends it out.

At the expense of redundant planning effort, this architecture enjoys several benefits such as

1. Monotonic effect of adding planners: Adding a planner always helps. This rests on the assumption of independence, i.e. each planner runs on its own thread, has an individual assignment of memory and resources that it does not have to share.

2. Submodular effect of adding planners: Adding a planner has diminishing returns. This incentivizes a natural stopping criteria, or budget for the ensemble. This, in conjunction with the monotonic property, also provides strong guarantees for greedily selecting planners as will be discussed in Section 5.5.1.

3. Ease in automation: The architecture allows different and mutually incompatible expert planners to be used in conjunction, thus allowing automatic selection of an ensemble.

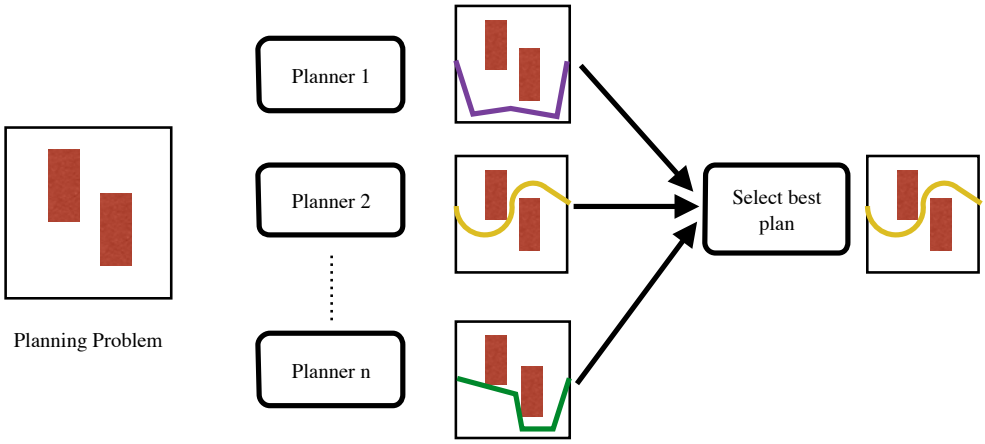


Figure 4.6: Ensemble of expert planners. Each expert planner computes a trajectory and the best one is sent to the robot.

Algorithm 3: Greedy Design of Ensemble

Sample planning problem dataset $\{\Gamma_i\}_{i=1}^N$ from $P(\Gamma)$;
$\mathcal{E} \leftarrow \emptyset$;
Initialize ϵ to a small value;
Initialize $\mathcal{D}$ with $\{\Gamma_i\}_{i=1}^N$;
while $|\mathcal{E}|$ *is less than a budget* **do**
    Design expert planner $\mathcal{P}^*$ to solve as many problems in $\mathcal{D}$; ▷ `Solve implies` $V(\mathcal{P}, \Gamma) > \epsilon$
    Add expert planner $\mathcal{P}^*$ to ensemble, $\mathcal{E} \leftarrow \mathcal{E} \cup \mathcal{P}^*$;
    Increase ϵ;
    Copy unsolved problems from $\{\Gamma_i\}$ to $\mathcal{D}$; ▷ `Unsolved implies no planner in ensemble solves`
return $\mathcal{E}$;

We now describe an approach to solving Challenge 1 using a simple greedy procedure as described in Algorithm 3. Let $\mathcal{E}$ be an ensemble of expert planners that we wish to design. We iteratively inspect the dataset of planning problems that have not been solved to an acceptable performance threshold. We then design a planner that makes a best effort to solve these problems. We increase ϵ. We update the unsolved dataset and repeat.

Let us examine the implication of this design principle by looking at Fig. 4.7. Since we sample $\{\Gamma_i\}_{i=1}^N$ from $P(\Gamma)$, we only have to reason about the performance of planners uniformly on this set. Since ϵ is small initially, we will be incentivized to design a general purpose planner with high α. As we increase ϵ, we will progressively require planners to have higher β. The attractive aspect of this algorithm is that it becomes progressively easier to design expert planners $\mathcal{P}^*$ as the planning problems database shrinks. This design principle effectively forces designing planners to chase down modes of the planning problem distribution. At the same time, we are not forced to think about the solution cost of the planners explicitly - it implicitly defines the set of unsolved problems that we focus on.

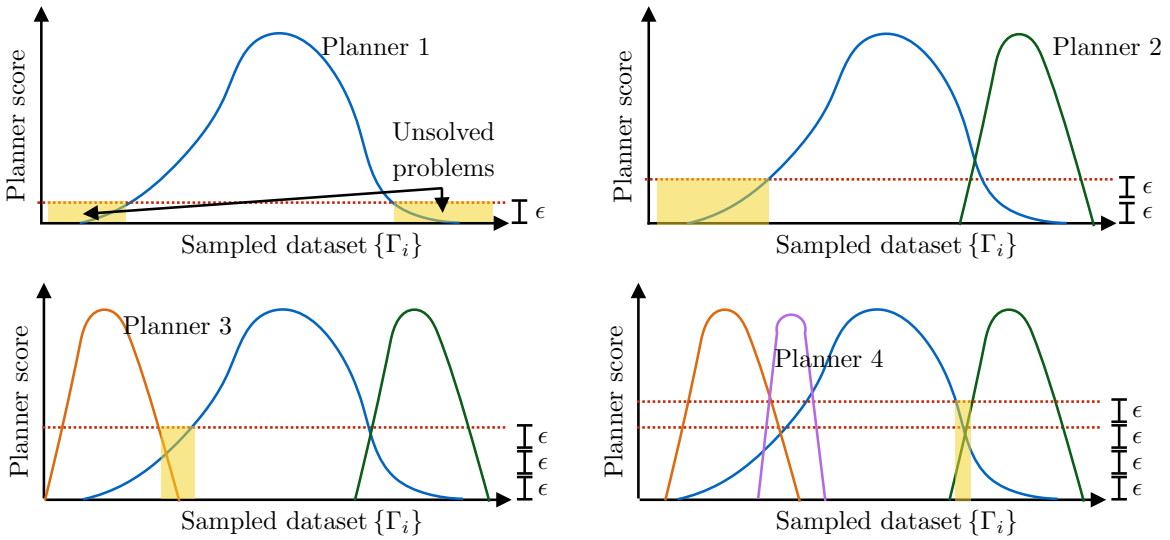
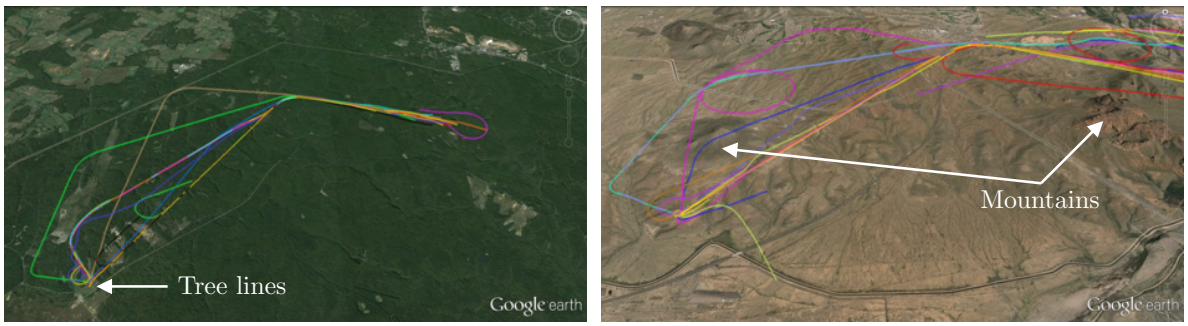


Figure 4.7: Greedy design of ensemble using Algorithm 3.



(a) (b)

Figure 4.8: Different planning problem distributions for an autonomous helicopter (a) Missions in Quantico, VA, where obstacles are towers, trees and no fly zones (b) Missions in Mesa, AZ, where obstacles are mountains and no fly zones.

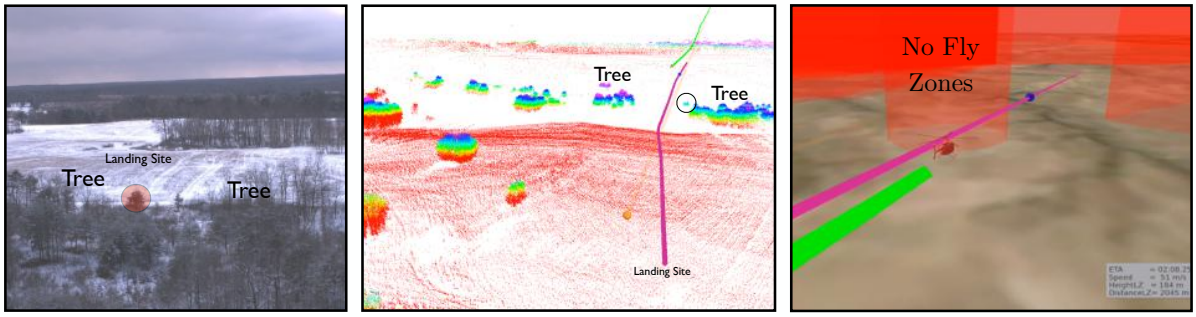
4.6 Case Study: Autonomous helicopter

We now show a case study for ensemble design and results on an autonomous full scale helicopter. Details regarding the dynamics and expert planners are described in Appendix A and C.

4.6.1 Planning Problem Distribution in Quantico, VA

We first consider a set of planning problem distributions where the helicopter had to operate in Quantico, Virginia. The planning problem distribution $P(\Gamma)$ consists of relatively obstacle-free space. Obstacles present in the environment consists of towers and trees as shown in Fig. 4.8(a). The planning problems also contained no-fly zones appearing dynamically.

We apply technique in Algorithm 3 to create an ensemble of planners. The first planner $\mathcal{P}$ in

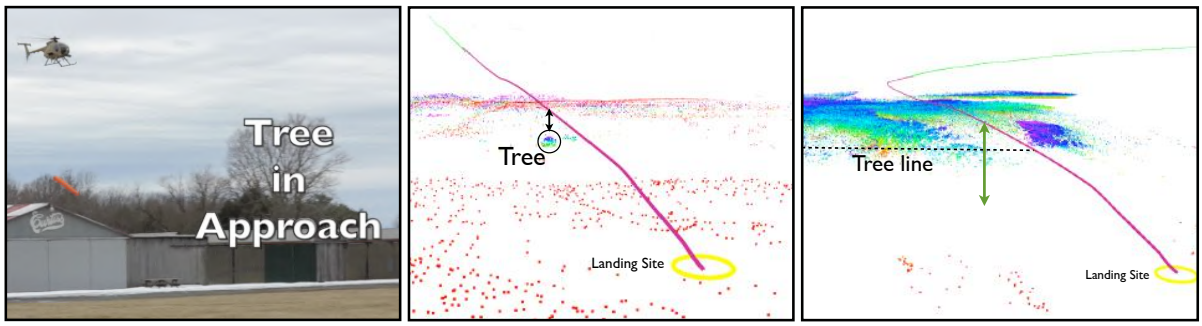


(a) (b) (c)

Figure 4.9: Different planning problems encountered during operation in Quantico, VA. (a) Perspective from the helicopter where the robot must avoid a tree while it comes down to a landing site (b) The system uses CHOMP[1] to optimize a trajectory to avoid the tree-line (c) The system also maneuvers around no-fly-zones.

the ensemble is a planner that has a good performance on most problems. Since most problems were amenable to local optimization, we created an expert planner CHOMP[1] (Appendix C) that used a Dubins visibility graph technique to compute an initial guess which was optimized using covariant gradient descent. This was useful in solving most planning problems with no-fly zones and trees as shown in Fig. 4.9. This serves as a good general purpose planner.

We prune out all solved problems in the dataset. We are left with problems where CHOMP[1] violated glide slope constraints and hence could not find a feasible solution. An example of such a scenario is shown in Fig. 4.10(b). We designed a new expert planner CHOMP[2] to run in parallel. CHOMP[2] is a modification of CHOMP[1] to always enforce constraints at every step at the expense of slower run-times and occasional inability to find solutions. The results of CHOMP[2] are shown in Fig. 4.10.



(a) (b) (c)

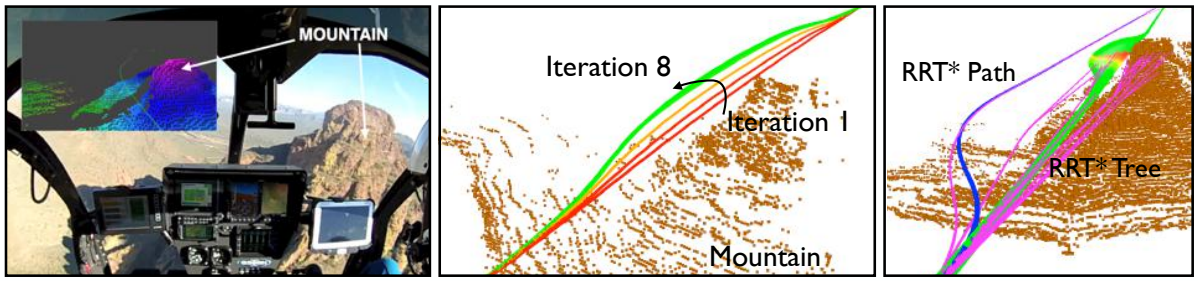
Figure 4.10: Unsolved problems by CHOMP[1] which are solved by CHOMP[2] (a) A tree on approach which is detected late in the approach stage (b) CHOMP[2] avoids the tree by optimizing over it but respecting glide slope constraints (c) CHOMP[2] solves other outlier problems as well

4.6.2 Planning Problem Distribution in Mesa, AZ

We now show the sensitivity of the expert planners in a different set of missions. Consider the planning problem distribution $P(\Gamma)$ in a set of mission runs in Mesa, AZ. Here the obstacles present consists of mountains in conjunction with dynamic no-fly zones.

We again apply the technique of Algorithm 3 to create an ensemble of planners. The first planner $\mathcal{P}$ was chosen as `CHOMP`^{1} as it still worked reasonably well for most cases and was not required to be modified. On pruning out problems, we end up with problems involving mountains where `CHOMP`^{1} failed due to reasons of poor local minima around the initial trajectory. We designed an expert planner `RRT*Tunnel`^{1} (Appendix C) that is a random graph search with a customized workspace sampling technique biased to be in a volume around the nominal trajectory.

Fig. 4.11 shows a scenario where the helicopter autonomously avoided a mountain. `CHOMP`^{1} produces high quality trajectory while `RRT*Tunnel`^{1} fails to find a good trajectory but solves it none the less.



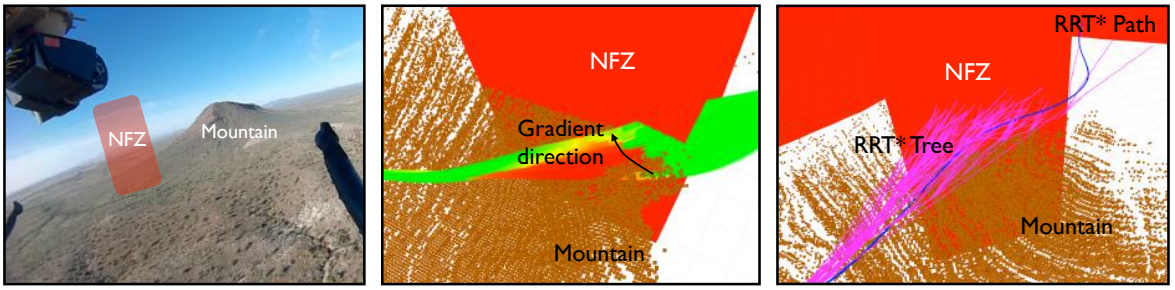
(a) (b) (c)

Figure 4.11: `CHOMP`^{1} enables avoidance of a mountain enroute (a) A cockpit view of the scenario (b) The iterations of `CHOMP`^{1} to smoothly optimize a path (c) `RRT*Tunnel`^{1} is unable to match the solution quality of `CHOMP`^{1}, but solves the problem as well.

However, there were situations where the reverse happened as shown in Fig. 4.12 where `CHOMP`^{1} fails and `RRT*Tunnel`^{1} succeeds.

4.7 Conclusion and discussion

In this Chapter, we presented an expert planner framework to solve the kinodynamic planning problem specified in Problem 2. This is a framework that takes as input the original problem, creates a surrogate path planning problem, invokes an appropriate path planning algorithm from Chapter 3, projects the solution to a high dimensional trajectory. Having created this framework, we returned to Challenge 1. We showed on one hand, we can design *general purpose planners* that have acceptable performance on a large set of problems, but achieve near-optimal quality on a small fraction of those problems. On the other hand, we have *precision planners* that are primed to work really well on a small set of problems, but completely fail on problems outside this set. We argued against the single hierarchical planning framework of encoding a complex set of rules to pick a planning strategy due to 2 main reasons - unpredictable effect



(a) (b) (c)

Figure 4.12: `RRT*Tunnel`[1] is able to find solutions where `CHOMP`[1] fails. (a) A scenario where the helicopter has to fly between a mountain and a no fly zone (b) This scenario creates a bad local minima where the gradient due to the presence of the mountain conflicts with the no fly zone. `CHOMP`[1] is unable to find a feasible path (c) `RRT*Tunnel`[1], which does not use the gradient information, samples the gap and is able to find a solution.

of design decisions and difficulty in automation. We instead advocated for hedging one's bets with an ensemble of expert planners. We discussed the properties that this framework enjoys - monotonicity, submodularity and ease in automation. We presented an approach to greedily construct this ensemble. Finally, we conduct a case study for an autonomous helicopter where we demonstrate how an ensemble was designed in practice.

The monotonicity and submodularity aspects of the ensemble are particularly attractive for a number of reasons. These properties allow greedy ensemble selection approaches to have guarantees, which will be exploited later in Chapter 5. The greedy selection logic has the side-effect that as more and more planners are added, the set of unsolved problems starts to decrease - thus making it easier to focus on them. The ensemble framework also allows one to deal with unsolved problems easily - create a planner to handle this exception case and add it to the ensemble. We will explore this in Chapter 6.

The explicit characterization of planners as general purpose and precision, while not directly used, is helpful when reasoning about the question of - "How much information do we need to compute a good ensemble?". One one hand, if we only have a library of precision planners, the combined coverage of the ensemble would be very less. This incentivizes investing a lot of resources in context extraction. On the other hand, if we have general purpose planners in the library, we would be better off spending more resources executing planners than to judiciously determine which planners to execute.

Finally, we want to revisit the premonitions of designing a single hierarchical framework for a planning system. Having a single planner is certainly efficient in terms of using information from collision checking. However, one concern is that allocating planning effort into deliberating about which planning strategy to use might be detrimental compared to having used the strategy throughout. This trade-off will be formally tackled in Chapter 7 to create planners that adapt their search with increasing information about the problem.

Part II

Black-box Adaptive Planners

5

Adaptive Ensembles of Expert Planners

In the previous Chapter, we addressed Challenge 1 - designing a good non-adaptive planner for a fixed problem distribution. We proposed a framework for designing an ensemble of planners. In this Chapter, we proceed to address Challenge 2 - dealing with varying distributions. Since it is not possible to manually change the ensemble, we need an automatic way of generating such ensembles. We propose to learn a meta-planner that can select from a library of black-box expert planners.

We begin by motivating the need for such a meta-planner in Section 5.1. We next present some pertinent relevant work in Section 5.2. We formulate the problem in Section 5.3. Before discussing the learning section, we spend some time offering various practical and algorithmic procedures for generating a library of expert planners in Section 5.4. We present algorithms to select a static ensemble in Section 5.5. A static ensemble meta-planner selects a fixed ensemble for a planning problem distribution. We show that this can be done efficiently without having to execute all planners using a lazy greedy algorithm. This algorithm requires trained priors on planner performance.

We then discuss the conditions that drive the need for a dynamic ensemble. A static ensemble suffices when general purpose planners - planners that do reasonably well on a large number of problems - have sufficient performance. When precision planners - planners that have high performance on some problems and fail catastrophically in others - are required to be selected, the ensemble cannot remain static. It needs to change for different planning problems. In Section 5.6, we present a framework to train a list of contextual predictors to select an ensemble using context from the planning problem. We show how the problem is a loss-sensitive classification which has better empirical performance than learning priors on planners. We present experimental evaluations in Section 5.7 for both the static and dynamic ensemble selection method on a wide range of planning problem datasets. Finally, we present flight test results for closed loop evaluation of an adaptive ensemble on two UAV platforms in Section 5.8.

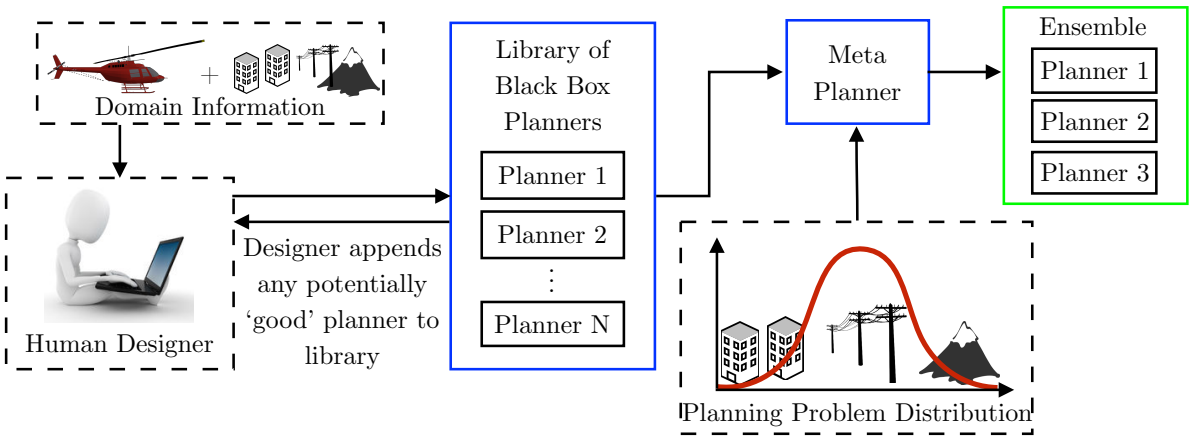


Figure 5.1: The black-box adaptive planning framework. The human designer has access only to domain information with which he creates a library of planners. These planners are *black-box* - the content of the planners is not visible to any other module, hence granting the designer full freedom. The objective is to design a meta-planner that takes as input the planning problem distribution and selects an ensemble of planners from the library.

5.1 Introduction

The problem of designing a real-time motion planning system for an application is a challenging one. A plethora of motion planning techniques exist to approach the problem [LaValle, 2006]. Many of these methods are endowed with worst case guarantees such as completeness [Canny, 1988a] or asymptotic optimality [Karaman and Frazzoli, 2011]. However, when run for a *finite time* budget, the efficacy of such approaches depends heavily on the underlying structure of the environment in which the robot operates - i.e. the configuration of obstacles, dynamics of the robot and typical start / goal queries. The problem is further exacerbated when robots are required to operate across environments, i.e. when the performance of the planning strategies fluctuate during an episode.

Machine learning approaches offer the ability to adapt motion planning algorithms to a specific environment. An attractive paradigm for integrating learning in planning is the framework of *black-box planner selection*. In this setting, we assume we have a library of planning strategies - these could be different algorithms, or different parameters of the same algorithm. Given an environment, we want to predict which element to use. The learner treats each of the library elements as a black-box, i.e. no assumption is made about the contents of the algorithm. Hence, this allows the human designer full flexibility in employing any approach in the library. An illustration of this framework is shown in Fig. 5.1.

The objective is to find a learning procedure that has good prediction performance, i.e, how well can the learnt procedure predict the best element in the library? Machine learning procedures provide guarantees on expected performance, which is the average performance over a probability distribution of environments. A predictor trained on this distribution has interesting behavior. It predicts the correct element on most environments, but has extremely low performance on environments which are infrequent. This is particularly unsatisfactory for motion

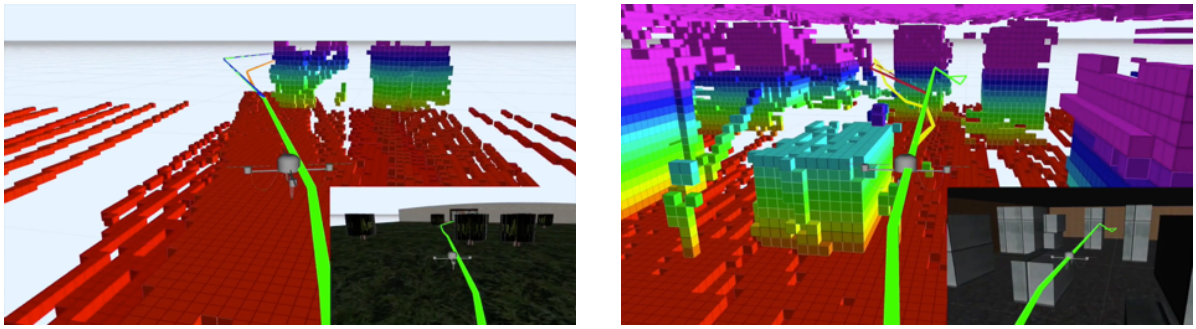
planning, which demands good prediction performance on all environments. This motivates us to examine the following question:

> Can we design a meta-planner, that selects planners from a library of candidate planners, such that it has good performance on most environments?

5.1.1 Motivation

Outdoors: High speed / Sparse obstacles

Indoors: Low speed / Clutter



(a) (b)

Figure 5.2: Different classes of planning problems encountered by an UAV in a cargo delivery application (a) When flying outdoors at high speeds, the planning problem requires finding smooth feasible paths that bend around an obstacle blocking the goal. (b) When flying indoors at low speeds, the environment has a lot of obstacles. While dynamics is not as important, the planner has to search the whole space to find a feasible path to goal.

Consider the problem of a UAV carrying out a cargo delivery operation. Such systems are required to fly repeatedly from outdoors to indoors. The configuration of obstacles vary a lot across these environments as shown in Fig. 5.2. In outdoor environments, the number of obstacles is sparse. However, since the robot is flying at a higher speed, the dynamics are more restrictive. Hence trajectory optimization approaches that search for smooth solutions about an initial straight line guess are better suited for these planning problems. In indoor environments. the space is much more densely filled. However, the robot is flying at a lower speed where dynamics is not a concern. Global sampling based approaches are able to find paths quickly and hence are most suited.

Such an application requires a meta-planner that can select from a library of approaches based on context extracted from the problem. Context here can be specified as the speed of the vehicle and density of obstacles at different locations. Interestingly, a small amount of context is sufficient for the meta-planner to identify a good set of planners to attempt on the problem. The meta-planner is not required to know all obstacle configurations in the environment.

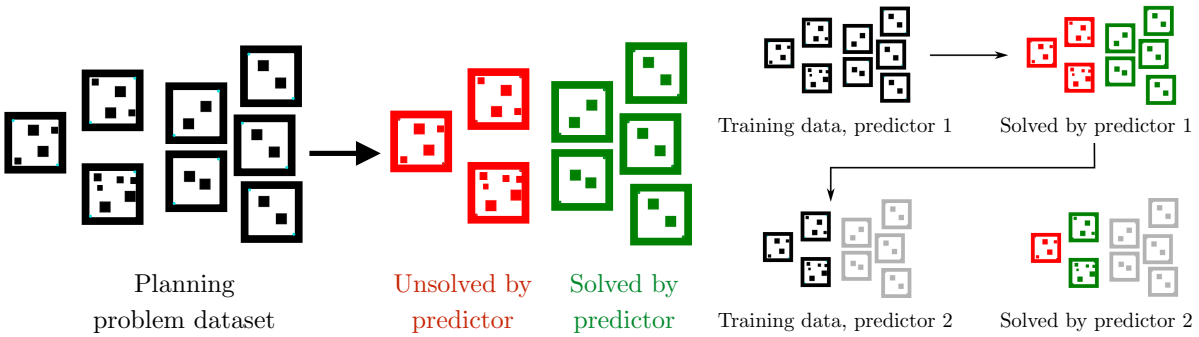


Figure 5.3: Training predictors to greedily focus on unsolved problems. On the left, we see that a predictor that maps a planning problem to planners in a library acts as a filter. The mapping results in the problem being solved or remaining unsolved. When a list of predictors are being trained to predict planners in an ensemble, each predictor should greedily focus on the unsolved problems of the previous predictors in the list.

5.1.2 Key ideas

Questions pertaining to the worst-case performance of predictors fall in the realm of learning theory. However, in this work, we do not follow this line of analysis. We instead advocate for a procedure that leverages a powerful attribute of motion planning - algorithms can be executed in parallel to solve the problem, each solution can be evaluated using a crisp objective function and the best solution can be returned. This in turn implies that the meta-planner is not burdened to commit to a single prediction - it can make a list of predictions. Since only the best prediction is relevant, the meta-planner is incentivized to hedge its bets and choose a diverse ensemble of planners.

To predict a diverse ensemble of planners, we propose a meta-planner consisting of a list of selector policies - given a planning problem, each policy predicts a planner for a particular slot in the ensemble. The key insight is that these policies can be trained efficiently using a greedy procedure - each subsequent policy in the list is trained on the failure case of the preceding policies. Intuitively, these policies act as *filters*. The first policy in the list is able to successfully solve a majority of the problems, thus passing down harder problems for the second policy to examine. The second policy only focuses on classifying these hard problems and so on. The intuition is that by focusing on the hard problems, the task of finding a dividing hyperplane becomes easier as depicted in Fig. 5.3. We will support these intuitions with mathematical justification to learn effective meta-planners.

The approach of employing list prediction in motion planning can be viewed as a form of data-driven redundancies. Redundancies are key to ensuring robust behaviours. One can interpret the learning rule as a way of designing redundancies by examining the correlated failures of different planning approaches from data. We also extend the idea of using data to the problem of what makes a good library of planners. We show how the framework of online set-cover leads to a practically useful algorithm for building a library of planners.

5.1.3 Contributions

Our contributions are as follows:

1. We cast the problem of dynamically selecting an ensemble of planners as list prediction. We derive a training procedure for list prediction in the framework of loss-sensitive multi-class classification (Section 5.6).

2. We examine the special case of predicting a static ensemble and propose a lazy evaluation based approach that does not require re-evaluating planners on new training data (Section 5.5).

3. We cast the problem of designing a library of planners as an online set-cover problem and derive an efficient algorithm (Section 5.4).

4. We extensively evaluate the approach on a spectrum of planning problems involving seed prediction for trajectory optimization, heuristic selection for 4D arm planning, and planner selection for autonomous helicopters. We show that list prediction outperforms single item prediction (Section 5.7).

5. We present extensive closed-loop evaluation of our framework on UAVs. The adaptive planning system improved the performance of the planning system from $< 50\%$ to 95.56%, thus enabling the UAV to fly robustly for hours across varying environment, speeds and problem difficulties (Section 5.8).

5.2 Background

5.2.1 Option prediction in planning

There has been a body of work in employing machine learning techniques to predict options in planning. Jetchev and Toussaint [2013] was an early work on predicting seeds for trajectory planning. Cost regression and classification were implemented, without the formalism of loss-sensitive classification. Their results show that cost regression is a more difficult task. Dragan et al. [2011] predicted the usefulness of end-effector goals for trajectory planning on a manipulator. Some heuristics in their procedures are explained when a surrogate loss is used to derive algorithms. Zucker [2009] generated a 'behavior library' of optimized trajectories and predicted the best trajectory given a query. Berenson et al. [2012] generated a library of past plans and used a heuristic to select one that can be repaired easily to solve a new environment. Pan et al. [2014] predicted if a seed trajectory will be successful for local optimization. Poffald et al. [2014] used a library of motion primitives and predicted the best primitive that can be adapted for a new environment. Wzorek et al. [2010] predicted a motion planning strategy, from a library, that can be applied to repair a plan. Palmieri and Arras [2015] learned to select the nearest neighbour from a candidate set of neighbours in an RRT method to improve the likelihood of connection.

5.2.2 List prediction

List prediction requires predicting a list of items that are diverse and relevant. Radlinski et al. [2008] examine the problem of recommending a diverse set of news articles to increase the chance that a user would like at least one article. The notion of diminishing returns due to redundancy is captured formally in the framework of submodularity [Krause and Golovin, 2012]. While exact submodular function optimization is intractable, greedy selection is known to have strong near-optimal performance guarantees.

There exists a host of method that decompose list prediction to a sequence of easier learning tasks that attempt to mimic the greedy strategy [Dey et al., 2013, Radlinski et al., 2008, Streeter and Golovin, 2009]. Dey et al. [2013] propose an algorithm ConseqOpt to reduce the list prediction problem to stacked sequential cost-sensitive classification. Each problem aims to predict an element in the library to maximize the expected marginal utility. The attractive aspect of this approach is that it offers a guarantee with respect to the class of all possible sequence of predictors, which is quite expressive. However, this comes at the expense of being less data-efficient than methods that make make realizability assumptions such as [Raman et al., 2012, Yue and Guestrin, 2011]. Ross et al. [2013] considered list prediction in a different setting where a single predictor is trained to predict a list. They derive a learning rule for such a policy and leverage the data efficiency of using a single learner.

5.3 Problem formulation

We build on the framework of designing an ensemble of expert planners that we presented in Chapter 4. Let $\Gamma \in \Omega$ denote a planning problem that the expert planner takes as input. Let $P(\Gamma)$ be the distribution over planning problems encountered by the robot.

We wish to train a meta-planner to select black-box planners from a library. Let $\mathcal{L} = \{\mathcal{P}_i\}_{i=1}^{|\mathcal{L}|}$ be a finite library of expert planners. By abuse of notation, let $J(\Gamma, \mathcal{P}) \in [0, J_{\max}]$ be the cost of a trajectory when applying planner $\mathcal{P}$ on problem Γ. Infeasible trajectories are assigned a cost of $J_{\max}$. The expected performance of a planner $\mathcal{P}$ on a distribution $P(\Gamma)$ over planning problems is $\mathbb{E}_{P(\Gamma)}[J(\Gamma, \mathcal{P})]$.

Let $\mathcal{E} \subseteq \mathcal{L}$ be an ensemble of expert planners of budget $|\mathcal{E}| = B$. The expected performance of the ensemble is $\mathbb{E}_{P(\Gamma)}\left[\min_{\mathcal{P}_i \in \mathcal{E}} J(\Gamma, \mathcal{P}_i)\right]$.

Let π be the meta-planner. The output of this meta-planner is an $\mathcal{E}$. The meta-planner can be either *static* or *dynamic* in nature. A static ensemble meta-planner outputs a fixed ensemble for a planning problem distribution. A dynamic ensemble meta-planner can output an ensemble that may change with planning problem Γ.

Let f be a feature vector representing context about planning problem Γ. The meta-planner is a policy that may use this context in selecting an ensemble. We now define the problem as follows

Problem 4 (Ensemble selecting meta-planner)**.** The meta-planner design problem is formally defined as training a policy $\pi(\Gamma, \mathcal{L})$ to select and ensemble from a library of planners $\mathcal{L}$ on the

distribution of planning problems $P(\Gamma)$ to optimize the following

$$\min \mathbb{E}_{P(\Gamma)} \left[\min_{\mathcal{P}_i \in \pi(\Gamma)} J\left(\Gamma, \mathcal{P}_i\right) \right] \tag{5.1}$$

5.4 Constructing a library of expert planners

In this section, we discuss different approaches of creating a good library of expert planners. We start with practical approaches and then use intuitions and insights to formalize the library creation process.

Let us recall the process of creating expert planners that we discussed in Chapter 4. We discussed in Section 4.2 the space of possible expert planners, its hybrid nature and difficulty in searching over this space. However, the problem of creating a good library of expert planners is considerably simpler. This is due to the fact that the objective is no longer to map individual planning problems to effective planners, but instead to ensure the library has good *coverage*, i.e. for every conceivable problem, there exists at least one effective planner in the library.

We start by examining the information available to us at the time of library creation. It is impractical to create a library of expert planners for all possible applications. Instead, the human engineer is given vital domain information to aid in library creation process. This comprises vehicle dynamics, expected velocity regimes of operation and example environments of operation. Formally speaking, the human engineer has access to the support Ω of all possible distributions $P(\Gamma)$. Hence a good library is one that covers as many problems in Ω.

5.4.1 Approach 1: Combine general purpose planners with precision planners

We first present a practical approach to constructing a library. We populate the library with planners that use general purpose planning algorithms with different parameter sets. This library is then executed on a database of problems uniformly sampled from Ω. The failure cases are manually inspected. Using techniques presented in Chapter 3, we design precision planners to solve these problems. These planners exploit some structure in the failure cases to come up with sampling schemes / heuristic / initial seeds to solve these problems. We can greedily design precision planners to solve as much of the failure cases as possible. Appendix C describes the library of planners designed for general UAVs using this pragmatic approach.

5.4.2 Approach 2: Greedy planner creation by hard negative mining

We present a more automatic approach for library creation using insights derived from Approach 1. One can think of the process of collecting failure cases and using it to refine the library as hard negative mining. We can formalize this process as follows. We start with an empty library $\mathcal{L} = \emptyset$. We iteratively sample problems from Ω. If no element in the library is able to solve the problem Γ_i, we invoke an oracle to create a library element $\mathcal{P}$ that can solve *at least* Γ_i. This process is repeated to build up the library. As elements are added to the library, the coverage of the library increases. The negative mining process searches for harder and harder problems that the library has not yet solved.

Algorithm 4: Greedy Randomized Rounding $(\Omega, \mathcal{F})$

Initialize $\mathcal{L} = \emptyset$;
Initialize planner membership variable $x_p = \frac{1}{M}$;
for $i = 1, \ldots, N$ **do**
  Receive planning problem Γ_i from Ω;
  if $\mathcal{L}$ *can not solve* Γ_i **then**
    $\mathcal{R} \leftarrow \emptyset$;
    for $p = 1, \ldots, |\mathcal{F}|$ **do**
      if $\mathcal{P}_p$ *can solve* Γ_i **then**
        $\mathcal{R} \leftarrow \mathcal{R} \cup p$;
    while $\sum_{r \in \mathcal{R}} x_p < 1$ **do**
      $x_r = 2x_r, \ \forall r \in \mathcal{R}$;
    for $r = 1, \ldots, \mathcal{R}$ **do**
      Add $\mathcal{P}_r$ to $\mathcal{L}$ with probability $\frac{\Delta x_r \log N}{1 - x_r \log N}$;
return $\mathcal{L}$;

We used this approach to create a library of trajectory optimization planners in Tallavajhula and Choudhury [2015]. In this case, the library is composed of different seed trajectories that the optimizer initializes from. When no seed in the library can solve a problem, a more expensive global planner is invoked to solve the problem. The solution of this planner is then used the seed.

5.4.3 Approach 3: Library selection as online set-cover

Approach 2, while being effective, results in an algorithm that has no performance guarantees. As a result, it can populate the library with many elements while having poor coverage. It also does not allow a principled way to incorporate an existing library of general purpose planners.

We want a framework where we have the following properties:

1. We cannot evaluate all candidate planners on all possible problems in Ω. Hence we need a framework that streams problems from Ω, maintains an existing library and only evaluates all candidate planners if no element in the library can solve the problem.

2. We want to minimize the size of the library to reduce computation during the training process for the meta-planner

We cast the problem of library selection as online set-cover. We are given a set of N planning problems $\Omega = \Gamma_1, \ldots, \Gamma_n$. We have a family of M candidate planners $\mathcal{F} = \mathcal{P}_1, \ldots, \mathcal{P}_m$. We want to construct a library $\mathcal{L} \subset \mathcal{F}$ by selecting planners from the family. The cost of selecting a planner is 1.

Each planner in the family $\mathcal{P}_i \in \mathcal{F}$ solves a set of planning problems in Ω, i.e. on executing the planner on these problems, the solution cost is below a threshold J_{thresh}. Hence each element of the family corresponds to a subset of Ω. A cover is a collection of such sets such that their union is Ω. We want to find a cover with minimum cost which is known to be a NP-hard problem.

We specifically examine the online variant where planning problems arrive one by one and have to be covered immediately. Additionally, once a planner is selected in the library, it cannot be removed. The online variant is more apt in our case as we do not wish to explicitly evaluate all planners on all problems, i.e. the membership function of planners to problems is not known upfront. It can be incrementally discovered by querying planners with problems.

We adopt the greedy algorithm with randomized rounding to solve this problem [Buchbinder et al., 2009]. The algorithm is derived by mapping the set-cover problem as a linear program. Every time a new problem arrives, this linear program needs to be re-solved. One can show that a near-optimal greedy algorithm exists to efficiently solve this. Since the set selection is a integer problem, one can use a monotone randomized rounding scheme to project the greedy solution into the integer domain. We refer the reader to Buchbinder et al. [2009] for details on the derivation of the algorithm.

Algorithm 4 describes our approach of incrementally building the library $\mathcal{L}$. We maintain a membership vector x_i for every planner $\mathcal{P}_i \in \mathcal{F}$. This membership is initialized to $\frac{1}{M}$. At iteration i, we receive a planning problem Γ_i. If the existing library cannot solve this problem, we cycle through other planners in the family $\mathcal{F}$. We create a relevant set $\mathcal{R}$ of planners that can solve the problem. The membership variables of this relevant set is doubled till the sum exceeds 1. Planners from the relevant set are added to the library with probability $\frac{\Delta x_r \log N}{1 - x_r \log N}$, where Δx_r is the change in membership during the doubling process.

This algorithm possesses the following performance guarantee

Theorem 5.1 (Near-optimality of Greedy Randomized Rounding)**.** Let L^* be the library size computed by the optimal algorithm. Then the expected library size $|\mathcal{L}|$ computed by the greedy randomized rounding algorithm is

$$\mathbb{E}\left[|\mathcal{L}|\right] \leq (\log N \log M) L^* \tag{5.2}$$

5.5 Predicting a static ensemble

A static ensemble meta-planner takes as input a planning problem distribution $P(\Gamma)$ and outputs a fixed ensemble $\mathcal{E}$ at every decision step. The motivation for a static ensemble is design simplicity and easier verification. For now, we will assume that there exists a static ensemble of expert planners that will match our desired performance requirement. We will discuss later in Section 5.6 when this might not be the case.

There are two paradigms in which this can be addressed - one where the library is sufficiently small such that all the planners in the library can be evaluated on all planning problems in the database. Another where the library is very large and a lazy approach is required to choose which planner to evaluate on the database.

5.5.1 Greedy selection of a static ensemble

We now formulate the ensemble selection problem when all planners in the library can be evaluated on the planning problem database. This will make the abstract Problem 4 concrete by requiring the policy π to be a static ensemble $\mathcal{E}$.

We define the notion of loss of predicting a planner $\mathcal{P}_i \in \mathcal{L}$ as a relative cost.

$$l\left(\Gamma, \mathcal{P}_i\right) = J\left(\Gamma, \mathcal{P}_i\right) - \min_{\mathcal{P} \in \mathcal{L}} J\left(\Gamma, \mathcal{P}\right) \tag{5.3}$$

The risk R is defined as an expected loss

$$R\left(\mathcal{P}_i\right) = \mathbb{E}_{P(\Gamma)}\left[l\left(\Gamma, \mathcal{P}_i\right)\right] \tag{5.4}$$

Let $\mathcal{E} \subseteq \mathcal{L}$ be an ensemble of expert planners of budget $|\mathcal{E}| = B$. Thus loss of an ensemble is

$$l\left(\Gamma, \mathcal{E}\right) = \min_{\mathcal{P}_i \in \mathcal{E}} J\left(\Gamma, \mathcal{P}_i\right) - \min_{\mathcal{P} \in \mathcal{L}} J\left(\Gamma, \mathcal{P}\right) \tag{5.5}$$

The risk is $R\left(\mathcal{E}\right) = \mathbb{E}_{P(\Gamma)}\left[l\left(\Gamma, \mathcal{E}\right)\right]$. The objective can be stated as minimizing an empirical risk

$$\begin{aligned} \widehat{R}\left(\mathcal{E}\right) &= \frac{1}{N} \sum_{i=1}^{N} l\left(\Gamma_i, \mathcal{E}\right) \\ \mathcal{E}^* &= \arg\min_{\mathcal{E} \subseteq \mathcal{L}} \widehat{R}\left(\mathcal{E}\right) \end{aligned} \tag{5.6}$$

The optimization problem in (5.6) is a combinatorial optimization problem known to be NP-Hard problem [Krause and Golovin, 2012]. Fortunately the minimization of linear combination on min() functions is equivalent to maximization of a submodular function [Krause and Golovin, 2012]. This allows greedy strategies to have strong guarantees.

Algorithm 5: Greedy Selection of a Static Ensemble $(P(\Gamma), \mathcal{L})$

```
1 Sample planning problem dataset {Γ_i}_{i=1}^N from P(Γ);
2 Evaluate all planners P ∈ L on {Γ_i}_{i=1}^N to compute a cost matrix J̄ ∈ R^{N×|L|};
3 E ← ∅;
4 while |E| < B do
5     Using J̄, select P* ← arg min_{P∈L\E} R̂(E ∪ P);
6     E ← E ∪ P*
7 return E;
```

Algorithm 5 describes the greedy ensemble selection which iteratively picks the planner that minimizes the combined risk the most. Line 1 samples a dataset of planning problems $\{\Gamma_i\}_{i=1}^N$ from the distribution $P(\Gamma)$. Line 2 applies each planner $\mathcal{P} \in \mathcal{L}$ on the dataset $\{\Gamma_i\}_{i=1}^N$ to create a cost matrix $\bar{\mathbf{J}} \in \mathbb{R}^{N \times |\mathcal{L}|}$. Then line 5 is iteratively repeated to compute the planner that minimizes the cumulative risk computed from $\bar{\mathbf{J}}$ and is appended to the selected ensemble $\mathcal{E}$.

This greedy strategy enjoys the following guarantee due to properties of maximization of monotonic submodular functions

Theorem 5.2 (Near-optimality of Greedy Algorithm)**.** Let $\mathcal{E}^*$ be the optimal ensemble that minimizes empirical risk given the cost matrix $\bar{\mathbf{J}}$. Let $L_{\max}$ be the maximum loss of any element on any problem, i.e. $L_{\max} = \max_{\Gamma, \mathcal{P}_i} l\left(\Gamma, \mathcal{P}_i\right)$. The ensemble $\mathcal{E}_{\text{greedy}}$ constructed by Algorithm 5 has the following performance guarantee

$$\widehat{R}\left(\mathcal{E}_{\text{greedy}}\right) \leq \left(1 - \frac{1}{e}\right) \widehat{R}\left(\mathcal{E}^*\right) + \frac{L_{\max}}{e} \tag{5.7}$$

Proof. We construct a non-negative monotone submodular function defined on a set of planners

$$F(\mathcal{E}) = L_{\max} - \widehat{R}(\mathcal{E}) \tag{5.8}$$

This is non-negative because $\widehat{R}(\mathcal{E}) \leq L_{\max}$ by definition. Monotonicity and sub-modularity are straight forward to verify since this is a negative of a min() function. The greedy algorithm has the following property [Krause and Golovin, 2012]

$$\begin{aligned} F(\mathcal{E}) &\geq \left(1 - \frac{1}{e}\right) F(\mathcal{E}^*) \\ L_{\max} - \widehat{R}(\mathcal{E}) &\geq \left(1 - \frac{1}{e}\right) \left(L_{\max} - \widehat{R}(\mathcal{E}^*)\right) \\ \widehat{R}(\mathcal{E}_{\text{greedy}}) &\leq \left(1 - \frac{1}{e}\right) \widehat{R}(\mathcal{E}^*) + \frac{L_{\max}}{e} \end{aligned} \tag{5.9}$$

□

5.5.2 Lazy greedy selection using learnt priors

We now examine the scenario where the library $\mathcal{L}$ can be very large. In this scenario, evaluating all planners in the library every time the target distribution changes might not be tractable. We can avail of the side information obtained from planning problem in the form of context. This context can be used to train priors that predict the performance of a planner. This prior can be used to decide which planners to evaluate.

Algorithm 6: Lazy Greedy Selection of a Static Ensemble$(P(\Gamma), \mathcal{L}, \pi)$

```
1  Sample planning problem dataset {Γ_i}_{i=1}^N from P(Γ);
2  Compute feature projections {f_i}_{i=1}^N for each problem;
3  Using {f_i}_{i=1}^N, compute estimated cost matrix J̃ ∈ ℝ^{N×|L|};
4  E ← ∅;
5  while |E| < B do
6      Using J̃, select P* ← argmin_{P∈L\E} R̂(E ∪ P);
7      Evaluate P* on {Γ_i}_{i=1}^N and update J̃;
8      if P* still best then
9          E ← E ∪ P*;
10 return E;
```

Algorithm 6 presents a lazy greedy selection approach using learnt priors. During the time of library creation, a model π is trained to predict the performance of a planner given context f. Details about how this is done is covered in Section 5.7. It is understood that this prediction might be noisy. Hence this prediction is not purely used for selection. Instead this prior is used to create an estimated cost matrix $\widetilde{\mathbf{J}}$ as shown in Line 4. The algorithm iteratively calls Line 6 to compute the best planner that minimizes risk. This planner is then evaluated on the dataset in Line 7 to update the cost matrix $\widetilde{\mathbf{J}}$. If the evaluation retains the ranking order among the planners, it is added to the ensemble.

An important concern is that the priors are trained on a different distribution than $P(\Gamma)$. This implies that there will be an error in the estimated cost matrix $\widetilde{\mathbf{J}}$ with the actual cost matrix $\bar{\mathbf{J}}$. However, we can still bound the errors due to this mismatch to provide a guarantee for the lazy greedy algorithm.

Theorem 5.3 (Near-optimality of Lazy Greedy Algorithm)**.** Let ε_i be the prediction error of the marginal gain using the prior for slot i. Let $L_{\max}$ be the maximum loss of any element on any problem, i.e. $L_{\max} = \max_{\Gamma,\mathcal{P}_i} l\left(\Gamma, \mathcal{P}_i\right)$. The ensemble $\mathcal{E}_{\text{greedy}}$ constructed by Algorithm 6 has the following performance guarantee

$$\widehat{R}\left(\mathcal{E}_{\text{greedy}}\right) \leq \left(1 - \frac{1}{e}\right) \widehat{R}\left(\mathcal{E}^*\right) + \frac{L_{\max}}{e} + \sum_{i=1}^{B} \varepsilon_i \tag{5.10}$$

Proof. We construct a non-negative monotone submodular function defined on a set of planners

$$F(\mathcal{E}) = L_{\max} - \widehat{R}\left(\mathcal{E}\right) \tag{5.11}$$

Streeter and Golovin [2009] bounds the performance of a noisy greedy algorithm that makes an additive error of ε_i at every iteration as follows

$$F(\mathcal{E}) \geq \left(1 - \frac{1}{e}\right) F(\mathcal{E}^*) - \sum_{i=1}^{B} \varepsilon_i \tag{5.12}$$

Combining (5.11) and (5.12) we have the proof. □

5.6 Predicting a dynamic ensemble

In Section 5.5, we discussed the procedure of computing a static ensemble meta-planner. This served as an adaptive analogue of manually designing an ensemble as discussed in Chapter 4. However, there are certain situations where a static ensemble does not meet a desired performance criteria.

Consider the situation where the library of expert planners are all trajectory optimization algorithms with different initial seed trajectories. It is unlikely that there exist a static ensemble of such algorithms that could perform well on a target distribution of planning problems. In such a situation, conditioning on the slightest amount of context would help narrow down choices. Hence, we are motivated to use context extracted from the planning problem to choose an ensemble.

To understand this more generally, let us revisit the discussion between precision planners and general purpose planners that we started in Chapter 4. As the planners become more and more precise to achieve desired solution qualities, such as trajectory optimization algorithms or algorithms that focus the implicit graph in a particular volume, the need for having a dynamic ensemble becomes stronger. As we will see in the experiment evaluation (Section 5.7), there are many such applications where this becomes a necessity.

A dynamic ensemble meta-planner takes as input a planning problem distribution $P(\Gamma)$ and outputs an ensemble $\mathcal{E}$ that can vary from decision step to decision step. A straight-forward way

to achieve this is to compute a policy that maps from the space of context f extracted from a planning problem Γ to the the space of ensembles as shown in Fig. 5.4. Before we delve into predicting ensembles from a context, we introduce the notion of a policy that predicts a single planner from a given context.

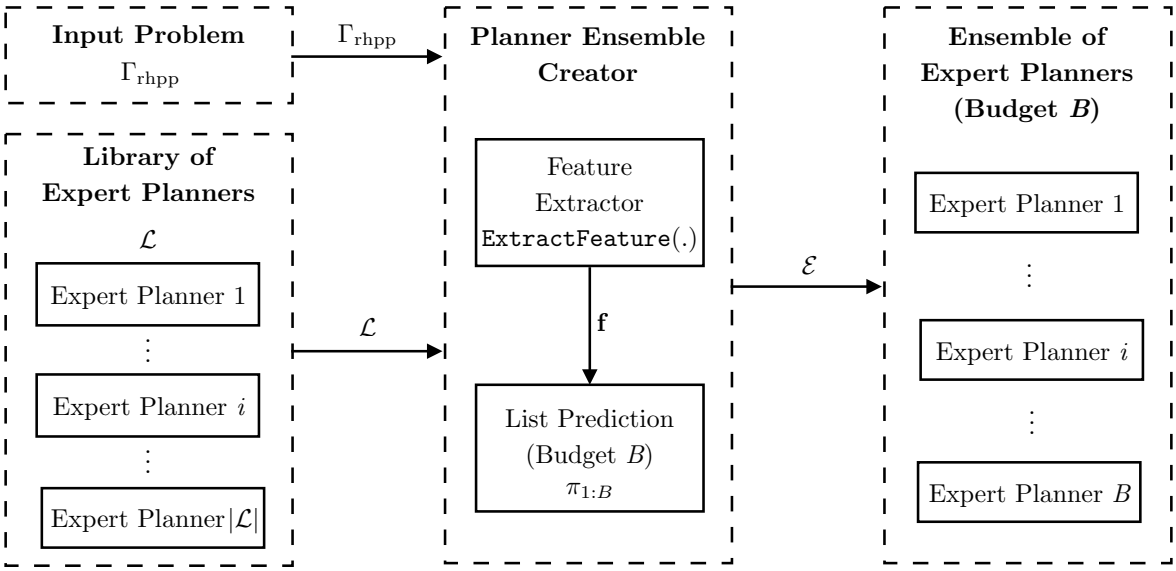


Figure 5.4: Framework for dynamic ensemble predictor. The input is the planning problem and a library of planners. The module extracts context, and asks a list of predictors to predict B expert planners.

5.6.1 Predicting a single planner from context

Let $\pi \in \Pi$ be a predictor that maps a planning problem Γ to a planner in the library $\mathcal{P}_i \in \mathcal{L}$. The loss of a predictor is

$$l(\Gamma, \pi) = J(\Gamma, \pi) - \min_{\mathcal{P} \in \mathcal{L}} J(\Gamma, \mathcal{P}) \tag{5.13}$$

The risk R is defined as an expected loss. An optimal predictor π^* minimizes this risk as follows

$$\begin{aligned} R(\pi) &= \mathbb{E}_{P(\Gamma)}[l(\Gamma, \pi)] \\ \pi^* &= \arg\min_{\pi \in \Pi} R(\pi) \end{aligned} \tag{5.14}$$

Observe that predicting one of $|\mathcal{L}|$ planners given planning problem Γ is like classifying Γ into one of L classes. So π is a multiclass classifier. The 'correct' class is the minimum loss element, but misclassification losses differ. Specifically, finding π is a case of loss-sensitive multiclass classification[1].

The loss l is non-convex, which makes it difficult to directly minimize the empirical risk. The solution approach detailed in Ávila Pires et al. [2013], which we follow, is to replace l with a

[1]Called *cost*-sensitive classification in the learning literature. Since we use costs to refer to element costs, we use the term *loss*-sensitive classification.

convex function, known as the surrogate loss. This is a well-known procedure in learning. To recall the simple case of binary classification, the $0-1$ loss is also non-convex. Binary classification is solved by replacing the loss with a convex surrogate. The step here can be thought of as a more general version of using surrogates. As per the analysis in Ávila Pires et al. [2013], minimizing the convex surrogate risk implies minimizing the true risk. Using the surrogate requires the predictor π to be defined in terms of a scoring function. Let $s(\Gamma, \mathcal{P}) \in \mathbb{R}$ be a function which assigns a score to a planner in the library $\mathcal{P}$ for the planning problem Γ. The predictor then picks the element with the highest score

$$\pi(\Gamma) = \arg\max_j s(\Gamma, \mathcal{P}_j) \tag{5.15}$$

Risk will be minimized in terms of the scoring function s, and therefore π. The predictor space Π also depends on the space of scoring functions. For any environment, the scores are required to be centered over the library elements. Intuitively, this means that the scores are required to be well-behaved. Otherwise, we would be free to assign arbitrarily high scores to low-loss elements, or arbitrarily low scores to high-loss elements. This constraint is expressed as

$$\sum_{j=1}^{|\mathcal{L}|} s(\Gamma, \mathcal{P}_j) = 0, \forall\, \Gamma \tag{5.16}$$

The last ingredient is $\psi(t) \in \mathbb{R}$, a convex function of t. The surrogate loss is then defined as

$$l_\psi(\Gamma, s) = \sum_{j=1}^{|\mathcal{L}|} l(\Gamma, \mathcal{P}_j)\psi(s(\Gamma, \mathcal{P}_j)) \tag{5.17}$$

Note that the scores $s(\Gamma, \mathcal{P})$ don't enter the loss l_ψ directly, but through the convex function ψ. The surrogate loss is large when either the true loss $l(\Gamma, \mathcal{P}_j)$ is large, or $\psi(s(\Gamma, \mathcal{P}_j))$ is large. Intuitively, minimizing this loss encourages high scores to be assigned to the low-loss elements. The empirical surrogate risk is defined in terms of the surrogate loss. With the following shorthand: $l_{ij} = l(\Gamma_i, \mathcal{P}_j)$, $s_{ij} = s(\Gamma_i, \mathcal{P}_j)$, the transformed optimization problem is

$$\hat{R}_\psi(s) = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{|\mathcal{L}|} l_{ij}\psi(s_{ij}) \tag{5.18a}$$

$$s = \arg\min_{\tilde{s}}\ \hat{R}_\psi(\tilde{s}) \tag{5.18b}$$

We detail the choices of the convex surrogate $\psi(t)$, scorer $s(\Gamma, \mathcal{P})$, and the optimization scheme in Tallavajhula and Choudhury [2015].

5.6.2 Predicting an ensemble from context

We frame the problem of predicting an ensemble of planners from context in the framework of list prediction using a list of single planner predictors - one for each slot in the ensemble. An algorithm for list prediction, ConseqOpt, was presented by Dey et al. [2013]. We broadly follow their treatment. Instead of a single predictor, we now want to find a list of predictors. The length of the list, or budget B, is fixed and decided by computational resources. The list of

predictors is denoted by $\langle \pi^1 \colon \pi^B \rangle$. The loss in (5.13) is extended to take a list of predictors as argument

$$l\left(\langle \pi^1 \colon \pi^B \rangle\right) = \min_{\pi^k \in \langle \pi^1 \colon \pi^B \rangle} J\left(\Gamma, \pi^k(\Gamma)\right) - \min_{\mathcal{P} \in \mathcal{L}} J(\Gamma, \mathcal{P}) \tag{5.19}$$

The risk of a list of predictors is the expected loss $R\left(\langle \pi^1 \colon \pi^B \rangle\right) = \mathbb{E}_{P(\Gamma)}\left[l\left(\Gamma, \langle \pi^1 \colon \pi^B \rangle\right)\right]$. The increased power of a list of predictors over a single predictor may seem to be offset by the difficult task of finding the optimal list. The optimal list requires a search over all lists of length B

$$\langle \pi^{1*} \colon \pi^{B*} \rangle = \arg \min_{\langle \pi^1 \colon \pi^B \rangle \in \Pi^B} R\left(\langle \pi^1 \colon \pi^B \rangle\right) \tag{5.20}$$

However, the risk is a monotone supermodular function. Refer to Krause and Golovin [2012] and Dey et al. [2013] for details regarding such functions. The implication of this property is that there exists a simple algorithm for finding a near-optimal list: greedy selection. Selecting the first predictor π^1 is exactly the minimization in (5.14). The second predictor is selected as $\pi^2 = \min_{\tilde{\pi} \in \Pi} R(\langle \pi^1, \tilde{\pi} \rangle)$, or minimizing the risk after fixing the predictor at the first level. In general, the greedy procedure is

$$\begin{aligned} R^k(\pi) &= R\left(\langle \pi^1 : \pi^{k-1}, \pi \rangle\right) \\ \pi^k &= \arg\min_{\pi \in \Pi} R^k(\pi),\ k = 1 \colon B \end{aligned} \tag{5.21}$$

We refer to $R^k(\pi)$ as the risk at level k. It is a function of $\langle \pi^1 \colon \pi^{k-1} \rangle$. We also use the shorthand $l_{ij}^k = l(\Gamma_i, \langle \pi^1 \colon \pi^{k-1}, \mathcal{P}_j \rangle)$ for the losses at the level k. They calculate the loss of planner $\mathcal{P}_j$, given the list of predictors $\langle \pi^1 \colon \pi^{k-1} \rangle$. For example, if one of the earlier predictors has already predicted the lowest loss element for planning problem Γ_i, then it does not matter which element is predicted at level k, and all the marginal losses will be zero, $l_{ij}^k = 0\ \forall j$. We again use the convex relaxation, optimizing the empirical surrogate risks at each level. The predictor π^k is defined in terms of a scoring function s^k at that level

$$\hat{R}_\psi^k(s) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{L} l_{ij}^k \psi(s_{ij}) \tag{5.22a}$$

$$s^k = \arg\min_{\tilde{s}} R_\psi^k(\tilde{s}) \tag{5.22b}$$

$$\pi^k(\Gamma) = \arg\max_j s^k(\Gamma, \mathcal{P}_j),\ k = 1 \colon B \tag{5.22c}$$

We close this section with an observation about the risks. If Ω is the space of environments, let $\Omega|\neg 1$ be the space of environments where loss is non-zero under π^1. Equivalently, $\Omega|\neg 1$ is the space of unsolved environments at level 2. More generally, $\Omega|\neg 1 \colon \neg(k-1)$ is the space of environments where loss is nonzero under the list $\langle \pi^1 \colon , \pi^{k-1} \rangle$, or the space of unsolved environments at level k. Then

$$\begin{aligned} R(\langle \pi^1 \colon \pi^{k-1}, \pi \rangle) &= \int_\Omega l(\Gamma, \langle \pi^1 \colon \pi^{k-1}, \pi \rangle) p(\Gamma) d\Gamma \\ &= \int_{\Omega|\neg 1 \colon \neg(k-1)} l(\Gamma, \langle \pi^1 \colon \pi^{k-1}, \pi \rangle) p(\Gamma) d\Gamma \end{aligned}$$

Algorithm 7: List Prediction for Dynamic Ensemble

Sample planning problem dataset $\{\Gamma_i\}_{i=1}^N$ from $P(\Gamma)$;
Compute feature projections $\{f_i\}_{i=1}^N$ for each problem;
Evaluate all planners $\mathcal{P} \in \mathcal{L}$ on $\{\Gamma_i\}_{i=1}^N$ to compute a cost matrix $\bar{\mathbf{J}} \in \mathbb{R}^{N \times |\mathcal{L}|}$;
Create loss matrix $\bar{\mathbf{L}}$ from $\bar{\mathbf{J}}$;
for $k \in \{1, \ldots, B\}$ **do**
    Train a predictor π^k to do loss sensitive classification using features $\{f_i\}_{i=1}^N$ and loss matrix $\bar{\mathbf{L}}$;
    Use the trained predictor π^k to update loss matrix $\bar{\mathbf{L}}$;
return $\langle \pi^1 : \pi^B \rangle$;

It is only the space of unsolved environments that appear in the risk at level k. The environments already correctly classified are of no concern.

In other words, the predictors are trained sequentially. Importantly, they focus on different planning problems. The second predictor will focus on the planning problems where the first predictor had low performance. The third will focus on planning problems where the first two had low performance, and so on.

5.7 Experimental evaluation

In this section, we present empirical evaluations of both static and dynamic ensemble meta-planners. We choose a spectrum of planning applications such as selecting from a library of initial seeds for trajectory optimization, a library of heuristics for 4D arm planning and a library of planners for UAV path planning. These results serve as concrete realizations of Challenge 2 in different settings and demonstrates the distinct advantage of an adaptive meta-planner scheduling effective black-box planners. In addition, the results also provide insight into the nature of planning problems and how they affect the selection of the ensemble. Our implementation is open sourced (`https://bitbucket.org/sanjiban/list_prediction_motion_planning`).

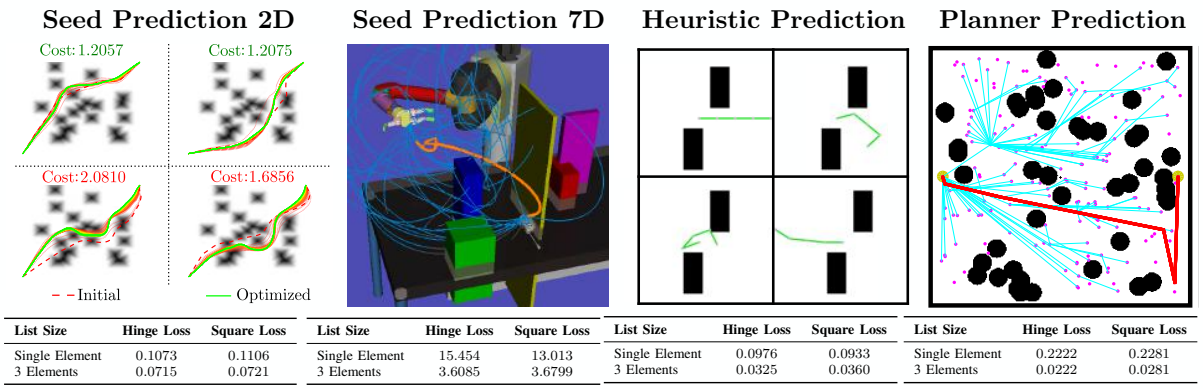
5.7.1 Dynamic ensemble

We evaluate the dynamic ensemble meta-planner presented in Algorithm 7 on 4 planning problem datasets, each obtained from different applications, as shown in Fig. 5.5. The dataset details are presented in Fig. 5.5(a). Empirical risks for all applications are shown in Fig. 5.5(b). We observe that the risk of a list is significantly lower than the risk of a single element, irrespective of the application. In addition, the risk is lower when using the hinge surrogate loss compared to the square surrogate loss. Fig. 5.5(c - f) shows sample lists predicted for each application.

We will go through each of these applications, explain the motivation and analyze the results. For further details about dataset collection, context extraction and learner parameters, we refer the reader to Tallavajhula and Choudhury [2015].

Application	Library Length L	List Budget B	Feature Dimension d	Train Data N	Validation Data	Test Data
Seed Prediction 2D	39	3	73	700	200	100
Seed Prediction 7D	30	3	17	310	112	100
Heuristic Prediction	101	3	1620	675	193	96
Planner Prediction	100	3	1764	579	166	82

(a)



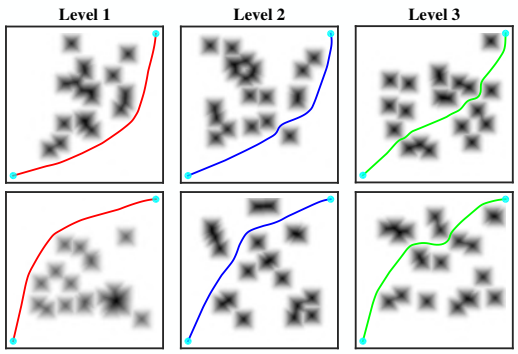
List Size	Hinge Loss	Square Loss
Single Element	0.1073	0.1106
3 Elements	0.0715	0.0721

List Size	Hinge Loss	Square Loss
Single Element	15.454	13.013
3 Elements	3.6085	3.6799

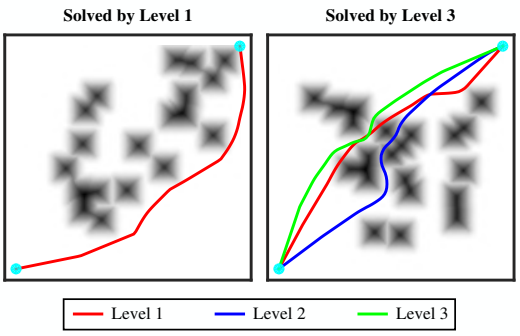
List Size	Hinge Loss	Square Loss
Single Element	0.0976	0.0933
3 Elements	0.0325	0.0360

List Size	Hinge Loss	Square Loss
Single Element	0.2222	0.2281
3 Elements	0.0222	0.0281

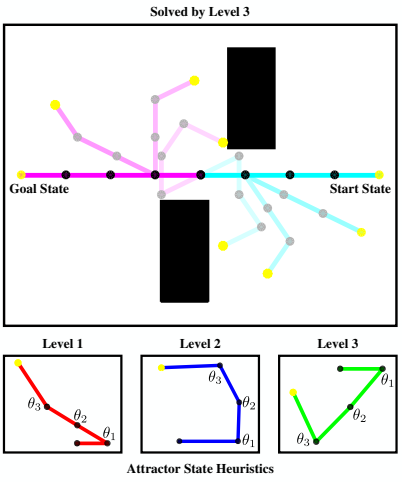
(b)



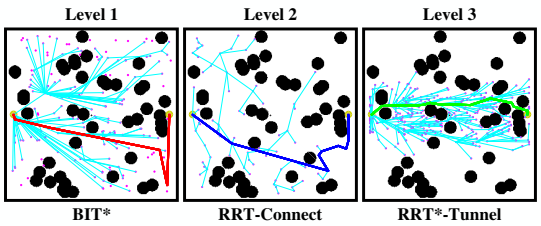
(c)



(d)



(e)



(f)

Figure 5.5: Evaluation of dynamic ensemble meta-planner on a spectrum of motion planner applications - Choosing seed trajectory for 2D trajectory optimization, 7D trajectory optimization, heuristic prediction for 4D arm planning and planner prediction for 2D system. (a) Details about the dataset for each application (b) Illustrations of the 4 applications and the average loss of the meta-planner for different ensemble sizes and surrogate losses (c) Training phase - the examples shown are problems solved by predictors at different levels and the trajectories shown are post-optimization. (d) Test phase - the environment on the left is easily solved by the level 1 predictor (red) while the environment on the right is solved only by level 3 predictor (right). (e) The predictor at level 3 predicts a heuristic (green) that tucks the arm in a non-trivial configuration that allows it to pass through the gap. (f) The predictor at level 3 predicts RRT*-Tunnel, as it concentrates sampling in a tunnel around the initial straightline solution, and finds a path through the gap in the wall.

Seed prediction for trajectory optimization of 2D point robot

For the problem of planning a trajectory from a start to a goal configuration, local trajectory optimization is an approach where an initial seed joining the start to goal is optimized. While these methods are fast, their solution quality is heavily dependent on the initial seed. A library of initial seeds is created, where the cost of a seed is the cost of the trajectory after optimizing the seed using an optimizer, CHOMP [Ratliff et al., 2009a]. The environment consists of square obstacles generated from a uniform random distribution. The distribution places large probability mass on environments where obstacles are clustered. The predictor at the first level predicts simple seeds that go around these clusters. But there are environments which require the optimal seed to be in a complicated homotopy class. These are infrequent, so they are ignored by the first predictor. Subsequent predictors focus on these environments, making customized predictions. Fig. 5.5(c),(d) make this point with specific examples.

Seed prediction for trajectory optimization of 7D robot arm

This is same as 2D seed prediction except in higher dimensions. The dataset used is obtained from Dey et al. [2013].

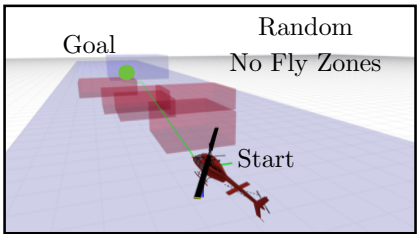
Heuristic prediction for 4D arm planning

Heuristics are essential to improving the runtime performance of search based planning. Recent approaches such as MHA* [Aine et al., 2016] create a framework that allows the use of an inadmissible heuristic as long as it is anchored by weighted A*. An element is an inadmissible heuristic. A heuristic is an exponential kernel on a chosen state, called an 'attractor state'. The heuristic penalizes arm states away from the attractor state. The objective is to the plan the motion of a 4 link arm from start to goal. The environment consists of two rectangular blocks, one above and one below the arm. The cost is the number of expansions done by the search. Environments frequently have a sufficient gap between the two blocks for the arm to pass through. The predictor at the first level predicts attractor states corresponding to simple arm tucking configurations. Environments in which the blocks are close together, leading to a narrow gap, are infrequent. These environments require a complicated tucking attractor state. The subsequent predictors solve such environments, as seen in Fig. 5.5(e).

Planner prediction for 2D planning

A library of planners is created by trying different algorithms and different parameters. The cost is the quality of the solution after a planning time budget. The objective is to the plan the motion of a 2D point robot from start to goal. The environment consists of circular obstacles. The distribution is such that the positions and radii of the obstacles are sampled uniformly. The first predictor predicts planners such as BIT*, RRT-Connect and Informed-RRT*. These planners do not make strong assumptions about structure in the environment, which results in good performance over a wide range of environments. Environments with structure are infrequent under the distribution. We observe that subsequent predictors predict planners which exploit

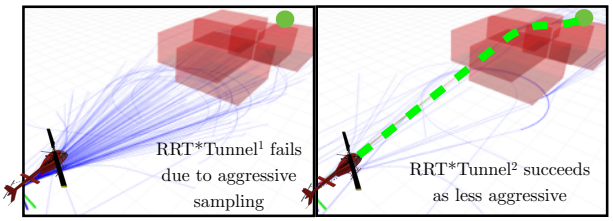
structure as shown in Fig. 5.5(f).



(a)

Learner	Ensemble Budget = 1	Ensemble Budget = 3
Hinge Loss + Linear	0.104	0.035
Square Loss + Linear	0.120	0.055
Unweighted Loss + Random Forest	0.101	0.021

(b)



(c)

Expert Planner	Failure Fraction	Cost (overall)	Cost (if success)
RRT*Uniform[1]	0.786	0.890	0.487
RRT*Workspace[1]	0.213	0.364	0.193
RRT*Workspace[2]	0.181	0.332	0.185
RRT*Tunnel[1]	0.167	0.256	0.107
RRT*Tunnel[2]	0.391	0.432	0.068
BIT*[1]	0.707	0.881	0.596
BIT*[2]	0.948	0.981	0.636
RRTConnect[1]	0.186	0.876	0.848
T-RRT[1]	0.248	0.744	0.660
T-RRT[2]	0.235	0.732	0.650
LBT-RRT[1]	0.925	0.963	0.506
LBT-RRT[2]	0.941	0.970	0.496
InformedRRT*[1]	0.996	0.996	0.000
STRIDE[1]	0.668	0.893	0.677
EST[1]	0.352	0.854	0.775
FixedDescent[1]	0.694	0.743	0.161
FixedDescent[2]	0.946	0.964	0.339
FixedDescent[3]	0.973	0.973	0.009
FixedDescent[4]	0.983	0.983	0.000
FixedDescent[5]	0.987	0.992	0.403
LatPrim[1]	0.223	0.448	0.289

(d)

Figure 5.6: Evaluation of dynamic ensemble selection on a dataset of planning problems encountered by an autonomous helicopter (a) The dataset is collected by randomly placing no fly zones and a goal point at a horizon (b) The 21 planners used in the experiment (refer to Appendix C) for details. The statistics of the planner on the dataset are fraction of times they fail, their average cost, and the average conditional cost. (c) Performance of different models for learning an ensemble with 1 and 3 elements (d) A scenario where the first predictor selects `RRT*Tunnel`[1] which fails to solve the problem due to sampling too aggressively. The second predictor selects `RRT*Tunnel`[2] which is able to solve the problem.

Planner prediction for autonomous helicopter planning

We now evaluate the dynamic ensemble selection approach on a dataset of planning problems encountered by an autonomous helicopter. The difference between the dataset collected for the static ensemble evaluation is that the planning problems are harder, requiring avoidance maneuvers in x,y, and z. The speed of the vehicle is higher, thus making curvature constraints more relevant. For this dataset, a static ensemble does not suffice. First a library of expert planners is created as shown in Fig. 5.6(b). Refer to Appendix C for details. A database of 1000 planning problems are created by placing 5 random no fly zones in a stretch of 2km as shown in Fig. 5.6(a). As we can see from Fig. 5.6(c), the dynamic ensemble brings down the risk dramatically from dynamically selecting a single planner. We also note that we had to resort to a more powerful model class, random forest, to drive the risk down to tolerable thresholds.

5.7.2 Static ensemble

We present evaluation of the lazy greedy selection procedure of a static ensemble presented in Algorithm 6. Before delving into the results we introduce a transformation applied to the problem to aid the learning process. Instead of regressing from context f to cost J, we introduce two metrics - *feasibility* and *performance*. *Feasibility* is a binary classification of whether a planner can solve a planning problem. *Performance* is a number in the range $[0, 1]$ that indicates the suboptimality of the solution produced by a planner conditioned on whether it could solve the problem. It is computed as $V = e^{-J/\eta}$, where η is a normalization constant. Thus for each planner, learning a prior consists of learning a classification model to feasibility and a regression model to performance. The metric *score* is then a product of feasibility and performance. The rationale for applying this transformation is that the feasibility function is often easier to learn that regressing to cost directly. Additionally, the performance function is also easier than the cost as only the examples where a feasible solution was found by the planner are examined.

We first evaluate on a small representative dataset of 2D planning problems with holonomic constraints as presented in Fig. 5.7. The planning problem distribution is a mixture distribution from 4 underlying distributions - Forest, Baffle, Gate and Corridor as shown in Fig. 5.7(a). Each of these distributions correspond to spatial arrangement of obstacles to favour different search strategies. We design 4 expert planners - CHOMP, RRT*-Tunnel, RRT* and RRT*-Obs - the details of which are specified in Fig. 5.7(b). These planners are selected to have orthogonal performance, i.e., each one dominates in at least one kind of environment. The context is represented by a feature vector of histogram of gradients at different resolutions. The learnt priors use a chi-square kernel on this feature space. Fig. 5.7(b) shows the accuracy of the prior on predicting feasibility and MSE of the prior in predicting performance. Fig. 5.7(c) shows the ensemble selection procedure applied on two target distributions. The predicted static ensembles also match the best ensemble arrived at using brute force combinatorial search. This result shows that the lazy greedy search is able to adapt to any target distribution to pick a combination of diverse planners.

We next evaluate the performance of the algorithm on a dataset of planning problems for an autonomous helicopter. First a library of expert planners is created (Fig. 5.8(f)). A dataset of 10000 planning problems is created by permutations and combinations of various scenarios that may favor one planning approach over another. Fig. 5.8(a) shows some samples from this dataset. The features used here are - coarse connectivity of workspace and a local dense distance field gradient. The priors are learnt using a random forest classification and regression. The two target planning problem distributions we wish to test are “Forest” and “City”. There is a further categorization of “Known” (where the full environment is accessible Fig. 5.8(b),(c)) and “Unknown” (where the environment is incrementally revealed, Fig. 5.8(d),(e)).

To clearly illustrate the ensemble selection process, Fig. 5.8(f) shows every step. Along with the priors, the actual value on evaluation is shown in brackets. In this case, RRT*-Tunnel and CHOMP were selected as the two planners. The lazy evaluation corrects for error in the estimation of the performance of CHOMP. The reason these two planners performed well is because the optimal solution was always within a tunnel of the initial guess and the cost function was usually convex around the initial guess. For the city application, we see in Fig. 5.8(g) that

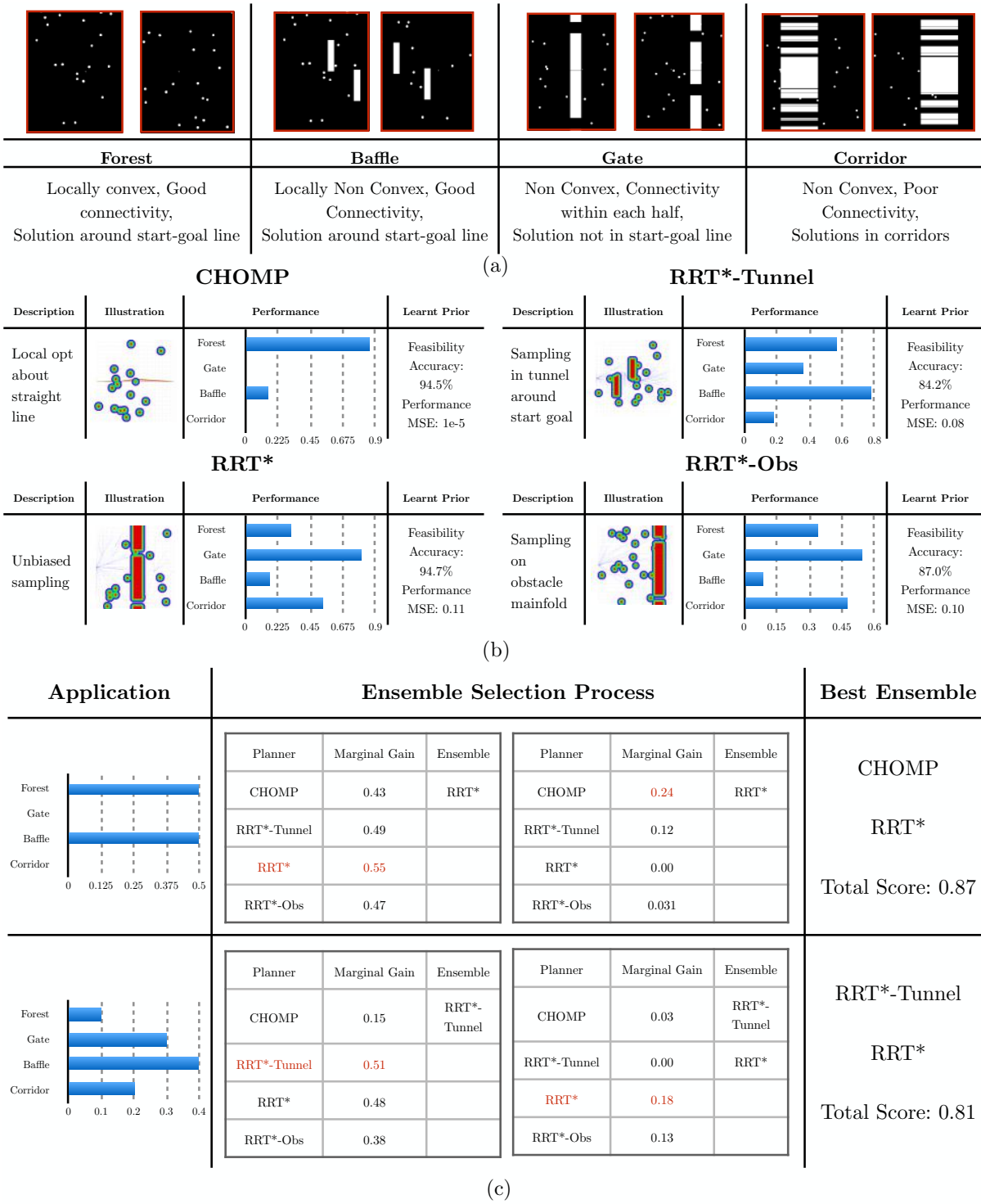
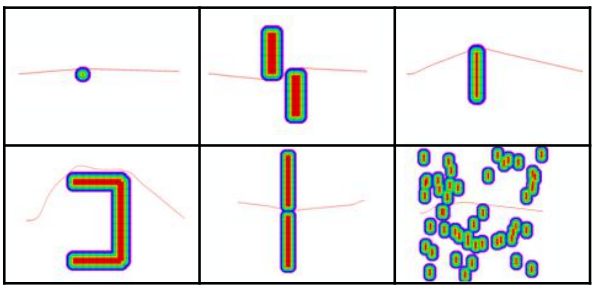
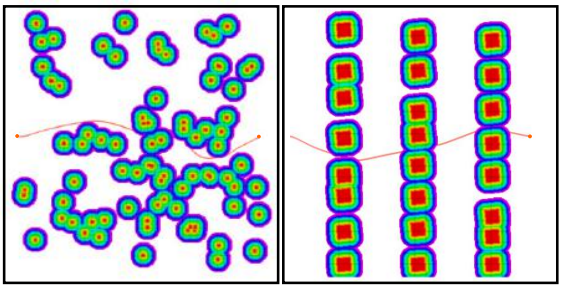


Figure 5.7: Evaluation of lazy greedy static ensemble selection on a representative dataset. (a) The 4 category of planning problems and their properties (b) The 4 planners, their performance on these problems and the accuracy of the learnt priors (c) The ensemble selection process that uses the learnt priors to adapt to 2 target distributions.



(a)

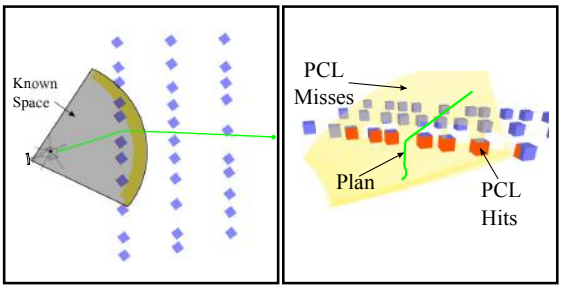


(b) (c)

Planners	Iteration 1 Prior (Eval)	Iteration 2 Prior (Eval)	Iteration 3 Prior (Eval)
A*	0.694	0.694	0.003
CHOMP	**0.989** (0.681)	0.681	**0.021** (0.069)
RRT*	0.700	0.700	0
RRT*Tunnel	0.983	**0.983** (0.848)	-
RRT*-Obs	0.483	0.483	0
RRT	0.540	0.540	0
Lazy-RRT	0.563	0.563	0
RRTConnect	0.510	0.510	0
LBTRRT	0.638	0.638	0.006
T-RRT	0.533	0.533	0
EST	0.529	0.529	0
PDST	0.588	0.588	0
KPIECE	0.491	0.491	0
BKPIECE	0.772	0.772	0
LBKPIECE	0.491	0.491	0
STRIDE	0.482	0.482	0
SPARS	0.342	0.342	0
SPARS2	0.330	0.330	0
SBL	0.456	0.456	0.002
PSBL	0.458	0.458	0.004

Ensemble		
Slot 1	**RRT*-Tunnel**	**RRT*-Tunnel**
Slot 2		**CHOMP**

(f)



(d) (e)

Criteria	**Ensemble Slot 1** Planner(Score)	**Ensemble Slot 2** Planner(Score)	**Total** Score
Known Forest	RRT*Tunnel(0.85)	CHOMP(0.68)	0.92
Unknown Forest	RRT*Tunnel(0.72)	CHOMP(0.51)	0.85
Known City	A*(0.68)	RRT*Tunnel(0.59)	0.78
Unknown City	RRT*Tunnel(0.80)	CHOMP(0.54)	0.88

(g)

Figure 5.8: Evaluation of lazy greedy ensemble selection method on dataset collected for planning for an autonomous helicopter (a) A training dataset is created from planning problems belonging to different classes. Each distribution focuses on a particular strength / weakness that planners have. (b) Samples of the distance field from planning problem belonging to "known" category. The red regions are the obstacles and upto the green expansion (50m) is considered to be in collision. "Known Forest" has obstacles uniformly sampled using a Poisson distribution. (c) "Known City" has obstacles arranged in the form of a city block (d) "Unknown" category limits what the robot can see. The point cloud hits and misses correspond to seen obstacles and maximum range returns respectively. As far as the robot can see there is only a front row of obstacles (e) The RRT*-Tunnel is easily able to plan through the first row and is unaware of the other obstacles - thus producing a high quality path (f) Ensemble selection process for "Known Forest" (g) Performance of the selected ensemble v/s individual planners for different application.

A* and RRT*-Tunnel were chosen. This is because this environment required reasoning in a global sense, discretization effects were acceptable and on occasion when the solution happened to lie near the initial guess, the RRT*-Tunnel was the only algorithm able to sample densely enough to get a solution.

For "unknown" environments, the forest results remained unchanged but the city results were completely changed. Since the city is incrementally revealed to the robot, it is for the most part easier, even convex around the initial guess on many occasions. This makes the use of A* wasteful when much higher quality solutions free of discretization effects can be obtained. The ensemble selection system is automatically able to leverage this assumption.

5.8 Closed-loop evaluation of dynamic ensemble on different UAV platforms

We evaluate the dynamic ensemble framework described in Section 5.6 on two distinct UAV platforms.

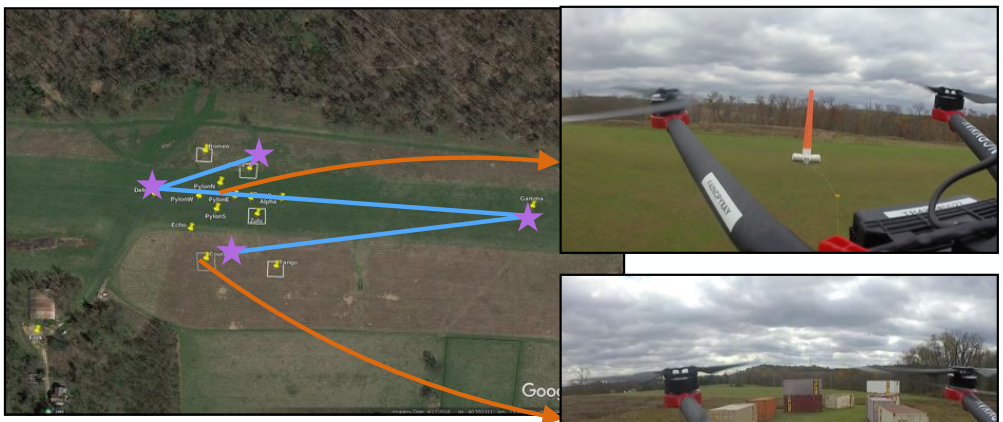
5.8.1 Evaluation on a large scale hexarotor

We first train a dynamic ensemble for planning onboard a large scale hexarotor. The motivation for this application was to ensure robust long term autonomy. The objective was to deploy this UAV in an area with obstacles as shown in Fig 5.9(a) and have it execute as many Monte-carlo missions as possible. Each such mission consists of $4-5$ waypoints chosen quasi-randomly. The hexarotor takes off, follows the mission, touches down and repeats. The hexacopter is shown in Fig 5.9(c) - it is a DJI M600 with two Velodyne VLP-16 which has a 100m range.

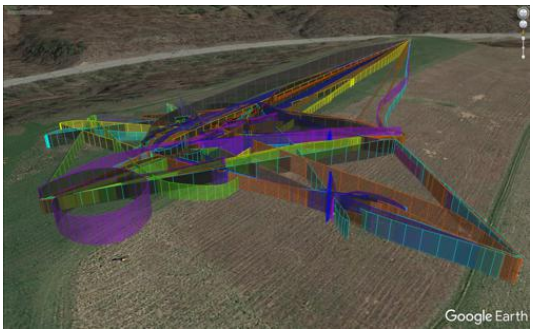
We present flight test results where the adaptive planner enabled the UAV to perform 45 missions for around 151 minutes of flight time as shown in Fig 5.9(b). It covered a total distance of over 15 km reaching top speeds of 10 m/s. The adaptive planning system was successful 95.56% of the time dramatically improving on the performance of a single hand designed planner ($< 50\%$).

Before we describe the flight test result, we shed some light on the training process. A training database of 1000 problems is collected by randomly sampling start and goal points on a hand designed map (resembling the testing area) as shown in Fig 5.9. We use a library of 8 planners from the list of expert planners in Appendix C. We use 4 general purpose planners - `RRT*Tunnel`^{1}, `RRT*Tunnel`^{2}, `BIT*`^{1} and `InformedRRT*`^{1}. We use 4 precision planners - `SingleDetour`^{1}, `SingleDetour`^{2}, `DoubleDetour`^{1} and `DoubleDetour`^{2}. The feature vector we extracted from each problem is 10 dimension containing terms such as distance between start/goal, density of obstacles at various locations, etc. More details on this can be found in Appendix D. We follow the list prediction framework as described in Section 5.6. The ensemble size is set to 2. The list prediction reduces risk to 0.08 from 0.20.

We see some interesting datapoints in the collected dataset that confirms some of the intuitions discussed in Chapter 4 about general purpose and precision planners. Fig 5.10(d) shows a datapoint where both classes of planners are able to have comparable performance. Note how planners such as `BIT*`^{1} or `RRT*Tunnel`^{1} search a large space to find a connection, while the precision planner `DoubleDetour`^{2} sweeps laterally to find a gap. Fig 5.10(e) shows a scenario



(a)



(b)

(c)

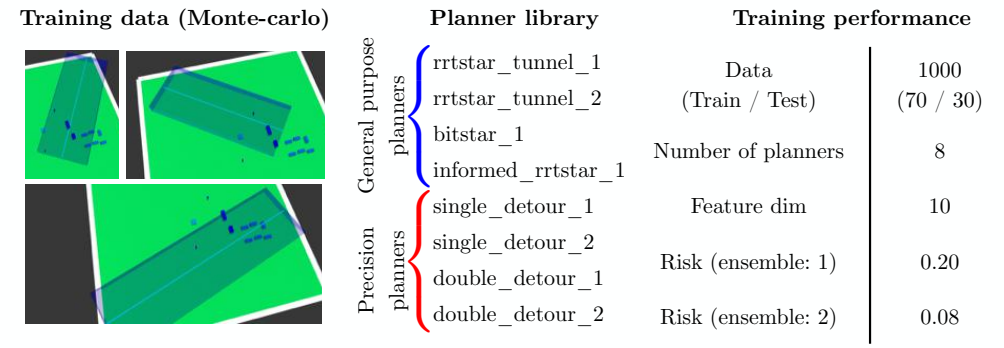
Total Time	Total Distance	Top speed	Missions	Planner success
151.9 min	15.833 km	10 m/s	45	95.56% (improvement over <50% for baseline system)

(d)

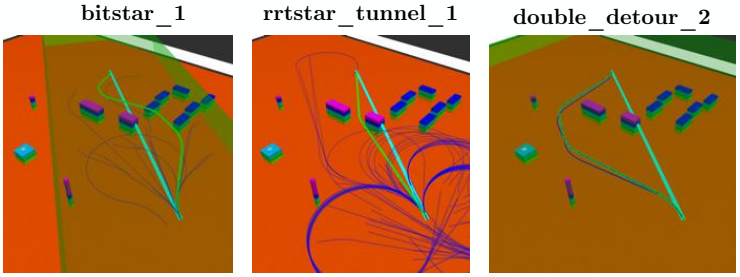
Figure 5.9: Evaluation of adaptive ensemble on a large hexarotor. The purpose of the system is to do long term autonomy. (a) The UAV is flying missions in an area where there are obstacles such as pylons and shipping containers (b) The UAV executes Monte-carlo missions where it repeatedly takes off and lands at different areas (c) The hexarotor is a DJI-M600 with two VLP-16 (d) Summary of the flight test

where such precision planners would completely fail. `RRT*Tunnel`[1] has to plan to climb, then finds a maneuver through a gap that reaches the goal. None of the other planners finds a solution. Fig 5.10(f) shows an opposite scenario where general purpose planners fail to find a path. As can be seen from the attempts by `RRT*Tunnel`[1], the goal point is critically near an obstacle, and the tree is unable to sample a point that can connect to the goal. The precision planner `DoubleDetour`[1] is precisely primed to solve such scenarios by finding the violation point and sampling perpendicular to it till a path is found.

We now analyze a 13 minute segment from the flight test as shown in Fig 5.11. The adaptive



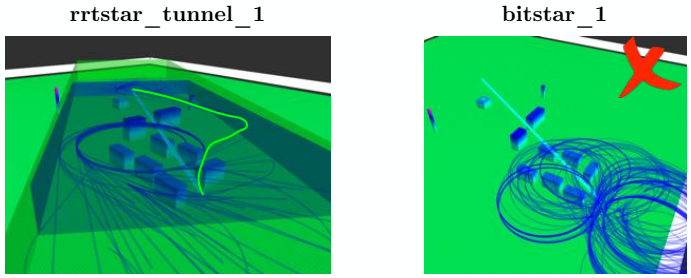
(a) (b) (c)



Planner	Cost
rrtstar_tunnel_1	16.09
rrtstar_tunnel_2	∞
bitstar_1	17.47
informed_rrtstar_1	∞
single_detour_1	∞
single_detour_2	15.87
double_detour_1	∞
double_detour_2	20.49

Example where general purpose planners *match* precision planners

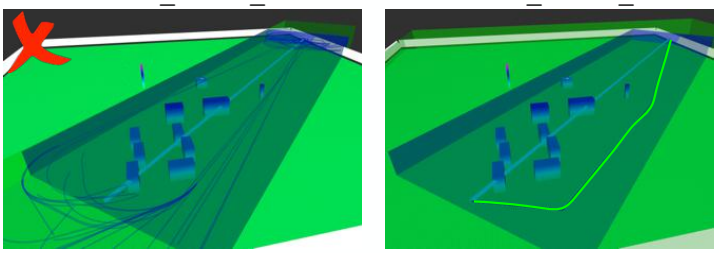
(d)



Planner	Cost
rrtstar_tunnel_1	18.21
rrtstar_tunnel_2	∞
bitstar_1	∞
informed_rrtstar_1	∞
single_detour_1	∞
single_detour_2	∞
double_detour_1	∞
double_detour_2	∞

Example where *only* rrtstar_tunnel_1 solves

(e)



Planner	Cost
rrtstar_tunnel_1	∞
rrtstar_tunnel_2	∞
bitstar_1	∞
informed_rrtstar_1	∞
single_detour_1	∞
single_detour_2	∞
double_detour_1	25.35
double_detour_2	∞

Example where *only* double_detour_1 solves

(f)

Figure 5.10: Details for training the adaptive ensemble meta-planner that was executed during the flight test.

ensemble cycling through different planners at different timesteps is shown in Fig 5.11(a). A zoomed in version is shown in Fig 5.11(b)(c). Interestingly, `RRT*Tunnel`[2] is never selected as

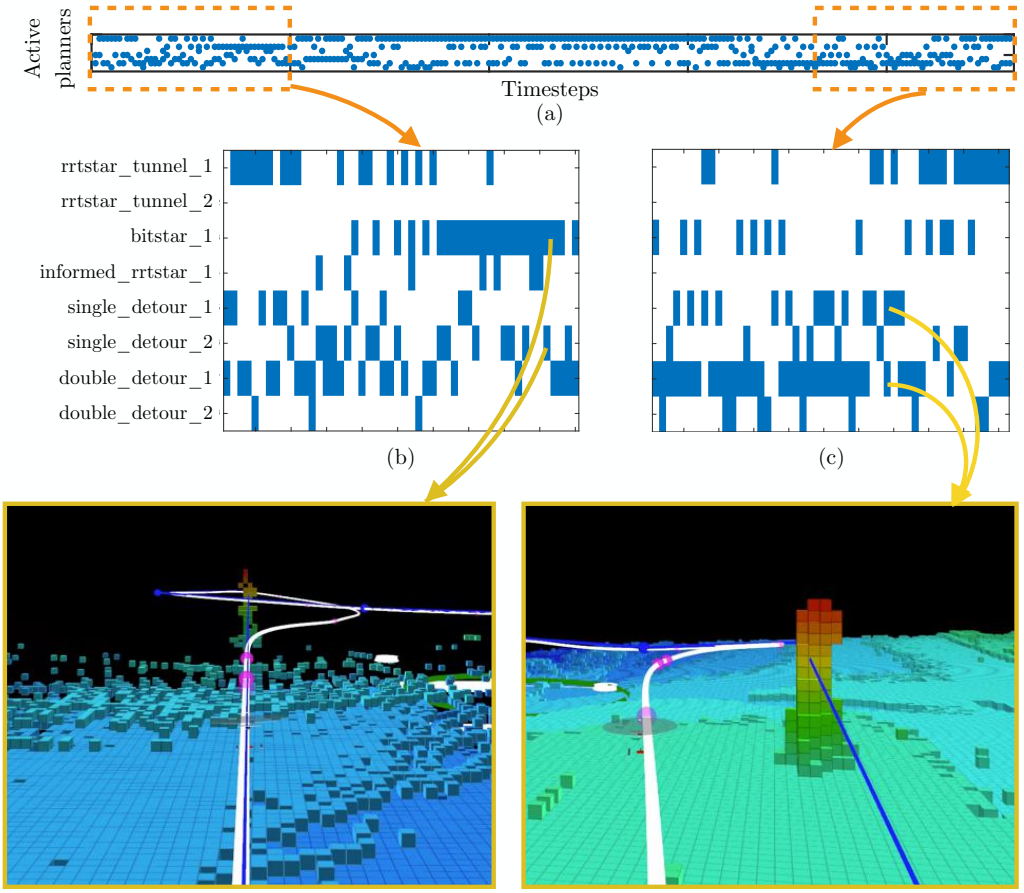


Figure 5.11: Analysis of adaptation of the planner during the flight test. The meta-planner switches between different planners. The active planner are shown by the blue dot.

it was not found to ever dominate another planner during training. Fig 5.11(d) shows a situation where the UAV is asked to circle around a pylon which flying at 10 m/s. Both `BIT*`[1] and `SingleDetour`[1] find a solution that smoothly avoids the pylon with sufficient clearance. On the other hand, Fig 5.11(e) shows a corner case where the goal point is very close to an obstacle. As mentioned before, this creates difficulty for general purpose planners. No general purpose planners are selected for this problem. `SingleDetour`[1] and `DoubleDetour`[1], both precision planners are selected and are able to find a solution.

5.8.2 Evaluation on a small scale quadrotor

We also train a dynamic ensemble for planning onboard a small scale quadrotor. The UAV is tasked with flying missions that involve operating in multiple speed-regimes and navigating in diverse environments as shown in Fig. 5.12 (a). The quadrotor is shown in shown in Fig 5.9(b) -

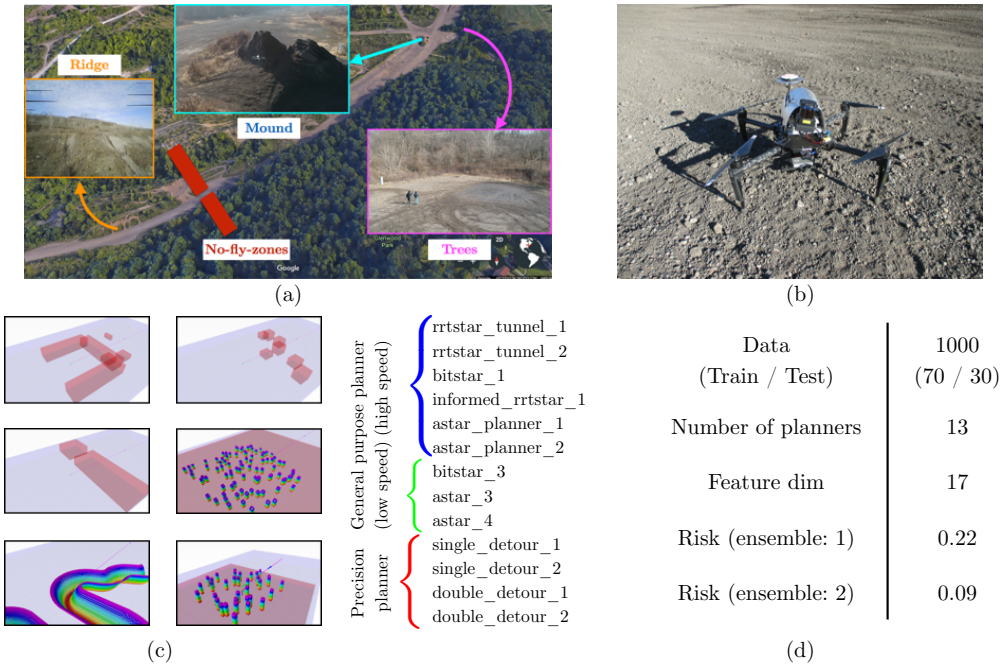


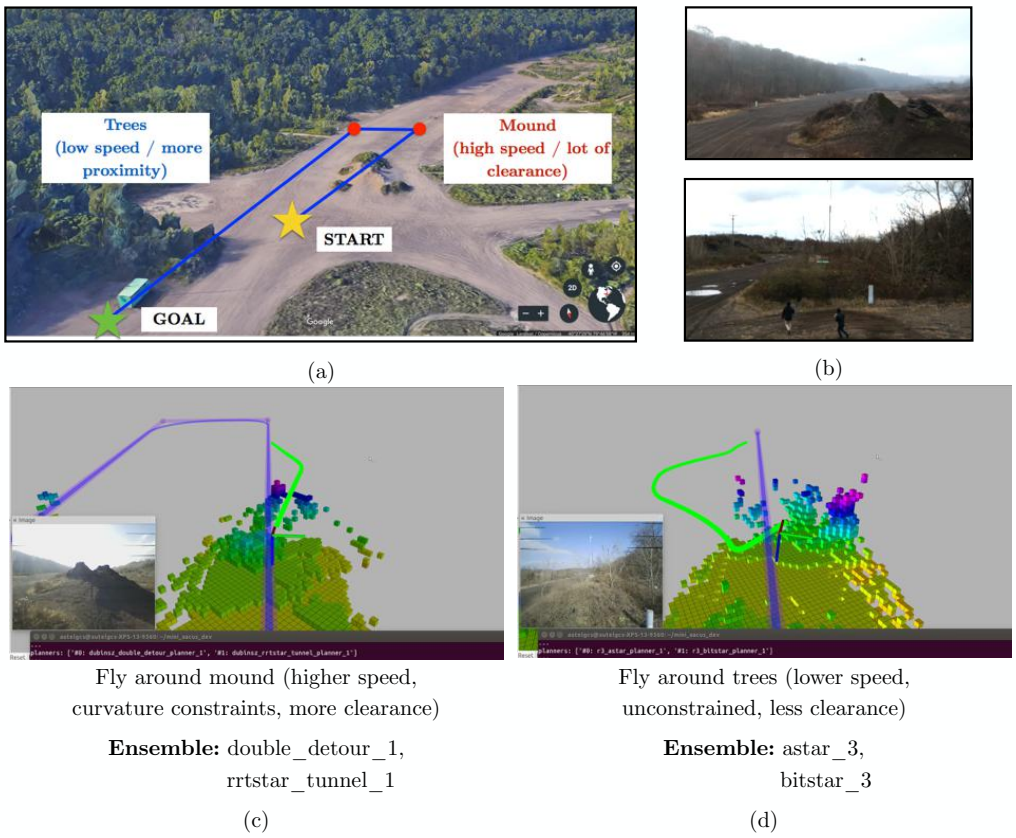
Figure 5.12: Closed loop evaluation on a small quadrotor. (a) Diverse environments in which quadrotor (shown in (b)) operates. (c) Training dataset (d) Library of expert planners

it is a DJI M100 with a spinning Hokyuo laser which has a 15m range. This short range implied the vehicle had to react very quickly to plan around obstacles.

We first describe the training process. We create a family of parametric distribution of environments to simulate the planning problem distribution we want the UAV to operate in. These problems cover the range of high speed and low speed operations. The obstacles arise from two categories - no-fly-zones and terrain. No-fly-zones arise due to presence of other aircrafts, regions in the world where the UAV is not allowed to pass through, etc. Terrain obstacles are trees, hills or ridges. Fig 5.9(c) shows a representative sample from each class of problem. A training database of 6000 problems is collected by randomly 1000 problems from each class.

We use a library of 13 planners from the list of expert planners in Appendix C. We use 6 general purpose planners which operate at high speed - `RRT*Tunnel`^{1}, `RRT*Tunnel`^{2}, `BIT*`^{1}, `InformedRRT*`^{1}, `A*`^{1}, `A*`^{2}. We use 3 general purpose planners which operate at low speed - `BIT*`^{3}, `A*`^{3}, `A*`^{4}. We use 4 precision planners - `SingleDetour`^{1}, `SingleDetour`^{2}, `DoubleDetour`^{1} and `DoubleDetour`^{2}. The feature vector we extracted from each problem is 17 dimensional containing density of obstacles at various locations. More details on this can be found in Appendix D. We follow the list prediction framework as described in Section 5.6. The ensemble size is set to 2. The list prediction reduces risk to 0.09 from 0.22.

We now analyze a test run from the flight test as shown in Fig 5.13(a). The mission requires the UAV to avoid a mound at high speed and trees at a lower speed. Fig 5.13(b) shows the UAV avoiding both obstacles. Fig 5.13(c) shows the instant where the UAV has to avoid the mound. The ensemble selected is `DoubleDetour`^{1} and `RRT*Tunnel`^{1}. Both these planners are able to find



(a) (b)

Fly around mound (higher speed, curvature constraints, more clearance)

Ensemble: double_detour_1, rrtstar_tunnel_1

(c)

Fly around trees (lower speed, unconstrained, less clearance)

Ensemble: astar_3, bitstar_3

(d)

Figure 5.13: (a) Test scenario with two distinct obstacle classes (shown in (b)) which result in different ensembles being selected ((c) and (d)).

a path that bends around the mound. Fig 5.13(d) shows the instant where the UAV has to avoid trees. The ensemble selected is $\texttt{A*}^3$, $\texttt{BIT*}^3$. This is because the UAV is in a cul-de-sac and close to obstacles.

5.9 Discussion and future work

In this Chapter, we took steps towards addressing Challenge 2. Building on the ensemble of expert planners framework in Chapter 4, we formulate the problem as design of meta-planners to select an ensemble from a library of expert planners. We first discuss a static ensemble meta-planner - where a fixed ensemble is selected for a distribution of planning problems. A static ensemble suffices in cases where there are general purpose planners that can have sufficient performance on a wide range of planning problems. We discussed two settings for the static ensemble meta-planner - first where all planners can be evaluated and the second where the library size is too large. While the former case can be efficiently solved with a straightforward greedy approach, we showed how the latter can be tackled by learning priors on the performance of the planners using context from the problem. We then discussed the need for a dynamic ensemble meta-planner where the library consists of precision planners that have high performance on a

small support of problems. We approached this problem by learning a list of predictors that map context extracted from the problem to a planner. Finally, we presented experimental evaluation on a spectrum of motion planning applications along with datasets of planning problems faced by an autonomous helicopter.

We return to the observation made about the accuracy versus complexity tradeoff of context extraction. We observed that models such as graph kernels, which create a coarse graph and matches statistics about all pairs shortest path, have surprisingly high accuracy in modeling planner performance. However, they perform a large amount of computation in terms of collision checking the world and planning using that information. This leads us to ask the question - "why can't planners themselves be used as feature extractors?". This can be interpreted in two ways. Firstly, in the black-box paradigm, we know that planning problems are not i.i.d but have a correlation between consecutive time steps. Hence performance of a planner at time step t contains a rich source of information at time step $t + 1$. Secondly, in the white-box paradigm, as planner proceeds to collision check the environment, it can do inference about the world to adapt its search strategy. This idea is pursued in Chapter 7 and Chapter 8.

We also want to point out the difference between learning priors on planner performance versus classification of which planner to use. The former is a harder problem that is likely to incur higher empirical risk. The latter is an easier problem. However, the problem with classification is that on addition of a new element to the library, the entire process must be repeated. A middle ground between the two frameworks would be an useful area to explore.

We saw that the problems that a predictor i focused on where problems where predictors $1, \ldots, i-1$ failed to solve. We wonder - what should be done with problems that all predictors fail to solve. Even if the predictors are very powerful, we cannot guarantee this situation will not arise. How should this be handled from a systems perspective? We will address this question in Chapter 6.

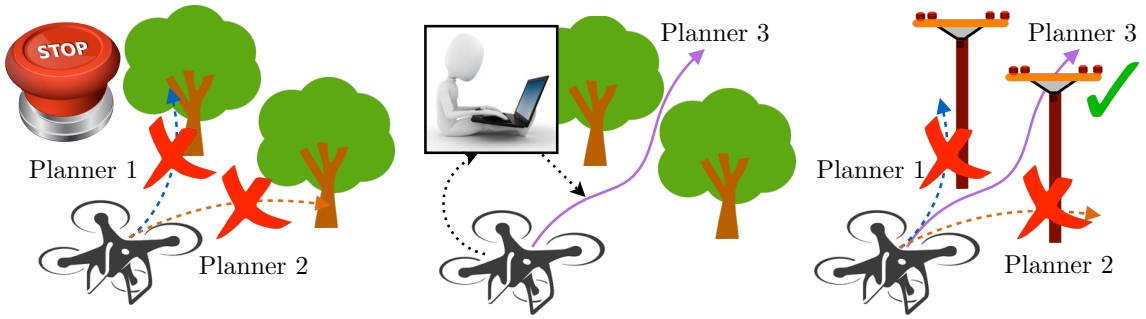
6

Online Exception Planners

In Chapter 5, we discussed approaches for training predictors to select planners from a library. A pertinent question is - what happens when the predicted planner fails to find a path? This can happen because the failure problem at test time might have rarely occurred in the training database. However, at test time, once the robot fails to solve a problem it continues to experience the same problem at the next time step. This phenomenon arises from the fact that the problem distribution is indeed not i.i.d. Additionally, it might also happen that no planner in the library is able to solve the current problem - hence iterating through all planners in the library is not fruitful.

In this Chapter, we view the problem in a different light - *exception handling*. The failure event is viewed as an exception. In such an event the robot may slow down and come to a stop. While it waits, it may ask a more computationally expensive oracle for a planner to specifically solve the failure problem. The oracle can take as long as it likes and returns a planner which the robot executes. However, should the robot encounter the same problem again, we would like it to invoke the computed planner. If the environment consists of only a handful pathological failure scenarios that recur, we envision the robot dynamically constructing an *exception planner library* to deal with such scenarios. We show that the online learning framework of *online paging* offers an elegant way to reason about this problem.

Section 6.1 introduces the problem with a motivating example. Section 6.2 provides a brief background on library construction in robot path planning. Section 6.3 formalizes the exception planner library construction problem in the framework of online paging. Section 6.4 describes the online algorithm. We present some empirical evaluations on different environments in Section 6.5 and conclude with future directions in Section 6.6.



Neither planner can solve. Robot has to e-stop.
(a)

Robot asks oracle for a planner that solves problem
(b)

On re-encountering similar problem, new planner solves
(c)

Figure 6.1: Online exception planners. (a) UAV navigating in the world encounters a difficult problem. Neither planner in its library is able to solve the problem and it has to come to a hover (b) It queries an oracle with the problem. Oracle takes some time and returns an exception planner that can at least solve this exception problem. (c) Next time the UAV encounters a similar problem, the exception planner is able to solve and the UAV does not have to stop.

6.1 Introduction

We begin with a simple example. Consider the scenario shown in Fig. 6.1 where we have a UAV navigating in an environment. The robot is equipped with a library of planners. We assume that the planners are computationally cheap to execute, hence the robot can afford to execute all of them at each time step.

The robot encounters a situation where it has to navigate between two trees as shown in Fig. 6.1(a). Unfortunately, neither of the two planners in its library is able to solve this problem. This forces the robot to execute an emergency stop. The robot then queries an oracle - in this case assume a human designer - with the planning problem that it was not able to solve. The human designer takes some processing time (which is fine since the robot has come to a stop) and comes back with a planner that can solve the problem. The robot accepts this planner into its library. The next time it encounters a similar problem - such as flying between two telephone poles - this new planner is able to solve the problem and the robot no longer has to come to a stop.

We call the event of stopping and querying the oracle an *exception event*. This is because this event disrupts the normal operation of the robot. The planner created by the oracle is called an *exception planner*. Since this planner is created online during the robot's operation, it is called an *online exception planner*.

The role of the online exception planner is to ensure that if the same problem is repeated twice, the robot does not have to stop. However, the robot cannot keep collecting exception planners - the library has a fixed budget due to computation constraints. Hence, with every new exception, old planners have to be removed. The robot has to judiciously decide which planners to keep, and which to throw away. It has to do this while making minimal assumptions about how planning problems arrive. This motivates us to ask the following question

How can we maintain a fixed library of exception planners without having to call the oracle repeatedly?

We show that this framework can indeed generalize to many different settings. We highlight two such instances. The simplest setting is that of receding horizon planning with a trajectory library [Green and Kelly , 2007]. Each planner is simply equivalent to checking a trajectory for validity. The oracle corresponds to checking a dense "mother set" [Dey et al., 2013] and returning the minimum cost trajectory. We use this setting in our experiments since it is easier to verify the behaviour of the library selection algorithm and compare against state-of-the-art baselines.

A more complex real-world setting is where there already exists an adaptive motion planning module. As alluded to in Section 2.2, the distribution of problems encountered is non i.i.d and unlikely to match the collected database. This can result in a situation where the robot encounters a difficult problem, and on failing to solve it, continues to experience the problem. Hence, even if the problem was infrequent in the training distribution, it becomes prominent in the test distribution.

In such a setting, when the robot encounters an exception event, i.e. the adaptive planning module is unable to solve the problem, it queries a library of exception planners. These exception planners are computationally cheap planners which can be trajectory lookups or a trajectory optimization algorithm with different initial seed. If none of the exception planners solve the problem, the UAV stops and invokes a computationally expensive global optimizer. The global optimizer returns a solution which is then added to the library. We want to minimize the number of times the UAV stops, while keeping the library size fixed and making no assumptions about the planning problem distribution.

Our key insight is that this problem can be cast in the paradigm of online paging [Sleator and Tarjan, 1985]. Online paging deals with the problem of deciding which pages to keep in a cache such that the number of cache misses are minimized. Hence the page requests are equivalent to arrival of planning problems, the cache is equivalent to the library of planners. Retrieving a page from the slow memory to the cache is equivalent to querying the oracle for a planner.

This equivalence allows us to import algorithms from online paging which have proven guarantees. We empirically evaluate these algorithms on receding horizon planning problems using a trajectory library and show that they work quite well in practice. Interestingly, they outperform traditionally used approaches for designing trajectory libraries using diversity measures. These results create a bridge between the two disparate fields and lead to interesting future directions.

6.2 Background

In this Chapter, we discuss how to dynamically create a library of planners online. The problem of creating a library of candidate maneuvers or trajectories has been well studied in the domain of grasp selection for manipulation [Berenson et al., 2007, Goldfeder et al., 2009] and receding horizon model predictive control [Green and Kelly , 2007]. Such libraries use efficient techniques to discretize the continuous action space so as to guarantee task performance and satisfy strict computational budget constraints. At runtime, the elements of the library are evaluated until success is achieved. Hence, the performance depends on the runtime content of library.

One of the state-of-the-art methods for selecting such a library for receding horizon planning is by Green and Kelly [2007]. They pose the problem of selecting a small library from a "mother-set" of discretized control space trajectories. The procedure iteratively constructs the library by selecting trajectories that minimize dispersion in trajectory space by using the Haussdorf distance metric. Here the dispersion criteria is used as a surrogate for survivability [Erickson and LaValle, 2009] by ensuring paths are sufficiently diverse.

Arora et al. [2015] create a more direct definition for survivability - the complement of the joint probability of the library being in collision with a point obstacle sampled uniformly at random. They show that this objective is monotone submodular and can be maximized with greedy trajectory selection.

Since both of these methods are not data-driven, the library content do not adapt to the distribution of obstacles the robot actually encounters. This can lead to artifacts where even randomly optimizing such criteria can lead to better performance [Knepper and Mason, 2009]. Dey et al. [2012] cast the problem of library selection in the online learning framework to adapt the library according to the obstacle distribution encountered. They use algorithms that *minimize regret*, i.e. the difference between the choice made by the online algorithm and the best choice in hindsight.

All of these methods assume a "mother-set" is available to make a selection. In this Chapter, we remove this assumption. We too frame our problem in the online learning setup, but focus on problems which are analyzed through *competitive ratio* - the worst case ratio between the online learner's performance and the optimal offline algorithm.

6.3 Problem formulation

We begin by stating the online paging framework. We will subsequently describe the online exception planner formulation in a similar style so as to show the equivalence.

6.3.1 Online paging

Consider a two-level memory divided into pages of fixed uniform size. There is a slow memory that can hold N pages. There is a fast memory (a cache) that can only hold $k < N$ of the pages at a time. Page requests arrive online, one request per time step. If the page request p_t at time t is already in the cache, zero cost is incurred. If p_t is not in the cache, it needs to be brought in. If the cache is full, one of the k pages must be evicted. This incurs a unit cost. This event is known as a cache miss (also a page fault).

We note that this stated model corresponds to the *demand paging* paradigm where a page is not moved out of the fast memory unless room is needed for a newly demanded page.

Let A be an online algorithm. Let z be a page request sequence. Let $\text{cost}(A, z)$ be the number of cache misses incurred by A on the sequence z.

We now define a metric for assessing A - the *competitive ratio*. The competitive ratio of A is its worst case performance to an offline optimal algorithm OPT which has full knowledge of

the sequence z up front:

$$\max_{z} \frac{\text{cost}(A, z)}{\text{cost}(OPT, z)} \tag{6.1}$$

An online algorithm A is deemed better than B iff it has a competitive ratio closer to 1. Note that no assumption is made on the sequence of page requests z.

6.3.2 Online exception planner selection

The robot is navigating using a library $\mathcal{L}$ of planners $\mathcal{P}_i \in \mathcal{L}$. Each of these planners are deemed to be computationally inexpensive. Hence the robot can afford to execute all of them and pick the best solution.

At time t, the robot encounters a planning problem Γ_t. It executes all planners in the library. Let the cost of the solution on executing a planner $\mathcal{P}_i$ on Γ_t be $J(\Gamma_t, \mathcal{P}_i)$. It selects the best plan computed by the library which has a cost $\min_{\mathcal{P}_i \in \mathcal{L}} J(\Gamma_t, \mathcal{P}_i)$.

If the best solution cost is above a threshold $J_{\max}$, then the solution is deemed unsafe. This leads to an *exception event*. The robot comes to a stop. It then queries an oracle $\mathcal{O}$ with the failure problem Γ_t. The oracle returns an *exception planner* $\mathcal{P}_t = \mathcal{O}(\Gamma_t)$ that can solve the problem, i.e. $J(\Gamma_t, \mathcal{P}_t) \leq J_{\max}$. The robot has to choose how to update the library with this planner, while keeping the library size fixed at k. This event incurs a cost of 1. We wish to minimize the cumulative cost due to exception.

We now show how this problem is equivalent to the online paging problem. A planner $\mathcal{P}_i$ corresponds to a page p_i. The library $\mathcal{L}$ corresponds to the cache, both of whose size is limited to k. A planning problem Γ_t is said to be solved by a planner $\mathcal{P}_i$ if $J(\Gamma_t, \mathcal{P}_i) \leq J_{\max}$. We can interpret a Γ_t as a "request" for any planner belonging to this set. Hence planning problem Γ_t is equivalent to a page request.

The oracle $\mathcal{O}$ returns a planner $\mathcal{P}_t = \mathcal{O}(\Gamma_t)$ such that $J(\Gamma_t, \mathcal{P}_t) \leq J_{\max}$. In other words, it returns a planner that belongs to the set of planners that can satisfy the request Γ_t. To hypothetically construct the slow memory of N pages, we assume the oracle is omniscient and pre-computes exception planners for all problems that arrive. Hence N is the size of the sequence. Now the oracle's job is simply to retrieve the planner corresponding to the requested problem. We will see that the oracle does not actually have to compute *all these* exception planners - the online paging algorithms will only invoke the oracle during a cache miss while still being competitive w.r.t to this omniscient set.

This brings us to the question: What is the offline optimal algorithm OPT? Before we answer that, we point out that the paging problem assumes that the requests are for unique pages. However, in our problem, multiple planners can solve a planning problem. We can overcome this problem by making things harder for the offline optimal algorithm. Given a planning problem, the offline algorithm has to select a planner (from the set of N planners computed by the omniscient oracle) that will result in the solution with minimum cost. On the other hand, the online algorithms are free to satisfy the request with any planner that results in a solution cost less than $J_{\max}$. Hence the competitive ratio of a online paging algorithm carries over to this framework - the flexibility to choose any planner that satisfies the threshold $J_{\max}$ implies the

Algorithm 8: LRU $(\Omega, \mathcal{O})$

```
Initialize L with k planners (can be random);
Initialize usage time t_i = 0 for all P_i ∈ L;
for t = 1, ..., |Γ| do
    Receive planning problem Γ_i from sequence Ω;
    Select best planner from library P_s = arg min_{P_i ∈ L} J(Γ_t, P_i);
    if J(Γ_t, P_s) ≤ J_max then
        Set usage time of selected planner t_s = t
    else
        Invoke oracle to get an exception planner P* = O(Γ_t) ;
        Find least recently used planner P_LRU = arg min_{P_i ∈ L} t_i ;
        L ← L \ P_LRU ;
        L ← L ∪ P* ;
return L;
```

online algorithm can only do better.

We also note that we do not model the effect on choosing planner $\mathcal{P}_t$ on the sequence of planning problems the robot will encounter from $t+1$ onwards. Since the paging framework makes no assumptions on the sequence, it is applicable to our setting.

6.4 Online paging: The Least Recently Used algorithm

A popularly used algorithm in paging / caching is the *Least Recently Used (*LRU*)* algorithm. On a cache miss, the algorithm evicts the page whose most recent request is as far back in the past as possible. The motivation for this algorithm arises from the *Furthest-in-the-Future (FIF)* algorithm. FIF is an offline algorithm which always evicts the page that will be requested furthest in the future. Belady [1966] showed this to be the optimal algorithm. LRU attempts to simulate this using the past as a predictor for the future.

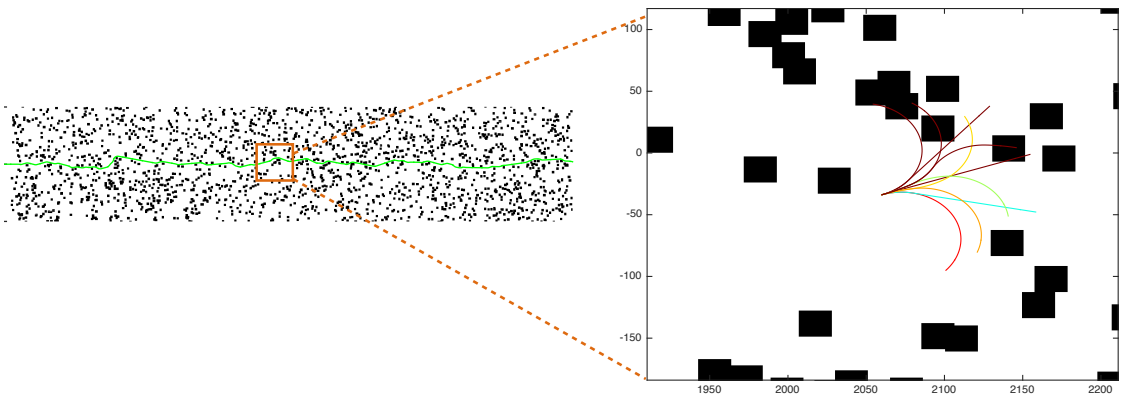
Algorithm 8 describes the LRU algorithm in the context of online exception planners. We begin with a library chosen at random (one can also start with an empty library and invoke the oracle the first k timesteps). We will deal with a planning problem sequence Ω. For each planner in the library $\mathcal{P}_i$, we maintain a usage time t_i. At time t, we receive problem Γ_t from Ω. All planners in the library are evaluated and the best planner $\mathcal{P}_s$ is selected. If the solution cost of this planner is less than $J_{\max}$, this planner is selected and the usage time is updated $t_s = t$. If not, an exception occurs (a cache miss) and the oracle is invoked to get a planner $\mathcal{P}^*$. We evict the least recently used planner from the library, i.e. with smallest t_i.

LRU has some desirable performance guarantees.

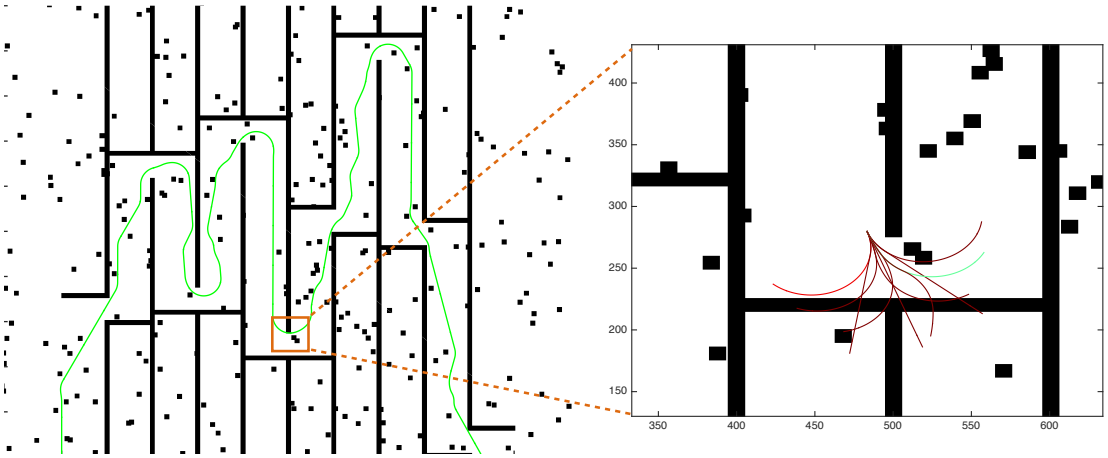
Theorem 6.1 (Upper Bound for LRU)**.** The competitive ratio of the LRU algorithm at most k, the size of the library.

Proof. Refer to Sleator and Tarjan [1985] □

Interestingly, no deterministic algorithm can have a better competitive ratio



(a)



(b)

Figure 6.2: Receding horizon planning on two kinds of environments. A robot equipped with a library of paths navigates from a global start to a global goal point. The robot only uses a small path of information to evaluate paths (as shown in the right). The paths are coloured from brown (high cost) to blue (low cost). By repeatedly following a fraction of the least cost path at each time step, the robot is eventually able to reach the goal (shown by the green trace on the left).

Theorem 6.2 (Lower Bound for Deterministic Algorithms)**.** Every deterministic paging algorithm has competitive ratio at least k.

Proof. Refer to Sleator and Tarjan [1985] □

6.5 Experiments

We empirically evaluate our approach on the simple setting of receding horizon planning with path sets [Green and Kelly , 2007]. This is a well examined paradigm and there exists good baseline approaches for us to compare against.

Fig. 6.2 shows the setup. We have a robot navigating in an environment with a library of

10 paths. The cost of a path is a linear combination of the length, proximity to obstacles and distance of terminal point from a goal point. At each time step, the robot evaluates each path and picks the best one. It then moves a fraction of the path forward, senses new obstacles and re-evaluates the path set. Hence the term receding horizon planning. In this setting, the oracle is equivalent to a brute force search algorithm that goes through a larger set of 100 paths and returns the best path. The assumption is that the robot has only enough onboard computation budget to evaluate 10 paths. We also assume there always exists a path in the larger set that solves the problem.

We evaluate our approach on two datasets - a sparse forest Fig. 6.2(a) and a maze like environment Fig. 6.2(b). We evaluate on 100 instances of such problems. The evaluation is done back to back, i.e. the library from one problem is carried over to the next. We evaluate different paging algorithms in terms of the number of times the oracle had to be invoked.

We compare LRU against a set of baseline online paging algorithms.

1. *First-in, First-out (FIFO)*: Replace the page that has been in memory the longest.
2. *Last-in, First-out (LIFO)*: Replace the page most recently moved to cache.
3. *Least Frequently Used (LFU)*: Replace the page that has been accessed the least.

We additionally compare against a baseline method for generating path sets that are diverse. The Green Kelly (GK) pathset selects paths that minimize dispersion using the Haussdorf metric [Green and Kelly , 2007]. This method is using some domain knowledge of what makes a good path - something that our algorithms do not have access to.

Fig. 6.3 shows the performance of all algorithms on both datasets. The graphs show the cumulative cache misses with time. Fig. 6.3(a) shows that LRU outperforms all baselines. Fig. 6.3(b) shows LRU outperforms all but one baseline - the GK pathset. Interestingly, the GK pathset performs poorly on the first problem.

To gain some more insight, we plotted the pathsets of LRU and GK for both datasets. Fig. 6.3(c) shows the GK pathset (which is dataset independent). This set is diverse with paths spreading out as much as possible. Fig. 6.3(d) shows the LRU pathset on the forest example. There is a stark difference between the two. The LRU pathset are much more forward facing than the GK pathset. This is because the pathset is data-driven and most probably the forest examples do not require hard turns as the ones in the GK pathset.

On the other hand, the LRU pathset for the maze environment shown in Fig. 6.3(e) looks much more similar to the GK pathset. This is because the maze environment requires sharp turns from one corridor to another, hence requiring such paths. In summary, LRU is able to adapt to different environments which non-adaptive approaches like GK pathset are unable to do.

6.6 Discussion and future directions

We addressed the problem of dynamically maintaining a library of exception planners - planners created on-the-fly to deal with failure scenarios - such that the number of exception events are

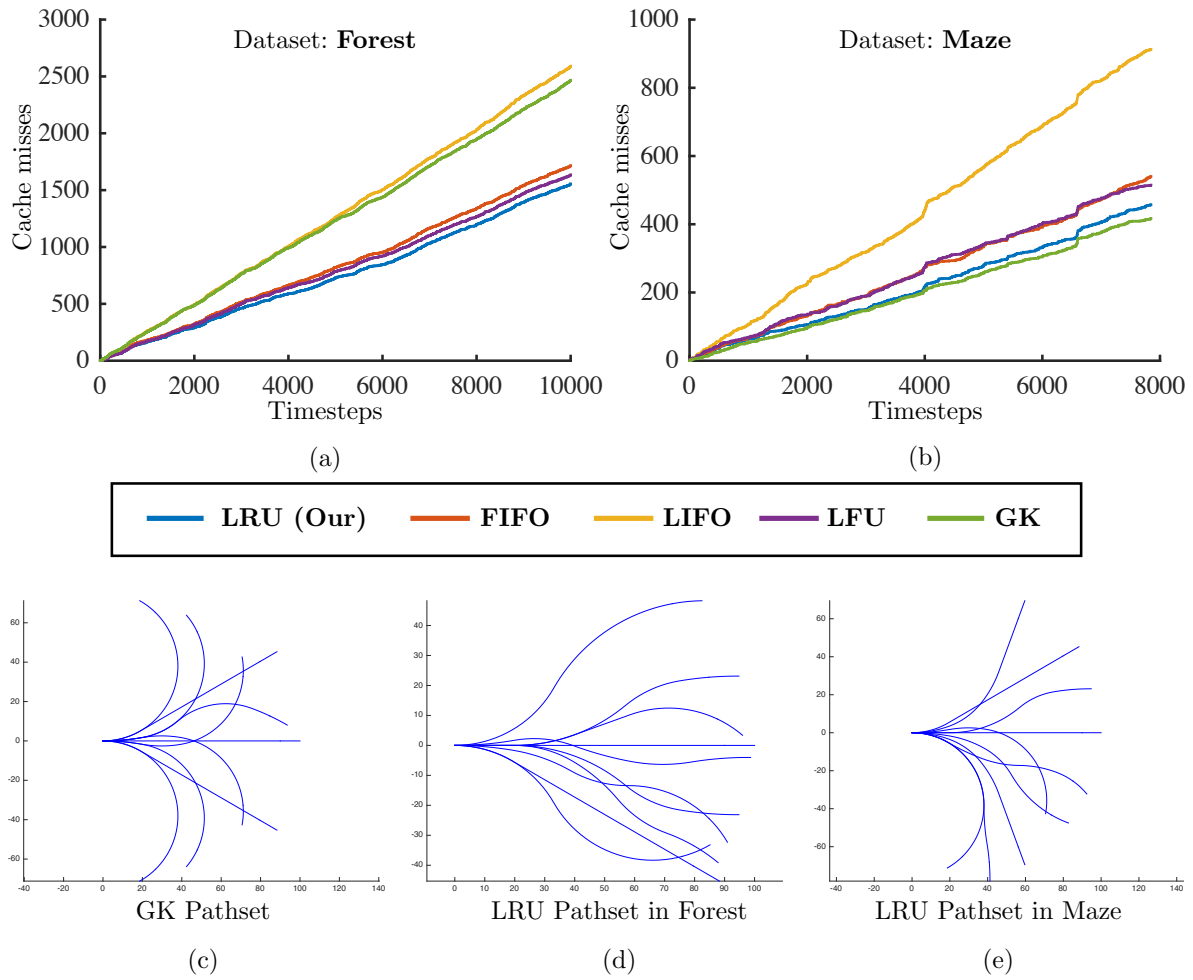


Figure 6.3: Comparison of different algorithms on two different datasets (a) On the Forest dataset, our method performs the best while the non-adaptive baseline (GK) has the second worst performance. (b) On the Maze dataset, our method perfroms second best while GK performs the best (c) The GK pathset mostly has paths that make sharp turns at different curvatures. This explains why it has poor performance in the Forest dataset - it is unable to navigate through the fine gaps between obstacles. (d) The adaptive LRU pathset (at the end of the trials) has different paths that all point forward. This pathset has adapted to the forest environment to find gaps between obstacles (e) The adaptive LRU pathset in the Maze environment looks much different. This pathset has adapted to find the turns required in the maze. It somewhat resembles the GK pathset which explains why the GK pathset succeeds in the Maze dataset.

minimized. We showed that this problem is equivalent to the online paging problem where a cache of pages must be maintained to minimize cache misses. We adopted an online algorithm, Least Recently Used, to solve this problem with good performance guarantees. We validated our approach on the simple setting of receding horizon planning with path sets in different environments. Our approach outperformed state-of-the-art baseline approaches for path set generation.

We now discuss some insights and directions for future work.

Q 1 (Paging with Request Sets)**.** Instead of a unique page request, can we extend the framework

to a set of requests?

The equivalence to online paging required a caveat - we had to assume that we are competing with an offline algorithm that assigns a unique planner for a planning problem. However, there are many planners which can solve a given planning problem. An interesting future direction would be to relax this assumption. A promising way forward would be to formulate the problem of requesting a set of pages as done in Epstein et al. [2009]. Here a set of requests are made and if any one is satisfied, a cache miss is avoided.

Q 2 (Models of Data in Planning)**.** What sort of data model explains the empirical success of LRU?

A long standing explanation for the success of LRU in paging has been that real data exhibits locality - recent requests are likely to be requested soon. However, it is possible to make more concrete claims. For example, Borodin et al. [1995] proposed access graphs as a natural way to capture locality of reference in page request sequences. An access graph is a graph where every node is a page. Consecutive requests can only be made if they are neighbours in the graph. On such models, one can show that LRU has better competitive ratio than FIFO. It would be an interesting future direction to explore the nature of data models in planning and to see if that explains the empirical success of certain path set creating algorithms.

Part III

White-box Adaptive Planners

7

Data-driven Planning via Imitation Learning

In previous Chapters, we worked with black-box planners which followed a two step strategy of extracting context from the problem and selecting which expert planner to invoke. We argued in Chapter 5 that this was computationally inefficient as the effort spent in extracting more and more meaningful context was not used by the planners themselves. This motivates us to address Challenge 3 which requires the design of a white-box planner that adapts its search strategies as it gain information about the planning problem.

We motivate the utility of a white-box adaptive planner in Section 7.1 in the context of two different domains - informative path planning and heuristic search based planning. We provide background on both domains in Section 7.2 and define the nature of the policy that we wish to train. We then map both problems to a common POMDP framework in Section 7.3. We draw a novel connection to imitation learning of clairvoyant oracles to solve such problems in Section 7.4. We show how one can train efficient policies using this connection in Section 7.5. We extensively evaluate the approach in both domains in Section 7.6 and Section 7.7. We summarize and discuss potential future directions in Section 7.8.

7.1 Introduction

Robot planning is the process of selecting a sequence of actions that optimize for a task specific objective. For instance, the objective for a navigation task would be to find collision free paths, while the objective for an exploration task would be to map unknown areas. The optimal solutions to such tasks are heavily influenced by the implicit structure in the environment, i.e. the configuration of objects in the world. State-of-the-art planning approaches, however, do not exploit this structure, thereby expending valuable effort searching the action space instead of focusing on potentially good actions. This motivates us to examine the following question:

> How can we enable planners to adapt their search strategies by inferring good actions in an efficient manner using only the information uncovered by the search up until

that time?

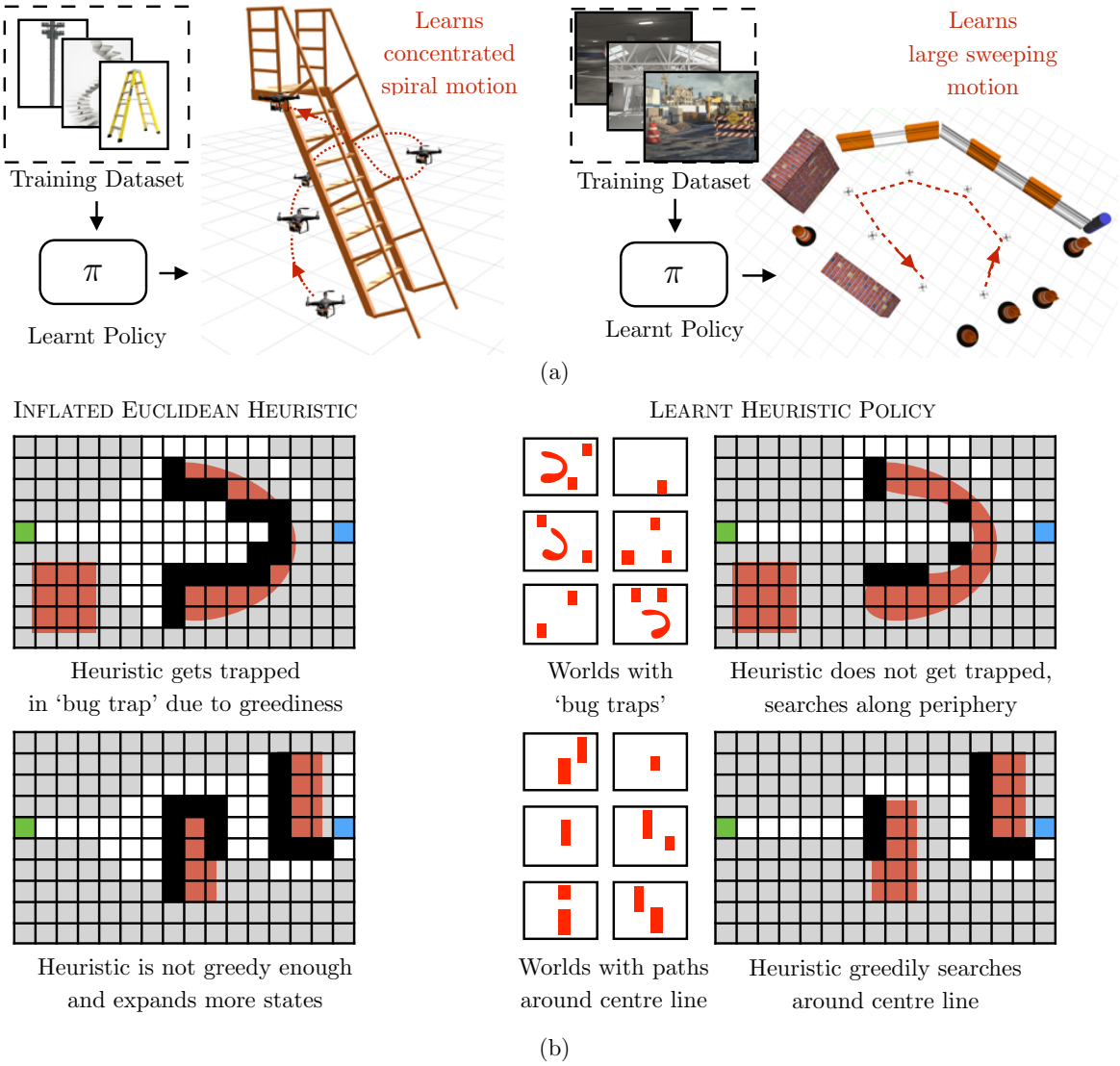


Figure 7.1: Sequential decision making in informative path planning and search based planning. The implicit structure of the environment affects the performance of policies in both tasks. (a) The effectiveness of a policy to gather information depends on the distribution of worlds. (left) When the distribution corresponds to a scene containing ladders, the learnt policy executes a helical motion around parts of the ladder already observed as it is unlikely that there is information elsewhere. (right) When the distribution corresponds to a scene from a construction site, the learnt policy executes a large sweeping motion as information is likely to be dispersed. (b) A learnt heuristic policy adapts to different obstacle configurations to minimize search effort. All schematics show the evolution of a search algorithm as the expansion of a search wavefront (expanded(white), invalid(black), unexpanded(grey)) from start (green) to goal (blue). A commonly used inflated Euclidean heuristic cannot adapt to different environments, e.g it gets stuck in bugtraps. On the other hand, the learnt policy is able to infer the presence of a bug trap when trained on such a distribution and switch to greedy behaviour when trained on other distributions.

7.1.1 Motivation

We look at two domains - informative path planning and search based planning. We briefly delve into these motivations and make the case for data-driven approaches in both.

Informative path planning

We consider the following information gathering problem - given a hidden world map, sampled from a prior distribution, the goal is to successively visit sensing locations such that the amount of relevant information uncovered is maximized while not exceeding a specified fuel budget. Consider a robot equipped with a sensor (RGBD camera) that needs to generate a map of an unknown environment. It is given a prior distribution about the geometry of the world, but has no other information. This geometry could include very diverse settings. First it can include a world where there is only one ladder, but the form of the ladder must be explored, which is a very dense setting. Second, it could include a sparse setting with spatially distributed objects, such as a construction site.

The important task for the robot is to now try to infer which type of environment it is in based on the history of measurements, and thus plan an efficient trajectory. At every time step, the robot visits a sensing location and receives a sensor measurement (e.g. depth image) that has some amount of information utility (e.g. surface coverage of objects with point cloud). As opposed to naive lawnmower-coverage patterns, it will be more efficient if the robot could use a policy that maps the history of locations visited and measurements received to decide which location to visit next such that it maximizes the amount of information gathered in the finite amount of battery time available.

The ability of such a learnt policy to gather information efficiently depends on the prior distribution of worlds in which the robot has been shown how to navigate optimally. Fig. 7.1(a) (left) shows an efficient learnt policy for inspecting a ladder, which executes a helical motion around parts of the ladder already observed to efficiently uncover new parts without searching naively. This is efficient because given the prior distribution the robot learns that information is likely to be geometrically concentrated in a particular volume given its initial observations of parts of the ladder. Similarly, Fig. 7.1(a) (right) shows an effective policy for exploring construction sites by executing large sweeping motions. Here again the robot learns from prior experience that wide, sweeping motions are efficient since it has learnt that information is likely to be dispersed in such scenarios. We wish to arrive at an efficient procedure for training such a policy.

Search based planning

Search based motion planning offers a comprehensive framework for reasoning about a vast number of motion planning algorithms [LaValle, 2006]. In this framework, an algorithm grows a *search tree* of feasible robot motions from a start configuration towards a goal [Pearl, 1984]. This is done in an incremental fashion by first selecting a leaf node of the tree, *expanding* this node by computing outgoing edges, checking each edge for validity and finally updating the tree with potentially new leaf nodes. It is useful to visualize this search process as a *wavefront of expanded*

nodes that grows from the start outwards till it finds the goal as illustrated in Fig. 7.1(b). In order to ensure real-time performance, heuristics are employed to guide the search.

While state-of-the-art methods propose different relaxation-based [Dolgov et al., 2008, Likhachev and Ferguson, 2009] and learning-based approaches [Paden et al., 2017] to computing heuristics as estimates of distance to goal, they run into a much more fundamental limitation - *a small estimation error can lead to a large search wavefront*. Minimizing the estimation error does not necessarily minimize search effort.

Instead, we focus on the latter objective of designing heuristics that explicitly reduce search effort in the interest of real-time performance. Our key insight is that *heuristics should adapt during search* - as the search progresses, they should actively infer the structure of the valid configuration space, and focus the search on potentially good areas. Moreover, we want to learn this behaviour from data - changing the data distribution should change the heuristic automatically. Consider the example shown in Fig. 7.1(b). When a heuristic is trained on a world with 'bug traps', it learns to recognize when the search is trapped and circumvent it. On the other hand, when it is trained on a world with narrow gaps, it learns a greedy behaviour that drives the search to the goal.

7.1.2 Key idea

It is natural to think of both these problems as a Partially Observable Markov Decision Process (POMDP). However the POMDP is defined on a belief over possible world maps, which is very large in size rendering even the most efficient of online POMDP solvers impractical.

Our key insight is that if the policies could fully observe and process the world map during decision making, they could quite easily disambiguate good actions from bad ones. This motivates us to frame the problem of learning a planning policy as a novel data-driven imitation [Ross and Bagnell, 2014] of a *clairvoyant oracle*. During the training process, the oracle has full knowledge about the world map (hence clairvoyant) and selects actions that maximize cumulative rewards. The policy is then trained to imitate these actions as best as it can using partial knowledge from the current history of actions and observations. As a result of our novel formulation, we are able to sidestep a number of challenging issues in POMDPs like explicitly computing posterior distribution over worlds and planning in belief space.

We empirically show that training such policies using imitation learning of clairvoyant oracles leads to much faster convergence and robustness to poor local minima than training policies via model free policy improvement. We leverage the fact that such oracles can be efficiently computed for our domains once the source of uncertainty is removed. We show in our analysis that imitation of such clairvoyant oracles during training is equivalent to being competitive with a *hallucinating oracle* at test time, i.e. an oracle that implicitly maintains a posterior over world maps and selects the best action at every time step. This offers some valuable insight behind the success of this approach as well as instances where such an approach would lead to a near-optimal policy.

7.1.3 Contributions

Our contributions are as follows:

1. We motivate the need to learn a planning policy that adapts to the environment in which the robot operates. We examine two domains - informative path planning and search based planning. We examine both problems through the lens of sequential decision making under uncertainty (Section 7.2).
2. We present a novel mapping of both these problems to a common POMDP framework (Section 7.3).
3. We propose a novel framework for training such POMDP policies via imitation learning of a clairvoyant oracle. We analyze the implications of imitating such an oracle (Section 7.4).
4. We present training procedures that deal with the non i.i.d distribution of states induced by the policy itself along with performance guarantees. We present concrete instances of the algorithm for both problem domains. We also show that for a certain class of informative path planning problems, policies trained in this fashion possess near-optimality properties (Section 7.5).
5. We extensively evaluate the approach on both problem domains. In each domain, we evaluate on a spectrum of environments and show that policies outperform state-of-the-art approaches by exhibiting adaptive behaviours. We also demonstrate the impact of this framework on real world problems by presenting flight test results from a UAV (Section 7.6 and Section 7.7).

7.2 Background

7.2.1 Informative path planning

We now present a framework for informative path planning where the objective is to visit maximally informative sensing locations subjected to time and travel constraints. We use this framework to pose the problem of computing a information gathering policy for a given distribution over worlds and briefly discuss prior work on this topic.

Framework

We now introduce a framework and set of notations to express the IPP problems of interest. The specific implementation details of the problem are described in detail in [Choudhury et al., 2017a].

We have a robot that is constrained to move on a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of nodes corresponding to all sensing locations. The start node is v_s. Let $\xi = (v_1, v_2, \ldots, v_p)$ be a sequence of connected nodes (a path) such that $v_1 = v_s$. Let Ξ be the set of all such paths.

Let $\phi \in \mathcal{M}$ be the world map in which the robot operates. The world map is usually represented in practice as a binary grid map where grid cells are either occupied or free. We assume that the world map is fixed during an episode.

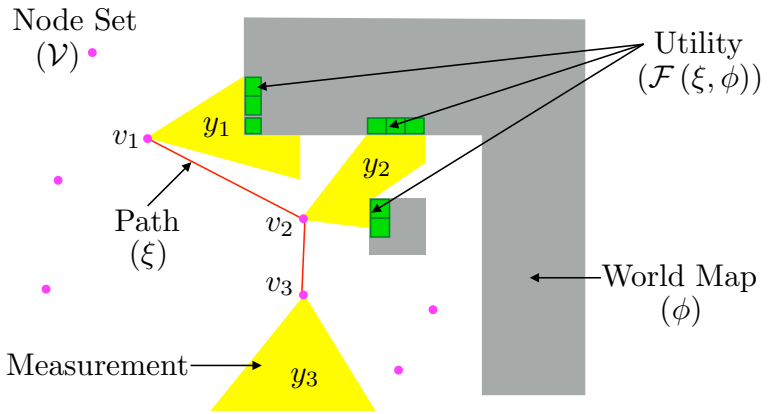


Figure 7.2: The informative path planning problem. Given a world map ϕ, the robot plans a path ξ which visits a node $v_i \in \mathcal{V}$ and receives measurement y_i, such that utility (information gathered) $\mathcal{F}(\xi, \phi)$ is maximized. Here the utility is the cardinality of all the cells uncovered (green), which is a union of the cells uncovered at each location (and hence a set cover function)

Let $y \in \mathcal{Y}$ be a measurement received by the robot. Let $\mathcal{H} : \mathcal{V} \times \mathcal{M} \to \mathcal{Y}$ be a measurement function. When the robot is at node v in a world map ϕ, the measurement y received by the robot is $y = \mathcal{H}(v, \phi)$. The measurement function is defined by a sensor model, e.g. a range limited sensor. A measurement is obtained by projecting the sensor model on the sensing node v and ray-casting to determine the surfaces of the underlying world ϕ that intersect with the sensor rays.

The objective of the robot is to move on the graph and maximize utility. Let $\mathcal{F} : 2^{\mathcal{V}} \times \mathcal{M} \to \mathbb{R}_{\geq 0}$ be a utility function. For a path ξ and a world map ϕ, $\mathcal{F}(\xi, \phi)$ assigns a utility to executing the path on the world. The utility of a measurement from a node is usually the amount of surface of the world covered by it. In such an instance, the function does not depend on the sequence of vertices in the path, i.e. is a set function. For simplicity, we assume that the measurement and utility function is deterministic.

As the robot moves on the graph, the travel cost is captured by the cost function $\mathcal{T} : \Xi \times \mathcal{M} \to \mathbb{R}_{\geq 0}$. For a path ξ and a world map ϕ, $J\xi\phi$ assigns a travel cost for executing the path on the world. In a practical setting, the total number of timesteps is bounded by T and the travel cost is bounded by B. Fig. 7.2 shows an illustration of the framework.

We are now ready to define the informative path planning problems. There are two axes of variations

1. Constraint on the motion of the robot
2. Observability of the world map

The first axis arises from whether the robot is subject to any travel constraints. For problems such as sensor placement, the agent is free to select any sequence of nodes and the travel cost between nodes is 0. For such situations, the graph is also fully connected to permit any sequence. For problems involving physical movements, the agent is constrained by a budget on the travel cost. Additionally, the graph may also not be fully connected.

The second axis arises from different task specifications which result in the world map being observable or being hidden. We categorize the problems on this axis to aid future discussions on imitating clairvoyant oracles in Section 7.5.

Problems with known world maps

For the first two variants, the world map ϕ is known and can be evaluated while computing a path ξ.

Problem 5 (KNOWN-UNC: Known World Map; Unconstrained Travel Cost)**.** Given a world map ϕ, a fully connected graph $\mathcal{G}$ and a time horizon T, find a path ξ that maximizes utility

$$\begin{aligned} \arg\max_{\xi \in \Xi} \quad & \mathcal{F}(\xi, \phi) \\ \text{s.t.} \quad & |\xi| \leq T + 1 \end{aligned} \tag{7.1}$$

In the case where the utility function is a set function, Problem 5 is a set function maximization problem which in general can be NP-Hard [Krause and Golovin, 2012]). Such problems occur commonly in the sensor placement problem [Krause et al., 2008]. However, in many instances the utility function can be shown to posses the powerful property of *monotone submodularity*. This property implies the following

1. *Monotonic improvement*: The value of the utility can only increase on adding nodes, i.e.

   $$\mathcal{F}(\mathcal{V}_1 \cup \mathcal{V}_2, \phi) \geq \mathcal{F}(\mathcal{V}_1, \phi)$$

   for all $\mathcal{V}_1, \mathcal{V}_2 \subseteq \mathcal{V}$

2. *Diminishing returns*: The gain in adding a set of nodes diminshes

   $$\begin{aligned} \mathcal{F}(\mathcal{V}_1 \cup \mathcal{V}_3, \phi) - \mathcal{F}(\mathcal{V}_3, \phi) \leq & \mathcal{F}(\mathcal{V}_1 \cup \mathcal{V}_2, \phi) \\ & - \mathcal{F}(\mathcal{V}_2, \phi) \end{aligned}$$

   for all $\mathcal{V}_1, \mathcal{V}_2, \mathcal{V}_3 \subseteq \mathcal{V}$ where $\mathcal{V}_2 \subseteq \mathcal{V}_3$.

For such functions, it has been shown that a greedy algorithm achieves near-optimality [Krause and Guestrin, 2007, Krause et al., 2008].

Problem 6 (KNOWN-CON: Known World Map; Constrained Travel Cost)**.** Given a world map ϕ, a time horizon T and a travel cost budget B, find a path ξ that maximizes utility

$$\begin{aligned} \arg\max_{\xi \in \Xi} \quad & \mathcal{F}(\xi, \phi) \\ \text{s.t.} \quad & \mathcal{T}(\xi, \phi) \leq B \\ & |\xi| \leq T + 1 \end{aligned} \tag{7.2}$$

Problem 6 introduces a routing constraint (due to $\mathcal{T}$) for which greedy approaches can perform arbitrarily poorly. Such problems occur when a physical system has to travel between nodes. Chekuri and Pal [2005], Singh et al. [2007] propose a quasi-polynomial time recursive greedy

approach to solving this problem. Iyer and Bilmes [2013] solve a related problem (submodular knapsack constraints) using an iterative greedy approach which is generalized by Zhang and Vorobeychik [2016]. Yu et al. [2014] propose a mixed integer approach to solve a related correlated orienteering problem. Hollinger and Sukhatme [2013] propose a sampling based approach. Arora and Scherer [2017] use an efficient TSP with a random sampling approach.

Problems with hidden world maps

We now consider the setting where the world map ϕ is hidden. Given a prior distribution $P(\phi)$, it can be inferred only via the measurements y_i received as the robot visits nodes v_i. Hence, instead of solving for a fixed path, we compute a policy that maps history of measurements received and nodes visited to decide which node to visit.

Problem 7 (HIDDEN-UNC: Hidden World Map; Unconstrained Travel Cost)**.** Given a distribution of world maps, $P(\phi)$, a fully connected graph $\mathcal{G}$, a time horizon T, find a policy that at time t, maps the history of nodes visited $\{v_i\}_{i=1}^t$ and measurements received $\{y_i\}_{i=1}^t$ to compute the next node v_{t+1} to visit at time $t+1$, such that the expected utility is maximized.

Such a problem occurs for sensor placement where sensors can optionally fail [Golovin and Krause, 2011]. Due to the hidden world map ϕ, it is not straight forward to apply the approaches of Problem KNOWN-UNC- we have to reason both about $P(\phi \mid \{v_i\}_{i=1}^t, \{y_i\}_{i=1}^t)$ and how the function will evolve. However, in some instances the utility function $\mathcal{F}$ has an additional property of *adaptive submodularity* [Golovin and Krause, 2011]. This is an extension of the submodularity property where the gain of the function is measured in expectation over the conditional distribution over world maps $P(\phi \mid \{v_i\}_{i=1}^t, \{y_i\}_{i=1}^t)$. Under such situations, applying greedy strategies to Problem 7 has near-optimality guarantees [Chen et al., 2015b, 2016b, Golovin et al., 2010, Javdani et al., 2013, 2014b]). However, these strategies require explicitly sampling from the posterior distribution over ϕ which make it intractable to apply for our setting.

Problem 8 (HIDDEN-CON: Hidden World Map; Constrained Travel Cost)**.** Given a distribution of world maps, $P(\phi)$, a time horizon T, and a travel cost budget B, find a policy that at time t, maps the history of nodes visited $\{v_i\}_{i=1}^t$ and measurements received $\{y_i\}_{i=1}^t$ to compute the next node v_{t+1} to visit at time $t+1$, such that the expected utility is maximized.

Such problems crop up in a wide number of areas such as sensor planning for 3D surface reconstruction [Isler et al., 2016] and indoor mapping with UAVs [Charrow et al., 2015, Nelson and Michael, 2015]. Problem 8 does not enjoy the adaptive submodularity property due to the introduction of travel constraints. Hollinger et al. [2011, 2012] propose a heuristic based approach to select a subset of informative nodes and perform minimum cost tours. Singh et al. [2009] replan every step using a non-adaptive information path planning algorithm. Inspired by adaptive TSP approaches by Gupta et al. [2010], Lim et al. [2015, 2016] propose recursive coverage algorithms to learn policy trees. However, such methods cannot scale well to large state and observation spaces. Heng et al. [2015] make a modular approximation of the objective function. Isler et al. [2016] survey a broad number of myopic information gain based heuristics that work well in practice but have no formal guarantees.

7.2.2 Search based planning

We now present a framework for search based planning where the objective is to find a feasible path from start to goal while minimizing search effort. We use this framework to pose the problem of learning the optimal heuristic for a given distribution over worlds and briefly discuss prior work on this topic.

Framework

We consider the problem of search on a graph, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where vertices $\mathcal{V}$ represent robot configurations and edges $\mathcal{E}$ represent potentially valid movements of the robot between these configurations. Given a pair of start and goal vertices, $(v_s, v_g) \in \mathcal{V}$, the objective is to compute a path $\xi \subseteq \mathcal{E}$ - a connected sequence of valid edges. The implicit graph $\mathcal{G}$ can be compactly represented by (v_s, v_g) and a successor function $\texttt{Succ}(v)$ which returns a list of outgoing edges and child vertices for a vertex $v \in \mathcal{V}$. Hence a graph $\mathcal{G}$ is constructed during search by repeatedly *expanding* vertices using $\texttt{Succ}(v)$. Let $\phi \in \mathcal{M}$ be a representation of the world that is used to ascertain the validity of an edge. An edge $e \in \mathcal{E}$ is checked for validity by invoking an evaluation function $\texttt{Eval}(e, \phi)$ which is an expensive operation and may require complex geometric intersection operations [Dellin and Srinivasa, 2016].

Algorithm 9 defines a general search based planning algorithm `Search` which takes as input the tuple $\langle v_s, v_g, \texttt{Succ}, \texttt{Eval}, \phi, \texttt{Select} \rangle$ and returns a valid path ξ. To ensure systematic search, the algorithm maintains the following lists - an open list $\mathcal{O} \subset \mathcal{V}$ of candidate vertices to be expanded and a closed list $\mathcal{C} \subset \mathcal{V}$ of vertices which have already been expanded. It also retains an additional invalid list $\mathcal{I} \subset \mathcal{E}$ of edges found to be in collision. These 3 lists together represent the complete information available to the algorithm at any given point of time. At a given iteration, the algorithm uses this information to select a vertex $v \in \mathcal{O}$ to expand by invoking $\texttt{Select}(\mathcal{O})$. It then expands v by invoking $\texttt{Succ}(v)$ and checking validity of edges using $\texttt{Eval}(e, \phi)$ to get a set of valid successor vertices $\mathcal{V}_{\text{succ}}$ as well as invalid edges $\mathcal{E}_{\text{inv}}$. The lists are then updated and the process repeated till the goal vertex v_g is uncovered. Fig. 7.3 illustrates this framework.

The optimal heuristic problem

In this work, we focus on the *feasible path problem* and ignore the optimality of the path. Although this is a restrictive setting, quickly finding the feasible path is a very important problem in robotics. Efficient feasible path planners such as RRT-Connect [Kuffner and LaValle, 2000] has proven highly effective in high dimensional motion planning applications such as robotic arm planning [LaValle, 2006] and mobile robot planning [Laumond et al., 1998]. Hence we ignore the traversal cost of an edge and deal with unweighted graphs. We defer discussions on how to relax this restriction to Section 7.8.

We view a heuristic policy as a *selection function* (Algorithm 9, Line 3) that selects a vertex v from the open list $\mathcal{O}$. The objective of the policy is to minimize the number of expansions until the search terminates. Note that the evolution of the open list $\mathcal{O}$ depends on the underlying world map ϕ which is hidden. Given a prior distribution over world maps $P(\phi)$, it can be inferred only via the outcome of the expansion operation $(\mathcal{V}_{\text{succ}}, \mathcal{E}_{\text{inv}})$. The history of outcomes is captured by

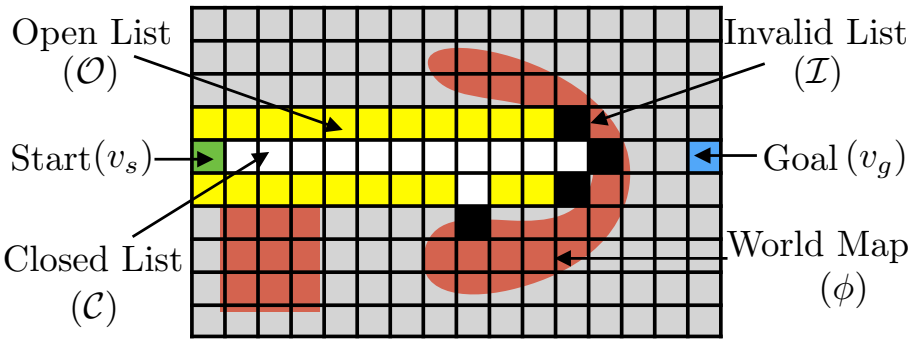


Figure 7.3: The search based planning problem. Given a world map ϕ, the agent has to guide a search tree from start v_s to goal v_g by expanding vertices. At any given iteration, the open list $\mathcal{O}$ represents the set of candidate vertices that can be expanded. The closed list $\mathcal{C}$ represents the set of vertices already expanded. The invalid list represents the set of edges that were found to be in collision with the world. The status of every other vertex is unknown. The search continues till the goal belongs to the open list, i.e. a feasible path to goal has been found.

Algorithm 9: $\texttt{Search}\langle v_s, v_g, \texttt{Succ}, \texttt{Eval}, \phi, \texttt{Select}\rangle$

$\mathcal{O} \leftarrow v_s$, $\mathcal{C} \leftarrow \emptyset$, $\mathcal{I} \leftarrow \emptyset$;
while $v_g \notin \mathcal{O}$ **do**
  $v \leftarrow \texttt{Select}(\mathcal{O})$;
  $(\mathcal{V}_{\text{succ}}, \mathcal{E}_{\text{inv}}) \leftarrow \texttt{Expand}(v, \texttt{Succ}, \texttt{Eval}, \phi)$;
  $\mathcal{O} \leftarrow \mathcal{O} \cup \mathcal{V}_{\text{succ}}$, $\mathcal{C} \leftarrow \mathcal{C} \cup v$, $\mathcal{I} \leftarrow \mathcal{I} \cup \mathcal{E}_{\text{inv}}$;
return $\texttt{Path}(v_s, v_g)$;

the state of the search, i.e. the combination of the 3 lists $\{\mathcal{O}, \mathcal{C}, \mathcal{I}\}$.

Problem 9 (Opt-Heur)**.** Given a distribution of world maps, $P(\phi)$, find a heuristic policy that at time t, maps the state of the search $\{\mathcal{O}_t, \mathcal{C}_t, \mathcal{I}_t\}$ to select a vertex $v_t \in \mathcal{O}_t$ to expand, such that the expected number of expansions till termination is minimized.

The problem of heuristic design has a lot of historical significance. A common theme is "Optimism Under Uncertainty". A spectrum of techniques exist to manually design good heuristics by relaxing the problem to obtain guarantees with respect to optimality and search effort [Pearl, 1984]. To get practical performance, these heuristics are inflated, as has been the case in the applications in mobile robot planning [Likhachev and Ferguson, 2009]. However, being optimistic under uncertainty is not a foolproof approach and could be disastrous in terms of search efforts depending on the environment (See Fig 2.5, LaValle [2006]).

Learning heuristics falls under machine learning for general purpose planning [Jiménez et al., 2012]. Yoon et al. [2006] propose using regression to learn residuals over FF-Heuristic [Hoffmann and Nebel, 2001]. Xu et al. [2007, 2009, 2010] improve upon this in a beam-search framework. Arfaee et al. [2011] iteratively improve heuristics. ús Virseda et al. [2013] learn combination of heuristic to estimate cost-to-go. Kendall rank coefficient is used to learn open list ranking [Garrett et al., 2016, Wilt and Ruml, 2015]. Thayer et al. [2011] learn heuristics online during search. Paden et al. [2017] learn admissible heuristics as S.O.S problems. However, these methods do not address minimization of search effort and also ignore the non i.i.d nature of the problem.

7.2.3 Partially observable Markov decision process

POMDPs [Kaelbling et al., 1998] provide a rich framework for sequential decision making under uncertainty. However, solving a POMDP is often intractable - finite horizon POMDPs are PSPACE-complete [Papadimitriou and Tsitsiklis, 1987] and infinite horizon POMDPs are undecidable [Madani et al., 2003]. Despite this challenge, the field has forged on and produced a vast amount of work by investigating effective approximations and analyzing the structure of the optimal solution. We refer the reader to [Ross et al., 2008] for a concise survey of modern approaches.

There are two main approaches to POMDP planning: offline policy computation and online search. In offline planning, the agent computes before hand a policy by considering all possible scenarios and executes the policy based on the observation received. Athough offline methods have shown success in planning near-optimal policies in several domains [Kurniawati et al., 2008, Smith and Simmons, 2012, Spaan and Vlassis, 2005], they are difficult to scale up due to the exponential number of future scenarios that must be considered.

Online methods interleave planning and execution. The agent plans with the current belief, executes the action and updates the belief. Monte-carlo sampling methods explicitly maintain probability over states and plan via monte-carlo roll-outs [Asmuth and Littman, 2011, McAllester and Singh, 1999]. This limits scalability since belief update can take time. In contrast, POMCP [Silver and Veness, 2010] maintains a set of particles to represent belief and employ UCT methods to plan with these particles. This allows the method to scale up for larger state spaces.

However, the disadvantage of purely online methods is that they require a lot of search effort online and can lead to poor performance due to evaluation on a small number of particles. Somani et al. [2013] present a state-of-the-art algorithm DESPOT that combines the best aspects of many algorithms. First it uses determinized sampling techniques to ensure that the branching factor of the tree is bounded [Kearns et al., 2000, Ng and Jordan, 2000]. Secondly, it uses offline precomputed policies to roll-out from a vertex, thus lower bounding its value. Finally, it tries to regularize the search by weighing the utility of a node to be robust against the fact that a finite number of samples is being used.

The methods we have talked about explicitly models the belief. For large scale POMDPs, this might be an issue. Model free approaches and representation learning offer attractive alternatives. Model free policy improvement has been successfully used to solve POMDPs [Li et al., 2009, Liu et al., 2013]. Predictive state representations [Boots et al., 2011, Littman and Sutton, 2002] that minimize prediction loss of future observations offer more compact representations than maintaining belief. There also has been a lot of success in employing deep learning to learn powerful representations [Hausknecht and Stone, 2015, Karkus et al., 2017].

7.2.4 Reinforcement learning and imitation learning

Reinforcement Learning (RL) [Sutton and Barto, 1998] especially deep RL has dramatically advanced the capabilities of sequential decision making in high dimensional spaces such as controls [Duan et al., 2016], video games [Silver et al., 2016] and strategy games [Silver et al., 2016].

Several conventional supervised learning tasks are now being solved using deep RL to achieve higher performance [Li et al., 2016, Ranzato et al., 2015]. In sequential decision making, the prediction of a learner is dependent on the history of previous outcomes. Deep RL algorithms are able to train such predictors by reasoning about the future accumulated cost in a principle manner.

We refer the reader to Kober et al. [2013] for a concise survey on RL and to Arulkumaran et al. [2017] for a survey on deep RL. Training such policies can be classified into two approaches - either *value function-based approach*, where a value function for an action is learnt, or *policy search*, where a policy is directly learnt. The value function methods can themselves be categorized in two categories - *model-free* algorithms and *model-based* algorithms.

Model-free methods are computationally cheap but ignore the dynamics of the world thus requiring a lot of samples. Q-learning [Watkins and Dayan, 1992] is a representative algorithm for estimating the long-term expected return for executing an action from a given state. When the number of state action pairs are too large in number to track each uniquely, a function approximator is required to estimate the value. Deep Q-learning [Mnih et al., 2015, Wang et al., 2016] addresses such a need by employing a neural-network as a function approximator and learning these network weights. However, the process of using the same network to generate both target values and update Q-values results in oscillations. Hence a number of remedies are required to maintain stability such as having a buffer of experience, a separate target network and an adaptive learning rate. These are indicative of the underlying sample inefficiency problem of a model-free approach.

Model-based methods such as R-Max [Brafman and Tennenholtz, 2002] learn a model of the world which is then used to plan for actions. While such methods are sample efficient, they require a lot of exploration to learn the model. Even in the case when the model of the environment is known, solving for the optimal policy might be computationally expensive for large spaces. Policy search approaches are commonly used where its easier to parameterize a policy than learn a value function [Peters and Schaal, 2006], however such approaches are sensitive to initialization and can lead to poor local minima.

In contrast with RL methods, imitation learning (IL) algorithms [Chang et al., 2015, Daumé et al., 2009, Ross and Bagnell, 2014, Venkatraman et al., 2014] reduce the sequential prediction problem to supervised learning by leveraging the fact that, for many tasks, at training time we usually have a (near) optimal cost-to-go oracle. This oracle can either come from a human expert guiding the robot [Abbeel and Ng, 2004] or from ground truth data as in natural language processing [Chang et al., 2015]. The existence of such oracles can be exploited to alleviate learning by trial and error - imitation of an oracle can significantly speed up learning. A traditional approach to using such oracles is to learn a policy or value function from a pre-collected dataset of oracle demonstrations [Finn et al., 2016, Ratliff et al., 2009b, Ziebart et al., 2008]. A problem with these methods is that they require training and test data to be sampled from the same distribution which is difficult in practice. In contrast, interactive approaches to data collection and training has been shown to overcome stability issues and works well empirically [Ross and Bagnell, 2014, Ross et al., 2011, Sun et al., 2017]. Furthermore, these approaches lead to strong performance through a reduction to no-regret online learning.

Recent approaches have also employed imitation of clairvoyant oracles, that has access to more information than the learner during training, to improve reinforcement learning as they offer better sample efficiency and safety. Kahn et al. [2017], Zhang et al. [2016] train policies that map current observation to action by extending guided policy search [Levine and Koltun, 2013] for imitation of model predictive control oracles. Tamar et al. [2016] consider a cost-shaping approach for short horizon MPC by offline imitation of long horizon MPC which is closest to our work. Gupta et al. [2017] develop a holistic mapping and planner framework trained using feedback from optimal plans on a graph.

Sun et al. [2017] also theoretically analyze the question of why imitation learning aids in reinforcement learning. They develop a comprehensive theoretical study of IL on discrete MDPs and construct scenarios to show that IL achieves better sample efficiency than any RL algorithm. Concretely, they conclude that one can expect at least a polynomial gap ad a possible exponential gap in regret between IL and RL when one has access to unbiased estimates of the optimal policy during training.

7.3 Problem formulation

7.3.1 POMDPs

A discrete-time finite horizon POMDP is defined by the tuple $(X, \mathcal{A}, \Omega, R, \mathcal{O}, Z, T)$ where

- X is a set of states
- $\mathcal{A}$ is a set of actions
- Ω is a set of state transition probabilities
- $R : X \times \mathcal{A}$ is the reward function
- $\mathcal{O}$ is the set of observations
- Z is a set of conditional observation probabilities
- T is the time horizon

At each time period, the environment is in some state $s \in X$ which cannot be directly observed. The initial state is sampled from a distribution $P(s)$. The agent takes an action $a \in \mathcal{A}$ which causes the environment to transition to state $s' \in X$ with probability $\Omega(s, a, s') = P(s_{t+1} = s' | s_t = s, a_t = a)$. The agent receives a reward $R(s, a)$. On reaching the new state s', it receives an observation $o \in \mathcal{O}$ according to the probability $Z(s', a, o) = P(o_{t+1} = o | s_{t+1} = s', a_t = a)$.

A *history* $\psi \in \Psi$ is a sequence of actions and observations $\psi_t = \{< o_1 >, < a_1, o_2 >, \ldots, < a_{t-1}, o_t >\}$. Note that the initial history $\psi_t =< o_1 >$ is simply the observation at the initial timestep. The history ψ_t captures all information required to express the belief over state. The belief $P(s_{t+1} | \psi_{t+1})$ can be computed recursively applying Bayes' rule

$$\eta \, Z(s_{t+1}, a_t, o_{t+1}) \sum_{s_t \in X} \Omega(s_t, a_t, s_{t+1}) \, P(s_t | \psi_t)$$

where η is a normalization constant.

The history can then also be used to compute an update $P(\psi_{t+1}|\psi_t, a_t)$:

$$\sum_{s_t \in X} \sum_{s_{t+1} \in X} P(s_t|\psi_t) \Omega\left(s_t, a_t, s_{t+1}\right) Z\left(s_{t+1}, a_t, o_{t+1}\right)$$

The agent's action selection behaviour can be explained by a policy $\pi(\psi_t) \in \Pi$ that maps history ψ_t to action a_t.

Let the state and history distribution induced by a policy π after t timesteps be $P(s, \psi|\pi, t)$. The value of a policy π is the expected cumulative reward for executing π for T timesteps on the induced state and history distribution

$$J\left(\pi\right) = \sum_{t=1}^{T} \mathbb{E}_{s_t, \psi_t \sim P(s,\psi|\pi,t)} \left[R\left(s_t, \pi(\psi_t)\right)\right] \tag{7.3}$$

The optimal policy maximizes the expected cumulative reward, i.e $\pi^* \in \arg\max_{\pi \in \Pi} J\left(\pi\right)$.

Given a starting history ψ, let $P(s', \psi'|\psi, \pi, i)$ be the induced state history distribution after i timesteps. The value of executing a policy π for t time steps from a history ψ is the expected cumulative reward:

$$\tilde{V}_t^{\pi}(\psi) = \sum_{i=1}^{t} \mathbb{E}_{s_i, \psi_i \sim P(s', \psi'|\psi, \pi, i)} \left[R\left(s_i, \pi(\psi_i)\right)\right] \tag{7.4}$$

The state-action value function $\tilde{Q}_t^{\pi}(\psi_t, a_t)$ is defined as the expected sum of one-step-reward and value-to-go:

$$\begin{aligned} \tilde{Q}_t^{\pi}(\psi, a) =& \mathbb{E}_{s \sim P(s|\psi)} \left[R\left(s, a\right)\right] + \\ & \mathbb{E}_{\psi' \sim P(\psi'|\psi, a)} \left[\tilde{V}_{t-1}^{\pi}(\psi')\right] \end{aligned} \tag{7.5}$$

7.3.2 Mapping informative path planning to POMDPs

We now map IPP problems Hidden-Unc and Hidden-Con to a POMDP. The state is defined to contain all information that is required to define the reward, observation and transition functions. Let the state be the set of nodes visited and the underlying world, $s_t = \{v_1, \ldots, v_t, \phi\}$. At the start of an episode, a world is sampled from a prior distribution $\phi \sim P(\phi)$ along with a graph $\mathcal{G} \sim P(\mathcal{G})$. The initial state is assigned by setting $s_1 = \{v_1, \phi\}$. Note that the state s_t is partially observable due to the hidden world map ϕ.

We define the action $a_t = v_{t+1}$ to be the next node to visit. We are now ready to map the utility and travel cost to the reward function definition. Given the agent is in state s_t and has executed a_t, we can extract the path $\xi = (v_1, v_2, \ldots, v_{t+1})$ and the underlying world ϕ. Hence we can compute the utility function $\mathcal{F}\left(\xi, \phi\right)$. We can also compute the travel cost function $\mathcal{T}\left(\xi, \phi\right)$.

Before we define the reward function, we note that for Problem Hidden-Con not all actions are feasible at all times due to connectivity of the graph and constraints due to travel cost. Hence we can define a feasible set of actions $\mathcal{A}_{\text{feas}}\left(s\right) \subset \mathcal{A}$ for a state as follows

$$\mathcal{A}_{\text{feas}}\left(s\right) = \{a \mid a \in \mathcal{A}, (v_t, v_{t+1}) \in \mathcal{E}, \mathcal{T}\left(\xi, \phi\right) \leq B\} \tag{7.6}$$

For Problem Hidden-Unc, let $\mathcal{A}_{\text{feas}}\left(s\right) = \mathcal{A}$.

Since the objective is to maximize the cumulative reward function, we define the reward to be proportional to the marginal utility of visiting a node. Given a node $v \in \mathcal{V}$, a path ξ and world ϕ, the marginal gain of the utility function $\mathcal{F}$ is $\Delta_{\mathcal{F}}(v \mid \xi, \phi) = \mathcal{F}(\xi \cup \{v\}, \phi) - \mathcal{F}(\xi, \phi)$. The one-step-reward function, $R(s, a)$, is defined as the marginal gain of the utility function. Additionally, the reward is set to $-\infty$ whenever an infeasible action is selected. Hence:

$$R(s, a) = . \begin{cases} \Delta_{\mathcal{F}}(a \mid \xi, \phi) & \text{if } a \in \mathcal{A}_{\text{feas}}(s) \\ -\infty & \text{otherwise} \end{cases} \tag{7.7}$$

The state transition function, $\Omega(s, a, s')$, is defined as the deterministic function which sets $v_{t+1} = a_t$. We define the observation to be the measurement $o_t = y_t$ and the observation model Z to be a deterministic function $o_t = \mathcal{H}(v_t, \phi)$.

Note that the history ψ_t, the sequence of actions and observations, is captured in the sequence of nodes visited $\{v_i\}_{i=1}^t$ and measurements received $\{y_i\}_{i=1}^t$. In our implementation, we encode this information in an occupancy map. The information gathering policy $\pi(\psi_t)$ maps this history to an action a_t, the sensing location to visit.

7.3.3 Mapping search based planning to POMDPs

We now map the problem of computing a heuristic policy to a POMDP setting. Let the state be the open list and the underlying world, $s_t = \{\mathcal{O}_t, \phi\}$. At the start of an episode, a world is sampled from a prior distribution $\phi \sim P(\phi)$ along with a start state v_s. The initial state is assigned by setting $s_1 = \{v_s, \phi\}$. Note that the state s_t is partially observable due to the hidden world map ϕ.

We define the action a_t as the vertex $v \in \mathcal{O}_t$ that is to be expanded by the search. The state transition function, $\Omega(s, a, s')$, is defined as the deterministic function which sets $\mathcal{O}_{t+1}$ by querying $\texttt{Expand}(v, \texttt{Succ}, \texttt{Eval}, \phi)$. The one-step-reward function, $R(s, a)$, is defined as -1 for every (s_t, a_t) until the goal is added to the open list. Additionally, the reward is set to $-\infty$ whenever an infeasible action is selected. Hence:

$$R(s, a) = . \begin{cases} -\infty & \text{if } a \notin \mathcal{O} \\ 0 & \text{if } v_g \in \mathcal{O} \\ -1 & \text{otherwise} \end{cases} \tag{7.8}$$

We define the observation to be the successor nodes and invalid edges, i.e. $o_t = \{\mathcal{V}_{\text{succ}}, \mathcal{E}_{\text{inv}}\}$ and the observation model Z to be a deterministic function $(\mathcal{V}_{\text{succ}}, \mathcal{E}_{\text{inv}}) = \texttt{Expand}(v, \texttt{Succ}, \texttt{Eval}, \phi)$.

Note that the history, the sequence of actions and observations, is contained in the information present in the concatenation of all lists, i.e $\psi_t = \{\mathcal{O}, \mathcal{C}, \mathcal{I}\}$. The heuristic is a policy $\pi(\psi_t)$ that maps this history to an action a_t, the vertex to expand.

Note that it is more natural to think of this problem as minimizing a one-step-cost than maximizing a reward. Hence when we subsequently refer to this problem instance, we refer to the cost $c(s, a) = -R(s, a)$ and the cost-to-go $\tilde{Q}_t^{\pi}(\psi, a)$. This only results in a change from maximization to minimization.

7.3.4 What makes these POMDPs intractable?

A natural question to ask if these problems can be solved by state-of-the-art POMDP solvers such as POMCP [Silver and Veness, 2010] or DESPOT [Somani et al., 2013]. While such solvers are very effective at scaling up and solving large scale POMDPs, there are a few reasons why there are not immediately applicable to our problem.

Firstly, these methods require a lot of online effort. In the case of search based planning, the effort required to plan in belief space defeats the purpose of a heuristic all together. In the case of informative path planning, the observation space is very large and belief updates would be time consuming.

Secondly, since both methods employ a particle filter based approach to tracking plausible world maps, they both are susceptible to a realizability problem. Its unlikely that there will be a world map particle that will explain all observations. That being said, the world maps can explain local correlations in observations. For example, when planning indoors the world maps can explain correlations in observations made at intersection of corridors. Hence, we would like to generalize across these local submaps.

7.4 Imitation of clairvoyant oracles

A possible approach is to employ model free Q-learning [Mnih et al., 2015] by featurizing the history ψ_t and collecting on-policy data. However, given the size of Ψ, this may require a large number of samples. Another strategy is to parameterize the policy class and employ policy improvement [Peters and Schaal, 2006] techniques. However, such techniques when applied to POMDP settings may lead to poor local minima due to poor initialization. Imitation learning [Ross and Bagnell, 2010] offers a more effective strategy than reinforcement learning in scenarios where there exist good policies for the original problem, however these policies cannot be executed online (e.g due to computational complexity) hence requiring imitation via an offline training phase. In this section, we extend this principle and show how imitation of *clairvoyant oracles* enables efficient learning of POMDP policies.

7.4.1 Imitation learning

We now formally define imitation learning as applied to our setting. Given a policy π, we define the distribution of histories $P(\psi|\pi)$ induced by it (termed as *roll-in*). Let $\mathcal{L}(\psi, \pi)$ be a loss function that captures how well policy π imitates an oracle. Our goal is to find a policy $\hat{\pi}$ which minimizes the expected loss as follows.

$$\hat{\pi} = \arg\min_{\pi \in \Pi} \mathbb{E}_{\psi \sim P(\psi|\pi)} \left[\mathcal{L}(\psi, \pi) \right] \tag{7.9}$$

This is a non-i.i.d supervised learning problem. Ross et al. [2011] propose ForwardTraining to train a non-stationary policy (one policy $\hat{\pi}_t$ for each timestep), where each policy $\hat{\pi}_t$ can be trained on distributions induced by previous policies $(\hat{\pi}_1, \ldots, \hat{\pi}_{t-1})$. While this solves the problem exactly, it is impractical given a different policy is needed for each timestep. For training a single policy, Ross et al. [2011] show how such problems can be reduced to no-regret

online learning using dataset aggregation (DAgger). The loss function they consider $\mathcal{L}$ is a mis-classification loss with respect to what the expert demonstrated. Ross and Bagnell [2014] extend the approach to the reinforcement learning setting where $\mathcal{L}$ is the reward-to-go of an oracle reference policy by aggregating *values* to imitate (AggreVaTe).

7.4.2 Solving POMDP via imitation of a clairvoyant oracle

To examine the applicability of imitation learning in the POMDP framework, we compare the loss function (7.9) to the action value function (7.5). We see that a good candidate loss function $\mathcal{L}(\psi, \pi)$ should incentivize maximization of $\tilde{Q}^{\pi}_{T-t+1}(\psi, \pi(\psi))$. A suitable approximation of the optimal value function $\tilde{Q}^{\pi^*}_{T-t+1}$ that can be computed at train time would suffice. However, we cannot resort to oracles that explicitly reasoning about the belief over states $P(s_t|\psi_t)$, let alone planning in this belief space due to tractability issues.

In this work, we leverage the fact that for our problem domains, we have access to the true state s_t at train time. This allows us to define oracles that are *clairvoyant* - that can observe the state at training time and plan actions using this information.

Definition 7.1 (Clairvoyant Oracle)**.** A clairvoyant oracle $\pi_{\mathrm{OR}}(s)$ is a policy that maps state s to action a with an aim to maximize the cumulative reward of the underlying MDP $(X, \mathcal{A}, \Omega, R, T)$.

The oracle policy defines an equivalent action value function *defined on the state* as follows

$$Q^{\pi_{\mathrm{OR}}}_t(s, a) = R(s, a) + \mathbb{E}_{s' \sim P(s'|s,a)} \left[V^{\pi_{\mathrm{OR}}}_{t-1}(s') \right] \tag{7.10}$$

Our approach is to imitate the oracle during training. This implies that we train a policy $\hat{\pi}$ by solving the following optimization problem

$$\hat{\pi} = \arg\max_{\pi \in \Pi} \mathbb{E}_{\substack{t \sim U(1:T), \\ s_t, \psi_t \sim P(s, \psi|\pi, t)}} \left[Q^{\pi_{\mathrm{OR}}}_{T-t+1}(s_t, \pi(\psi_t)) \right] \tag{7.11}$$

While we will define training procedures to concretely realize (7.11) later in Section 7.5, we offer some intuition behind this approach. Since the oracle π_{OR} knows the state s, it has appropriate information to assign a value to an action a. The policy $\hat{\pi}$ attempts to imitate this action from the partial information content present in its history ψ. Due to this realization error, the policy $\hat{\pi}$ visits a different state, updates the history, and queries the oracle for the best action. Hence while the learnt policy can make mistakes in the beginning of an episode, with time it gets better at imitating the oracle.

7.4.3 Analysis using a hallucinating oracle

The learnt policy imitates a clairvoyant oracle that has access to more information (state s compared to history ψ). This results in a large realizability error which is due to two terms - firstly the information mismatch between s and ψ, and secondly the expressiveness of feature space. This realizability error can be hard to bound making it difficult to apply the performance guarantee analysis of Ross and Bagnell [2014]. It is also not desirable to obtain a performance bound with respect to the *clairvoyant oracle* $J(\pi_{\mathrm{OR}})$.

To alleviate the information mismatch, we take an alternate approach to analyzing the learner by introducing a purely hypothetical construct - a *hallucinating oracle*.

Definition 7.2 (Hallucinating Oracle)**.** A hallucinating oracle $\tilde{\pi}_{\mathrm{OR}}$ computes the instantaneous posterior distribution over state $P(s|\psi)$ and returns the expected clairvoyant oracle action value.

$$\tilde{Q}^{\tilde{\pi}_{\mathrm{OR}}}_{T-t+1}(\psi, a) = \mathbb{E}_{s\sim P(s|\psi)}\left[Q^{\pi_{\mathrm{OR}}}_{T-t+1}(s, a)\right] \tag{7.12}$$

We show that by imitating a clairvoyant oracle, the learner effectively imitates the corresponding hallucinating oracle

Lemma 7.1. The **offline** imitation of **clairvoyant** oracle (7.11) is equivalent to **online** imitation of a **hallucinating** oracle as shown

$$\hat{\pi} = \arg\max_{\pi\in\Pi} \mathbb{E}_{\substack{t\sim U(1:T),\\ \psi_t\sim P(\psi|\pi,t)}}\left[\tilde{Q}^{\tilde{\pi}_{\mathrm{OR}}}_{T-t+1}(\psi_t, \pi(\psi_t))\right]$$

Proof. Refer to Choudhury et al. [2017a]. □

Note that a hallucinating oracle uses the same information content as the learnt policy. Hence the realization error is purely due to the expressiveness of the feature space. The empirical risk of imitating the hallucinating oracle will be significantly lower than the risk of imitating the clairvoyant oracle.

Lemma 7.1 now allows us to express the performance of the learner with respect to a hallucinating oracle. This brings us to the key question - how good is a hallucinating oracle? Upon examining (7.12) we see that this oracle is equivalent to the well known QMDP policy first proposed by Littman et al. [1995]. The QMDP policy ignores observations and finds the $Q_{\mathrm{MDP}}(s, a)$ values of the underlying MDP. It then estimates the action value by taking an expectation on the current belief over states $P(s|\psi)$. This estimate amounts to assuming that any uncertainty in the agent's current belief state will be gone after the next action. Thus, the action where long-term reward from all states (weighed by the probability) is largest will be the one chosen.

Littman et al. [1995] points out that policies based on this approach are remarkably effective. This has been verified by other works such as Koval et al. [2014] and Javdani et al. [2015]. This naturally leads to the question of why we cannot directly apply QMDP to our problem. The QMDP approach requires explicitly sampling from the posterior over states online - a step that we cannot tractably compute as discussed in Section 7.3.4. However, by imitating clairvoyant oracles, we implicitly obtain such a behaviour.

Imitation of clairvoyant oracles has been shown to be effective in other domains such as receding horizon control via imitating MPC methods that have full information [Kahn et al., 2017]. Sun et al. [2017] show how the partially observable acrobot can be solved by imitation of oracles having full state. Karkus et al. [2017] introduce imitation of QMDP in a deep learning architecture to train POMDP policies end to end.

The connection with a hallucinating oracle also provides valuable insight into potential failure situations. Littman et al. [1995] point out that policies based on this approach will not take actions to gain information. We discuss such situations in Section 7.8.

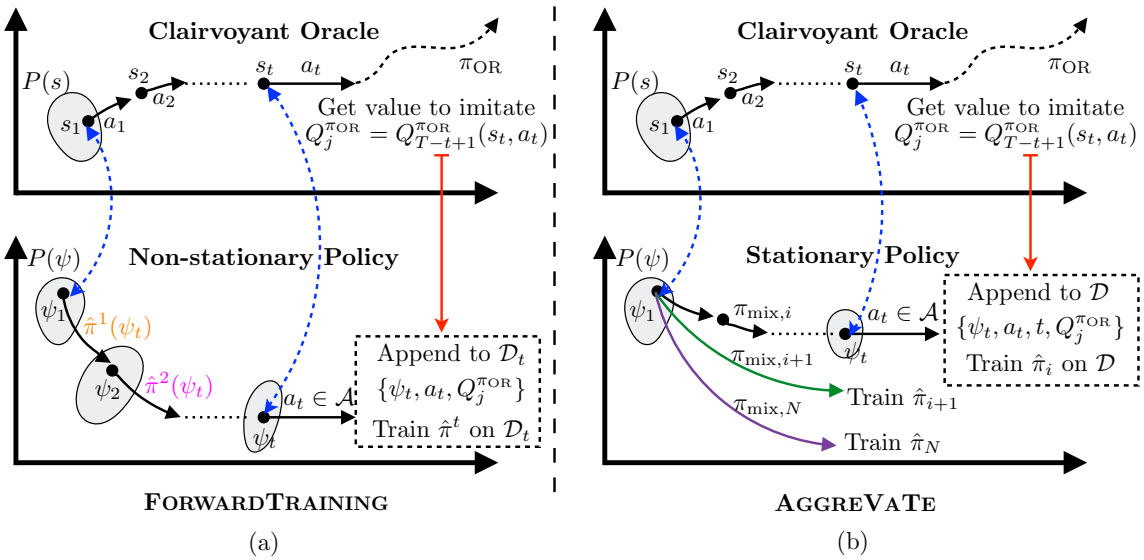


Figure 7.4: Overview of the two approaches for training policies. (a) ForwardTraining is used to train a non-stationary policy, i.e a sequence of policies $\hat{\pi}^1, \ldots, \hat{\pi}^T$ at each time-step. To train a policy at time-step t, a state s is sampled from initial distribution $P(s)$. The policies $\hat{\pi}^1, \ldots, \hat{\pi}^{t-1}$ are then used to roll-in to get (s_t, ψ_t). The oracle is queried to get $Q_{T-t+1}^{\pi_{\text{OR}}}(s_t, a_t)$ which is then used to update the dataset and train policy $\hat{\pi}^t$. (b) AggreVaTe is used to train a stationary policy. The training process is iterative where dataset collection is interleaved with learning. At iteration i, a mixture policy $\pi_{\text{mix},i}$ is used to roll-in to get (s_t, ψ_t). The oracle is queried to get $Q_{T-t+1}^{\pi_{\text{OR}}}(s_t, a_t)$. The data is then aggregated to the whole dataset which is used to update the entire policy $\hat{\pi}^i$.

7.5 Approach

7.5.1 Algorithms

We introduced imitation learning and its applicability to POMDPs in Section 7.4. We now present a set of algorithms to concretely realize the process. The overall idea is as follows - we are training a policy $\hat{\pi}(\psi)$ that maps features extracted from the history ψ to an action a. The training objective is to imitate a clairvoyant oracle that has access to the corresponding full state s. In order to define concrete algorithms, we need to reason about two classes of policies - non-stationary and stationary.

Non-stationary policy

For the non-stationary case, we have a policy for each timestep $\hat{\pi}^1, \ldots, \hat{\pi}^T$. The motivation for adopting such a policy class is that the problems arising from the non i.i.d distribution immediately disappears. Such a policy class can be trained using the ForwardTraining algorithm [Ross et al., 2011] which sequentially trains each policy on the distribution of features induced from the previous set of policies. Hence the training problem for each policy at timestep t is reduced to supervised learning.

Algorithm 10 describes the ForwardTraining procedure to train the non-stationary policy.

Algorithm 10: ForwardTraining (Non-stationary policy)

```
1  for t = 1 to T do
2      Initialize D_t ← ∅ ;
3      for j = 1 to m do
4          Sample initial state s_1 from dataset P(s);
5          Execute policy π̂^1, ..., π̂^{t-1} to reach (s_t, ψ_t). ;
6          Execute any action a_t ∈ A. ;
7          Collect value to go Q_j^{π_OR} = Q_{T-t+1}^{π_OR}(s_t, a_t) ;
8          D_t ← D_t ∪ {ψ_t, a_t, Q_j^{π_OR}} ;
9      Train cost-sensitive classifier π̂^t on D_t ;
10 return Set of policies for each time step π̂^1, ..., π̂^T .;
```

The policies are trained in a sequential manner. At each time-step t, the previously trained policies $\hat{\pi}^1, \ldots, \hat{\pi}^{t-1}$ are used to create a dataset of ψ_t by rolling-in (Lines 1–5). For each such datapoint ψ_t, there is a corresponding state s_t. A random action a_t is sampled and the oracle is queried for the cost-to-go $Q^{\pi_{\text{OR}}}_{T-t+1}(s_t, a_t)$ (Line 7). This is then added to the dataset $\mathcal{D}_t$ which is used to train the policy $\hat{\pi}^t$. This is illustrated in Fig. 7.4.

We can state the following property about the training process

Theorem 7.2. ForwardTraining has the following guarantee

$$J\left(\hat{\pi}\right) \geq J\left(\tilde{\pi}_{\text{OR}}\right) - 2T\sqrt{\mathcal{A}\,\varepsilon_{\text{class}}} + T\varepsilon_{\text{or}}$$

where $\varepsilon_{\text{class}}$ is the regression error of the learner and ε_{or} is the local oracle suboptimality.

Proof. Refer to Choudhury et al. [2017a]. □

However, there are several drawbacks to using a non-stationary policy. Firstly, it is impractical to have a different policy for each time-step as it scales with T. While this might be a reasonable approach when T is small (e.g. sequence classification problems [Cohen and Carvalho, 2005]), in our applications T can be fairly large. Secondly, and more importantly, each policy operates on data for only that time-step, thus preventing generalizations across timesteps. Each policy sees only $\frac{\mathcal{D}}{T}$ fraction of the training data. This leads to a high empirical risk.

Stationary policy

A single stationary policy $\hat{\pi}$ enjoys the benefit of learning on data across all timesteps. However, the non i.i.d data distribution implies the procedure of data collection and training cannot be decoupled - the learner must be involved in the data collection process. Ross and Bagnell [2014] show that such policies can be trained by reducing the propblem to a no-regret online learning setting. They present an algorithm, AggreVaTe that trains the policy in an interactive fashion where data is collected by a mixture policy of the learner and the oracle, the data is then *aggregated* and the learner is trained on this aggregated data. This process is repeated.

Algorithm 11 describes the AggreVaTe procedure to train the stationary policy. To overcome the non i.i.d distribution issue, the algorithm interleaves data-collection with learning and

Algorithm 11: AggreVaTe (Stationary policy)

1 Initialize $\mathcal{D} \leftarrow \emptyset$, $\hat{\pi}_1$ to any policy in Π ;
2 **for** $i = 1$ ***to*** N **do**
3 Initialize sub-dataset $\mathcal{D}_i \leftarrow \emptyset$;
4 Let roll-in policy be $\pi_{\text{mix},i} = \beta_i \pi_{\text{OR}} + (1 - \beta_i)\hat{\pi}_{i-1}$;
5 Collect m data points as follows: ;
6 **for** $j = 1$ ***to*** m **do**
7 Sample initial state s_1 from dataset $P(s)$;
8 Sample uniformly $t \in \{1, 2, \ldots, T\}$;
9 Execute $\pi_{\text{mix},i}$ up to time $t - 1$ to reach (s_t, ψ_t) ;
10 Execute any action $a_t \in \mathcal{A}$;
11 Collect value-to-go $Q_j^{\pi_{\text{OR}}} = Q_{T-t+1}^{\pi_{\text{OR}}}(s_t, a_t)$;
12 $\mathcal{D}_i \leftarrow \mathcal{D}_i \cup \{\psi_t, a_t, t, Q_j^{\pi_{\text{OR}}}\}$;
13 Aggregate datasets: $\mathcal{D} \leftarrow \mathcal{D} \bigcup \mathcal{D}_i$;
14 Train cost-sensitive classifier $\hat{\pi}_{i+1}$ on $\mathcal{D}$;
15 **return** *best* $\hat{\pi}_i$ *on validation*;

iteratively trains a set of policies $(\hat{\pi}_1, \hat{\pi}_2, \ldots, \hat{\pi}_N)$. Note that these iterations are not to be confused with time steps - they are simply learning iterations. A policy $\hat{\pi}_i$ is valid for all timesteps. At iteration i, data is collected by rolling-in with a mixture of the learner and the oracle policy (Lines 1–9). The mixing fraction is chosen to be $\beta_i = (1 - \alpha)^{i-1}$. Mixing implies flipping a coin with bias β_i and executing the oracle if heads comes up. A random action a_t is sampled and the oracle is queried for the cost-to-go $Q_{T-t+1}^{\pi_{\text{OR}}}(s_t, a_t)$ (Line 11).

The key step is to ensure that *data is aggregated*. The motivation for doing so arises from the fact that we want the learner to do well on the distribution it induces. Ross and Bagnell [2014] show that this can be posed as the mixture of learners $(\hat{\pi}_1, \hat{\pi}_2, \ldots, \hat{\pi}_N)$ doing well on the induced loss sequences $l_i(\pi)$ at every iteration. If we were to treat each iteration as a game in an online adversarial learning setting, this would be equivalent to having bounded regret with respect to the best policy in hindsight on the loss sequence $(l_1, l_2, \ldots, l_N)$. The strategy of dataset aggregation is an instance of follow the leader and hence has bounded regret. Hence, data is appended to the original dataset and used to train an updated learner $\hat{\pi}_{i+1}$ (Lines 13–14).

AggreVaTe can be shown to have the following guarantee

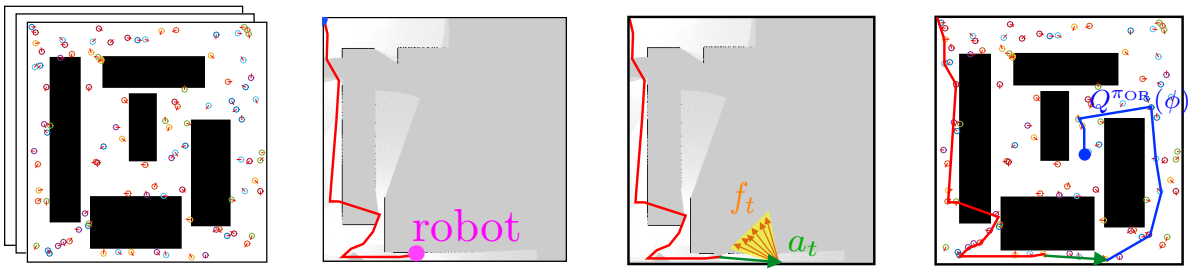
Theorem 7.3. N iterations of AggreVaTe, collecting m regression examples per iteration guarantees that with probability at least $1 - \delta$

$$\begin{aligned} J(\hat{\pi}) \geq & J(\tilde{\pi}_{\text{OR}}) \\ & - 2T\sqrt{|\mathcal{A}|\left(\varepsilon_{\text{class}} + \varepsilon_{\text{reg}} + \mathcal{O}\left(\sqrt{\log 1/\delta / Nm}\right)\right)} \\ & - \mathcal{O}\left(\frac{R\,T \log T}{N}\right) + T\varepsilon_{\text{or}} \end{aligned}$$

where $\varepsilon_{\text{class}}$ is the empirical regression regret of the best regressor in the regression class on the aggregated dataset, ε_{reg} is the empirical online learning average regret on the sequence of training examples, R is the range of oracle action value and ε_{or} is the local oracle suboptimality.

Proof. Refer to Choudhury et al. [2017a]. □

7.5.2 Application to informative path planning



Sample a world ϕ from database $P(\phi)$ | Roll-in with policy π_{mix} to get history ψ_t | Execute a random action a_t and featurize (ψ_t, a_t) as f_t | Roll out with oracle π_{OR} add $\{f_t, Q^{\pi_{\text{OR}}}\}$ to data

Figure 7.5: An overview of QvalAgg in IPP where a learner $\hat{\pi}$ is trained to imitate a clairvoyant oracle π_{OR}. There are 4 key steps. Step 1: A world map ϕ is sampled from database representing $P(\phi)$. Step 2: A mixture policy π_{mix}, of the learner and oracle is used to roll-in on ϕ to a timestep t to get history ψ_t. Step 3: A random action a_t is chosen and (ψ_t, a_t) is featurized as f_t. Step 4: A clairvoyant oracle π_{OR} is given full access to world map ϕ to compute the cumulative reward to go $Q^{\pi_{\text{OR}}}$. The pair $(f_t, Q^{\pi_{\text{OR}}})$ is added to data to update the learner. This process is repeated to train a sequence of learners.

We now consider the applicability of Algorithm 10 and Algorithm 11 for learning a policy to plan informative paths. We refer to the mapping of the IPP problem to a POMDP defined in Section 7.3.2. We first need to define a clairvoyant oracle in this context. Recall that the state $s_t = \{v_1, \ldots, v_t, \phi\}$ is the set of nodes visited and the underlying world. A clairvoyant oracle takes a state action pair (s_t, a_t) as input and computes a value. Depending on whether we are solving Problem Hidden-Unc or Hidden-Con, we explore two different kinds of oracles:

1. *Clairvoyant One-step-reward*
2. *Clairvoyant Reward-to-go*

Solving Hidden-Unc by imitating clairvoyant one-step-reward

We first define a Clairvoyant One-step-reward oracle in the IPP framework.

Definition 7.3 (Clairvoyant One-step-reward)**.** A Clairvoyant One-step-reward returns an action value $Q_t^{\pi_{\text{OR}}}(s, a) = R(s, a)$ that considers only the one-step-reward. In the context of Hidden-Unc, it uses the world map ϕ, the curent path $\{v_1, \ldots, v_t\}$, the next node to visit $v_{t+1} = a_t$ to compute the value $Q^{\text{OR}}(\phi, \{v_1, \ldots, v_t\}, v_{t+1})$ as the marginal gain in utility, i.e.

$$\Delta_{\mathcal{F}}(v_{t+1} \mid \{v_1, \ldots, v_t\}, \phi)$$

To motive the use of Clairvoyant One-step-reward, we refer to the discussion on the structure of the Problem Hidden-Unc in Section 7.2. We assume that the utility function is *adaptive monotone submodular* - it has the property of montonicity and diminishing returns under the belief over world maps. This property implies the following

1. *Adaptive Monotonicity*: The expected value of the utility can only increase on adding a node, i.e.
$$\mathbb{E}_{\phi\sim P(\phi|\psi)}\left[\Delta_{\mathcal{F}}\left(v \mid \mathcal{V}_{\psi}, \phi\right)\right] \geq 0$$
for all $v \in \mathcal{V}$, where $\psi = \{v_i\}_{i=1}^{p}, \{y_i\}_{i=1}^{p}$, and $\mathcal{V}_{\psi} = \{v_i\}_{i=1}^{p}$.

2. *Adaptive Submodularity*: The expected gain in adding a node diminshes as more nodes are visited, i.e.
$$\mathbb{E}_{\phi\sim P(\phi|\psi)}\left[\Delta_{\mathcal{F}}\left(v \mid \mathcal{V}_{\psi}, \phi\right)\right] \geq \mathbb{E}_{\phi\sim P(\phi|\psi')}\left[\Delta_{\mathcal{F}}\left(v \mid \mathcal{V}_{\psi'}, \phi\right)\right]$$
for all $v \in \mathcal{V}$, where $\psi \subseteq \psi'$ (history ψ is contained in history ψ')

For such functions, Golovin and Krause [2011] show that greedily selecting vertices to visit is near-optimal. We use this property to show that the Clairvoyant One-step-reward induces a one-step-oracle which is equivalent to the greedy policy and hence near optimal. This implies the following Lemma

Theorem 7.4. *N iterations of AggreVaTe with Clairvoyant one-step-reward collecting m regression examples per iteration guarantees that with probability at least $1-\delta$*
$$J\left(\hat{\pi}\right) \geq \left(1-\frac{1}{e}\right) J\left(\pi^{*}\right) - 2T\sqrt{|\mathcal{A}|\left(\varepsilon_{\text{class}} + \varepsilon_{\text{reg}} + \mathcal{O}\left(\sqrt{\log 1/\delta / Nm}\right)\right)} - \mathcal{O}\left(\frac{R\,T\log T}{N}\right)$$
where $\varepsilon_{\text{class}}$ is the empirical regression regret of the best regressor in the regression class on the aggregated dataset, ε_{reg} is the empirical online learning average regret on the sequence of training examples, R is the maximum range of one-step-reward.

Proof. Refer to Choudhury et al. [2017a]. □

We will shown in Section 7.6 that such policies are remarkably effective. An added benefit of imitating the Clairvoyant One-step-reward is that the empirical classification loss $\varepsilon_{\text{class}}$ is lower since only the expected one-step-reward of an action needs to be learnt.

Solving Hidden-Con by imitating clairvoyant reward-to-go

Unfortunately, Problem Hidden-Con does not posses the adaptive-submodular property of Hidden-Unc due to the introduction of the travel cost. Hence imitating the one-step-reward is no longer appropriate. We define the Clairvoyant Reward-to-go oracle for this problem class

Definition 7.4 (Clairvoyant Reward-to-go)**.** A Clairvoyant Reward-to-go returns an action value $Q_t^{\pi_{\text{OR}}}(s,a)$ that corresponds to the cumulative reward obtained by executing a and then following the oracle policy π_{OR}. In the context of Hidden-Con, it uses the world map ϕ, the current path $\{v_1,\ldots,v_t\}$, the next node to visit $v_{t+1} = a_t$ to solve the problem Known-Con and compute

a future sequence of nodes $\{v_{t+2}, \ldots, v_T\}$. This provides the value $Q^{\mathrm{OR}}(\phi, \{v_1, \ldots, v_t\}, v_{t+1})$ as the marginal gain

$$\Delta_{\mathcal{F}}\left(\{v_{t+1}, \ldots, v_T\} \mid \{v_1, \ldots, v_t\}, \phi\right)$$

The corresponding oracle policy π_{OR} is obtained by following the computed path.

Note that solving Known-Con is NP-Hard and even the best approximation algorithms require some computation time. Hence the calls to the oracle must be minimized.

Training and testing procedure

We now present concrete algorithms to realize the training procedure. Given the two axes of variation - problem and policy type - we have four possible algorithms

1. RewardFT: Imitate one-step-reward using non-stationary policy by ForwardTraining (Algorithm 10)
2. QvalFT: Imitate reward-to-go using non-stationary policy by ForwardTraining (Algorithm 10)
3. RewardAgg: Imitate one-step-reward using stationary policy by AggreVaTe (Algorithm 11)
4. QvalAgg: Imitate reward-to-go using stationary policy by AggreVaTe (Algorithm 11)

Table. 7.1 shows the algorithm mapping.

Table 7.1: Mapping from problem and policy type to algorithm

Policy \ **Problem**	Hidden-Unc	Hidden-Con
Non-stationary policy	RewardFT	QvalFT
Stationary policy	RewardAgg	QvalAgg

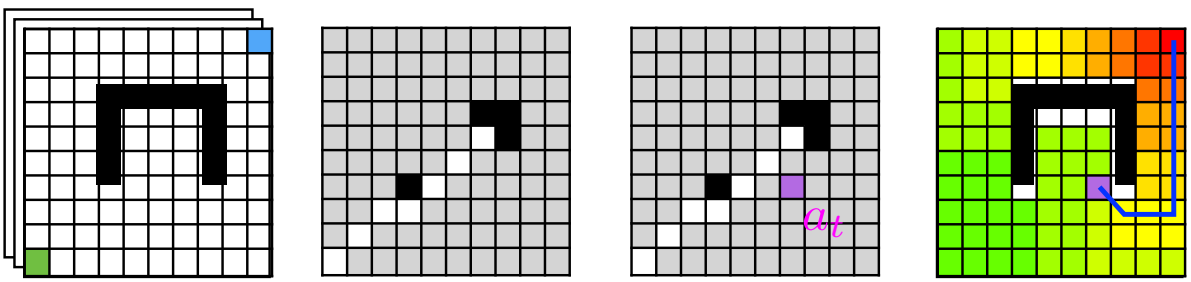
For completeness, we concretely define the training procedure for QvalAgg in Algorithm 12. The procedure for the remaining three algorithms can be inferred from this. The algorithm iteratively trains a sequence of policies $(\hat{\pi}_1, \hat{\pi}_2, \ldots, \hat{\pi}_N)$. At every iteration i, the algorithm conducts m episodes. In every episode a different world map ϕ and start vertex (v_s) is sampled from a database. The roll-in is conducted with a mixture policy $\pi_{\mathrm{mix},i}$ which blends the learner's current policy, $\hat{\pi}_{i-1}$ and the oracle's policy, π_{OR} using blending parameter β_i. The blending is done in an episodic fashion, with probability β_i the Clairvoyant Reward-to-go oracle is invoked to compute a path which is followed. With probability $1-\beta_i$, the learner is invoked for the whole episode. In a given episode, the roll-in is conducted to a timestep t which is uniformly sampled. At the end of the roll-in, we have a path $\{v_1, \ldots, v_t\}$ and a history ψ_t. A random action $a_t \in \mathcal{A}$ is sampled which defines the next vertex to visit $v_{t+1} = a_t$. The Clairvoyant Reward-to-go oracle is invoked with the world ϕ and the path already travelled $\{v_1, \ldots, v_t\}, v_{t+1}\}$. It then invokes a

Algorithm 12: QvalAgg

Initialize $\mathcal{D} \leftarrow \emptyset$, $\hat{\pi}_1$ to any policy in Π ;
for $i = 1$ ***to*** N **do**
  Initialize sub-dataset $\mathcal{D}_i \leftarrow \emptyset$;
  Let roll-in policy be $\pi_{\text{mix},i} = \beta_i \pi_{\text{OR}} + (1 - \beta_i)\hat{\pi}_i$;
  Collect m data points as follows: ;
  for $j = 1$ ***to*** m **do**
    Sample world ϕ from dataset $P(s)$;
    Sample start node v_s for $P(v_s)$;
    Sample uniformly $t \in \{1, 2, \ldots, T\}$;
    Execute $\pi_{\text{mix},i}$ up to time $t-1$ to get path $\{v_1, \ldots, v_t\}$ and history ψ_t ;
    Sample a random action $a_t \in \mathcal{A}$ as the next vertex to visit $v_{t+1} = a_t$;
    Invoke Clairvoyant Reward-to-go oracle to get $Q_j^{\pi_{\text{OR}}} = Q^{\text{OR}}\{\phi, \{v_1, \ldots, v_t\}, v_{t+1}\}$. ;
    $\mathcal{D}_i \leftarrow \mathcal{D}_i \cup \{\psi_t, a_t, t, Q_j^{\pi_{\text{OR}}}\}$;
  Aggregate datasets: $\mathcal{D} \leftarrow \mathcal{D} \bigcup \mathcal{D}_i$;
  Train cost-sensitive classifier $\hat{\pi}_{i+1}$ on $\mathcal{D}$;
return *best* $\hat{\pi}_i$ *on validation*

solver to Hidden-Con to complete the path and return the reward to go $Q_j^{\pi_{\text{OR}}}$. This history action pair (ψ_t, a_t) is projected to a feature space along with label $Q_j^{\pi_{\text{OR}}}$. The data is aggregated to the dataset which is eventually used to train policy $\hat{\pi}_{i+1}$. Fig. 7.5 illustrates this approach.

7.5.3 Application to search based planning



Sample a world ϕ from database $P(\phi)$ | Roll-in with policy π_{mix} to get history ψ_t | Execute a random action a_t and featurize (ψ_t, a_t) as f_t | Roll out with oracle π_{OR} add $\{f_t, Q^{\pi_{\text{OR}}}\}$ to data

Figure 7.6: An overview of SaIL in search based planning where a learner $\hat{\pi}$ is trained to imitate a clairvoyant oracle π_{OR}. There are 4 key steps. Step 1: A world map ϕ is sampled from database representing $P(\phi)$ along with start goal pair (v_s, v_g). Step 2: A mixture policy $\pi_{\text{mix},}$ of the learner and oracle is used to roll-in on ϕ to a timestep t to get history ψ_t which is the combination of open list, closed list and invalid edges. Step 3: A random vertex a_t from the open list is chosen and (ψ_t, a_t) is featurized as f_t. Step 4: A clairvoyant oracle π_{OR} is given full access to world map ϕ to compute the cumulative cost to go $Q^{\pi_{\text{OR}}}$. The pair $(f_t, Q^{\pi_{\text{OR}}})$ is added to data to update the learner. This process is repeated to train a sequence of learners.

We now consider the applicability of Algorithm 11 for heuristic learning in search based planning. Unlike the IPP problem domain, there is no incentive to use a non-stationary policy or imitate Clairvoyant One-step-rewards. Hence we only consider training a stationary policy imitating Clairvoyant Reward-to-go.

Algorithm 13: SAIL $(P(\phi), P(v_s, v_g), k)$

```
Initialize D ← ∅, π̂_1 to any policy in Π ;
for i = 1 to N do
    Initialize sub dataset D_i ← ∅ ;
    Collect mk data points as follows: ;
    for j = 1 to m do
        Sample world map φ ~ P(φ) ;
        Sample (v_s, v_g) ~ P(v_s, v_g) ;
        Invoke clairvoyant oracle planner to compute Q^{π_OR}(φ, v) ∀ v ∈ V;
        Sample uniformly k timesteps {t_1, t_2, ..., t_k} where each t_i ∈ {1, ..., T} ;
        Rollout search with π_{mix,i} = β_i π_OR + (1 − β_i) π̂_i ;
        At each t ∈ {t_1, t_2, ..., t_k} pick a random action a_t to get (ψ_t, v) ;
        Query oracle for Q^OR(φ, a_t) ;
        D_i ← D_i ∪ {ψ_t, a_t, t, Q^OR(φ, a_t)} ;
    Aggregate datasets: D ← D ⋃ D_i ;
    Train cost-sensitive classifier π̂_{i+1} on D ;
return best π̂_i on validation
```

We first need to define a clairvoyant oracle for this problem. Given access to the world map ϕ, the oracle has to solve for the optimal number of expansions to reach the goal. This allows us to define a *clairvoyant oracle planner* that employs a *backward* Dijkstra's algorithm, which given a world ϕ and a goal vertex v_g plans for the optimal path from every $v \in \mathcal{V}$ using dynamic programming.

Definition 7.5 (Clairvoyant Oracle Planner)**.** Given full access to the state s, which contains the open list $\mathcal{O}$ and world ϕ, and a goal v_g, the oracle planner encodes the cost-to-go from any vertex $v \in \mathcal{V}$ as the function $Q_t^{\pi_{\text{OR}}}(s, a)$ which implicitly defines an oracle policy, $\pi_{\text{OR}}(s) = \arg\min_{v \in \mathcal{O}} Q_t^{\pi_{\text{OR}}}(s, a)$.

The clairvoyant oracle planner provides a look-up table $Q^{\text{OR}}(\phi, v)$ for the optimal cost-to-go from any vertex irrespective of the current state of the search.

A key distinction between this oracle and the one defined for an IPP problem in Section 7.5.2 is that we are able to efficiently get the cost-to-go value for all states by dynamic programming - we do not need to repeatedly invoke the oracle. We exploit this fact by extracting multiple labels from an episode even though the oracle is invoked only once. Additionally, this allows us a better roll-in procedure where the oracle and learner are interleaved. We adapt the AGGREVATE framework to present an algorithm, *Search as Imitation Learning* (SAIL).

Algorithm 13, describes the SAIL framework which iteratively trains a sequence of policies $(\hat{\pi}_1, \hat{\pi}_2, \ldots, \hat{\pi}_N)$. For training the learner, we collect a dataset $\mathcal{D}$ as follows - At every iteration i, the agent executed m different searches (Algorithm 9). For every search, a different world ϕ and the pair (v_s, v_g) is sampled from a database. The agent then rolls-out a search with a mixture policy $\pi_{\text{mix},i}$ which blends the learner's current policy, $\hat{\pi}_i$ and the oracle's policy, π_{OR} using blending parameter β_i. During the search execution, at every timestep in a set of k uniformly sampled timesteps, we select a random action from the set of feasible actions and collect a datapoint $\{\psi_t, a_t, t, Q^{\text{OR}}(\phi, a_t)\}$. The policy $\pi_{\text{mix},i}$ is rolled out till the end of the episode and

Dataset	Sample World Maps	Problem	RewardAgg / QvalAgg	Average Entropy	Occlusion Aware	Unobserved Voxels	Rear Side Voxels	Rear Side Entropy
Concentrated Parallel Lines (2D)		HIDDEN-UNC	**(0.42,0.45)**	(0.20, 0.22)	(0.06, 0.09)	(0.20, 0.25)	(0.36, 0.41)	(0.30, 0.34)
		HIDDEN-CON	**(0.18, 0.27)**	(0.16, 0.20)	(0.07, 0.09)	(0.14, 0.18)	(0.19, 0.24)	**(0.21, 0.26)**
Distributed Blocks (2D)		HIDDEN-UNC	**(0.37,0.41)**	(0.26, 0.30)	(0.11, 0.16)	(0.22, 0.29)	(0.22, 0.29)	(0.24, 0.28)
		HIDDEN-CON	(0.20, 0.26)	**(0.21, 0.26)**	(0.11, 0.16)	(0.15, 0.20)	(0.15, 0.18)	(0.16, 0.19)
Poisson Forest of Circular Discs (2D)		HIDDEN-UNC	(0.58, 0.61)	**(0.59, 0.62)**	(0.49, 0.53)	(0.39, 0.45)	(0.53, 0.55)	(0.42, 0.47)
		HIDDEN-CON	**(0.54, 0.59)**	**(0.54, 0.59)**	(0.42, 0.46)	(0.34, 0.41)	(0.37, 0.43)	(0.39, 0.44)
Tabular World of Rectilinear Blocks (2D)		HIDDEN-UNC	**(0.43, 0.53)**	(0.31, 0.35)	(0.20, 0.26)	(0.28, 0.35)	(0.35, 0.44)	(0.25, 0.31)
		HIDDEN-CON	**(0.27, 0.33)**	(0.26, 0.29)	(0.18, 0.23)	(0.21, 0.28)	(0.18, 0.24)	(0.21, 0.27)
Bookshelves and Tables (3D)		HIDDEN-UNC	**(0.14, 0.31)**	(0.01, 0.04)	(0.01, 0.04)	(0.01, 0.04)	(0.01, 0.22)	(0.01, 0.19)
		HIDDEN-CON	**(0.05, 0.24)**	(0.01, 0.04)	(0.01, 0.04)	(0.01, 0.04)	(0.01, 0.22)	(0.01, 0.19)
Cluttered Construction Site (3D)		HIDDEN-UNC	**(0.14, 0.20)**	(0.01, 0.12)	(0.01, 0.09)	(0.01, 0.09)	(0.01, 0.11)	(0.01, 0.10)
		HIDDEN-CON	**(0.08, 0.12)**	(0.01, 0.12)	(0.01, 0.09)	(0.01, 0.09)	(0.01, 0.11)	(0.01, 0.10)
Office Desk and Chairs (3D)		HIDDEN-UNC	**(0.69, 0.80)**	(0.46, 0.59)	(0.51, 0.63)	(0.51, 0.63)	(0.59, 0.67)	(0.61, 0.72)
		HIDDEN-CON	**(0.55, 0.72)**	(0.46, 0.59)	(0.48, 0.63)	(0.48, 0.63)	(0.43, 0.52)	(0.41, 0.53)

Figure 7.7: Results for Problems HIDDEN-UNC and HIDDEN-CON on a spectrum of 2D and 3D exploration problems. The train size is 100 and test size is 10. Numbers are the confidence bounds (for 95% CI) of cumulative reward at the final time step. Algorithm with the highest median performance is emphasized in bold.

all the collected data is aggregated with dataset $\mathcal{D}$. At the end of N iterations, the algorithm returns the best performing policy on a set of held-out validation environment or alternatively, a mixture of $(\hat{\pi}_1, \hat{\pi}_2, \ldots, \hat{\pi}_N)$. Fig. 7.6 illustrates the SAIL framework.

Note that while the oracle is invoked once per ϕ, we obtain k datapoints - this is critical for speeding up training. We also note that even though the time complexity of `Select` is $O(|\mathcal{O}_t|)$ at timestep t, SAIL can have better overall complexity if it can achieve a squared reduction in number of expansions compared to uninformed search as discussed more in Choudhury et al. [2017a].

7.6 Experiments on informative path planning

In this section, we extensively evaluate our approach on a set of 2D and 3D informative path planning problems across a spectrum of synthetic and real world environments. We examine a class of informative path planning problem where a robot, equipped with a range limited sensor, possibly constrained by time and fuel resources, is tasked with 3D reconstruction of structures in the world. We choose a variety of environments to highlight the importance of adaptive behaviours for information gathering. Our implementation is open sourced for both MATLAB and C++ (`https://bitbucket.org/sanjiban/matlab_learning_info_gain`). For details on the problem, baseline heuristics, dataset and other learning details, we refer the reader to Choudhury et al. [2017a]

7.6.1 Analysis of results

Fig. 7.7 shows the utility of all algorithms on various synthetic datasets. The two numbers are lower and upper 95% confidence intervals of the episodic utility of each algorithm. The best performance on each dataset is highlighted. For Problem Hidden-Unc, RewardAgg is employed along with baseline heuristics. For Problem Hidden-Con, QvalAgg is employed with baseline heuristic augmented with motion penalization. The train size is 100 and test size is 10. We present a set of observations to interpret these results.

O 1. The learnt policy from RewardAgg/ QvalAgg has a consistently competitive performance across all datasets.

Fig. 7.7 shows the performance of all algorithms on a set of 2D and 3D datasets. We see that out of the 10 datasets, the learners perform better than any heuristic on 8. On 2 of the datasets, the Average Entropy heuristic outperforms the learner by a small margin. On examining the datasets, we see that the unknown space exploration behaviour of Average Entropy results in good performance in environments that either lack spatial correlation or contain objects distributed in the environment.

O 2. The performance of heuristics vary widely across datasets, however, the performance of the learner is robust.

We can see that the relative ranking of Average Entropy and Rear Side Voxel interchanges from Dataset 1 to 2. This motivates the need for adaptive policies that assign different utility to unknown cells conditioned on the environment in which the robot is operating. The learner's policy on the other hand adapts to different environments and hence maintains a consistently good performance. Interestingly, it also outperforms the heuristic pointwise across datasets, which is indicative of the fact that the adaptation happens during exploration as well.

O 3. The performance margin of RewardAgg in Problem Hidden-Unc as compared to heuristics is much larger than that of QvalAgg in Problem Hidden-Con

This is seen to be especially true in Dataset 1, 2 and 4. As conjectured in Section 7.5.2, this can be attributed to two reasons. Firstly, the near-optimality guarantee in Theorem 7.4 of imitating a Clairvoyant one-step-reward bounds the performance of the learner. Secondly, the empirical regression regret of imitating one step reward values will be much lower than trying to estimate the action values using features from the history ψ_t, i.e. it is easier to predict the immediate utility of going to a sensing location than trying to predict the future utility.

O 4. The performance of Average Entropy in the Poisson Forest dataset is at par with the learner.

The Poisson Forest dataset is created by sampling circles in the environment from a spatial Poisson distribution where the density of the forest is specified. The lack of spatial correlation, implies it is equally likely to find objects anywhere in the world - an assumption that Average Entropy optimizes.

7.6.2 Case study A: Adaptation to different environments

We created a set of 2D exploration problems to gain a better understanding of the learnt policies and baseline heuristics. We did this both for Problem Hidden-Unc (Fig. 7.8) and Hidden-Con (Fig. 7.9). The dataset comprises of 2D binary world maps, uniformly distributed nodes and a simulated laser. The problem details are $T = 30$ and $|\mathcal{A}| = 300$. The cost budget for Hidden-Con is $B = 2500$. The train size is 100, test size is 100. RewardAgg and QvalAgg is executed for 10 iterations.

Dataset 1: Parallel lines

We first examined Problem Hidden-Unc. Fig. 7.8 (a) shows a dataset created by applying random affine transformations to a pair of parallel lines. This dataset is representative of information being concentrated in an area in the environment, e.g. powerline inspection. Fig. 7.8 (c) shows a comparison of RewardAgg, RewardFT with baseline heuristics. While Rear Side Voxel outperforms Average Entropy, RewardAgg outperforms both. Fig. 7.8 (e) shows progress of each. Average Entropy explores the whole world without focusing, Rear Side Voxel exploits early while RewardAgg trades off exploration and exploitation.

The same trend can be observed in Problem Hidden-Con. Fig. 7.9 (c) shows a comparison of QvalAgg with baseline heuristics. The heuristic Rear Side Voxel performs the best, while QvalAgg is able to match the heuristic. Fig. 7.9 (e) shows progress of QvalAgg along with two relevant heuristics - Rear Side Voxel and Average Entropy. Rear Side Voxel takes small steps focusing on exploiting viewpoints along the already observed area. Average Entropy aggressively visits the unexplored area which is mainly free space. QvalAgg initially explores the world but on seeing parts of the lines reverts to exploiting the area around it.

Dataset 2: Distributed blocks

We first examined Problem Hidden-Unc. Fig. 7.8 (b) shows a dataset created by randomly distributing rectangular blocks around the periphery of the map. This dataset is representative of information being distributed around. Fig. 7.8 (d) shows that Rear Side Voxel saturates early, Average Entropy eventually overtaking it while RewardAgg outperforms all. Fig. 7.8 (f) shows that Rear Side Voxel gets stuck exploiting an island of information. Average Entropy takes broader sweeps of the area thus gaining more information about the world. QvalAgg shows a non-trivial behavior exploiting one island before moving to another.

The same trend can be observed in Problem Hidden-Con. Fig. 7.9 (d) shows that the heuristic Average Entropy performs the best, while QvalAgg is able to match the heuristic. Rear Side Voxel saturates early on and performs worse. Fig. 7.9 (f) shows a similar trend as Fig. 7.8 (f).

7.6.3 Case study B: Train on synthetic, test on real

To show the practical impact of our framework, we show a scenario where a policy is trained on synthetic data and tested on a real dataset. Fig. 7.10 (a) shows some sample worlds created

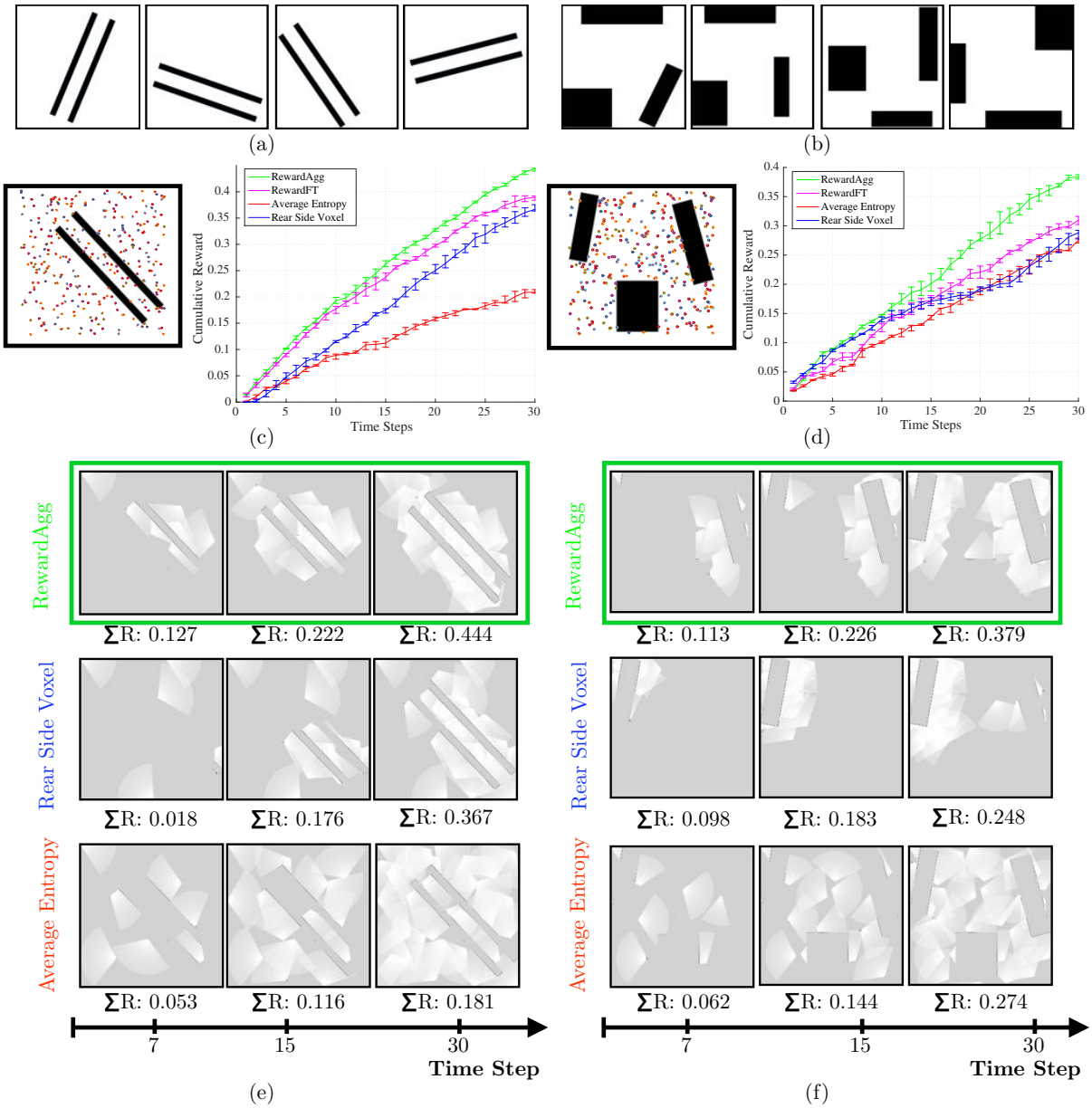


Figure 7.8: Case study of Problem HIDDEN-UNC using REWARDAGG, REWARDFT and baseline heuristics. Two different datasets of 2D exploration are considered - (a) dataset 1 (parallel lines) and (b) dataset 2 (distributed blocks). Problem details are: $T = 30, |\mathcal{A}| = 300$, 100 train and 100 test maps. A sample test instance is shown along with a plot of cumulative reward with time steps for different policies is shown in (c) and (d). The error bars show 95% confidence intervals. (e) and (f) show snapshots of the execution at time steps $7, 15$ and 30.

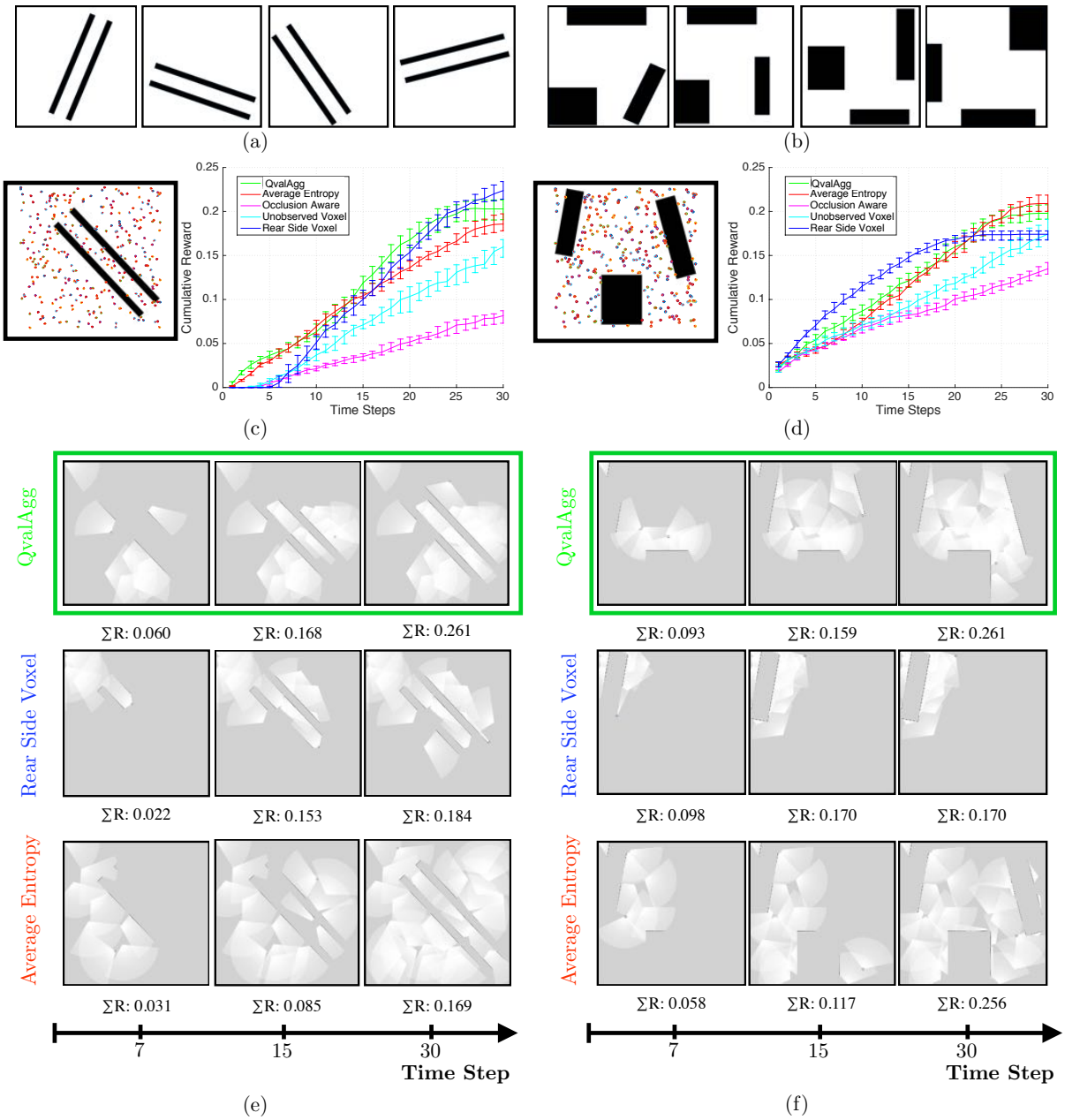


Figure 7.9: Case study of Problem HIDDEN-UNC usingQVALAGG with baseline heuristics on a 2D exploration problem on 2 different datasets - dataset 1 (concentrated information) and dataset 2 (distributed information). The problem details are: $T = 30, B = 2500, |\mathcal{A}| = 300$, 100 train and 100 test maps. A sample test instance is shown along with a plot of cumulative reward with time steps for different policies is shown in (c) and (d) The error bars show 95% confidence intervals Snapshots of execution of QVALAGG, Rear Side Voxel and Average Entropy are shown for (e) dataset 1 and (f) dataset 2. The snapshots show the evidence grid at time steps $7, 15$ and 30.

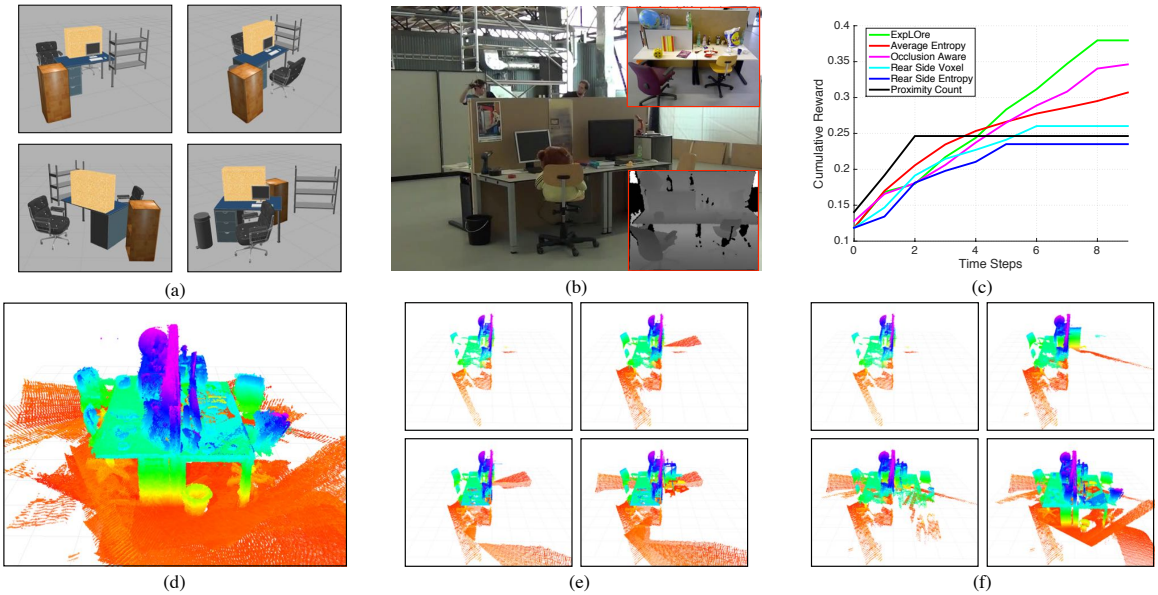


Figure 7.10: Comparison of QvalAgg with baseline heuristics on a 3D exploration problem where training is done on simulated world maps and testing is done on a real dataset of an office workspace. The problem details are: $T = 10$, $B = 12$, $|\mathcal{A}| = 50$. (a) Samples from 100 simulated worlds resembling an office workspace created in Gazebo. (b) Real dataset collected by Sturm et al. [2012] using a RGBD camera. (c) Plot of cumulative reward with time steps for QvalAgg and baseline heuristics on the real dataset. (d) The 3D model of the real office workspace formed by cumulating measurements from all poses. (e) Snapshots of execution of Occlusion Aware heuristic at time steps $1, 3, 5, 9$. (f) Snapshots of execution of QvalAgg heuristic at time steps $1, 3, 5, 9$.

in Gazebo to represent an office desk environment on which QvalAgg is trained. Fig. 7.10 (b) shows a dataset of an office desk collected by TUM Computer Vision Group Sturm et al. [2012]. The dataset is parsed to create a pair of pose and registered point cloud which can then be used to evaluate different algorithms. Fig. 7.10 (c) shows that QvalAgg outperforms all heuristics. Fig. 7.10 (f) shows how QvalAgg learns a desk exploring policy by circumnavigating around the desk. This shows the powerful generalization capabilities of the approach. In contrast, the best heuristic Occlusion Aware gets stuck in a local minima(Fig. 7.10 (e))

7.6.4 Case study C: Policy search vs imitation learning

We compared our approach to a baseline approach of policy search. We picked the problem setting Hidden-Unc, the dataset 'Concentrated Parallel Lines' and the trained policy using RewardAgg. We created a parametrized policy which was linear on the space of the information gain heuristics. The policy, parameterized by $\theta \in \mathbb{R}^6$, assigns at time t to each vertex v, picks the action with the highest score as follows

$$\arg\max_{v_t \in \mathcal{V}} \theta^T \mathcal{IG}\left(v_t\right)$$

We train such a policy using a black-box sample efficient policy search method, Covariance Matrix Adaptation Evolution Strategy (CMAES) [Hansen, 2016]. CMAES is allowed 1000 roll-outs, the same number of calls to oracle as RewardAgg (Note that CMAES actually has access

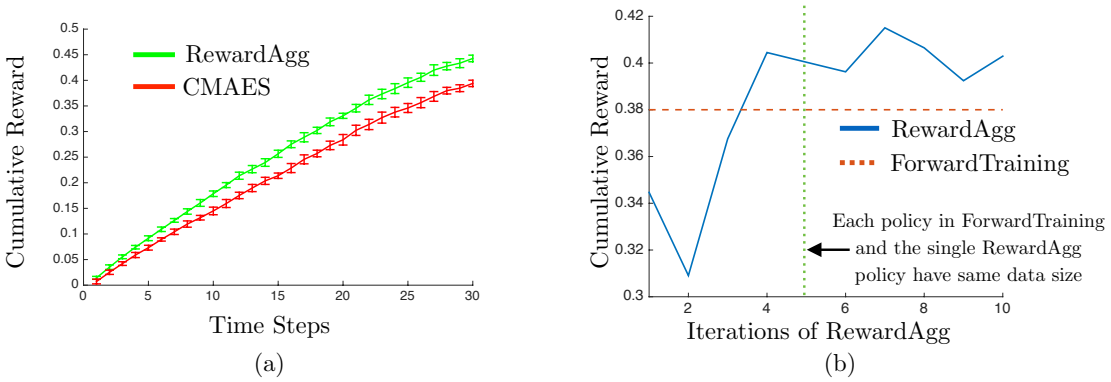


Figure 7.11: (a) Comparison of RewardAgg with CEM policy search. Both algorithms are given access to the same amount of data. The final policy from CEM and the best validation policy of RewardAgg are then executed on a test dataset. RewardAgg outperforms CEM not only overall but pointwise at each timestep. (b) Comparison of RewardAgg with ForwardTraining. Each policy in ForwardTraining is trained with a dataset size of 500. RewardAgg is trained with 100 samples per iteration for 10 iteration. The performance of both policies on test dataset is shown. RewardAgg surpasses ForwardTraining at the 4^{th} iteration and never drops below. At iteration 5 the single policy of RewardAgg has the same dataset size as each policy of the 10 policies of ForwardTraining. However the single policy still outperforms the nonstationary policy.

to more information as they are full rollouts compared to single reward calls in RewardAgg). Fig. 7.11(a) shows comparison between the final policy trained by CMAES and the best policy on validation trained by RewardAgg on a held out test dataset. We see that RewardAgg outperforms CMAES not only on the cumulative reward by also at each time step. This confirms our hypothesis that model free policy improvement is slow to converge on account of sample inefficiency. It should be noted that the CMAES policy outperforms all the baseline heuristics as expected.

7.6.5 Case study D: ForwardTraining vs AggreVaTe

We compared the training framework of ForwardTraining, which trains a different policy for every time-step with AggreVaTe that trains a single policy across all time steps. We wished to examine the following question - 'How much data does a the single AggreVaTe policy need to be competitive with ForwardTraining?'. We picked the problem setting Hidden-Unc and the dataset 'Concentrated Parallel Lines'. We trained ForwardTraining where each policy π_t is given 500 datapoints (hence for episode length $T = 30$, a total of $15,000$ datapoints are used). We trained RewardAgg where each iteration has 100 datapoints, and the the number of iterations is 10. Hence the RewardAgg policy matches the same datasize as ForwardTraining at iteration 5. Fig. 7.11(b) shows a comparison between ForwardTraining and RewardAgg. We see that RewardAgg outperforms ForwardTraining by iteration 4, following which the performance converges and oscillates at values above ForwardTraining. Interestingly, at iteration 5 RewardAgg outperforms ForwardTraining even though each policy in ForwardTraining has access to the same dataset size as RewardAgg. We conjecture that this might be because of the generalization effect across time-steps - ForwardTraining might be over-fitting as it

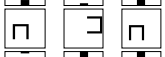
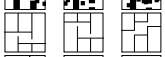
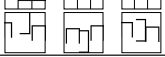
Dataset	Sample Worlds	SAIL	SL	CEM	QL	$h_{\mathbf{EUC}}$	$h_{\mathbf{MAN}}$	A*	MHA*
Alternating Gaps		**0.039**	0.432	0.042	1.000	1.000	1.000	1.000	1.000
Single Bugtrap		0.158	0.214	**0.057**	1.000	0.184	0.192	1.000	0.286
Shifting Gaps		**0.104**	0.464	1.000	1.000	0.506	0.589	1.000	0.804
Forest		**0.036**	0.043	0.048	0.121	0.041	0.043	1.000	0.075
Bugtrap+Forest		**0.147**	0.384	0.182	1.000	0.410	0.337	1.000	0.467
Gaps+Forest		**0.221**	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Mazes		**0.103**	0.238	0.479	0.399	0.185	0.171	1.000	0.279
Multiple Bugtraps		**0.479**	0.480	1.000	0.835	0.648	0.617	1.000	0.876

Figure 7.12: Normalized cost of baselines on different environments (best in bold). The cost corresponds to average expansions on a test set of planning problems normalized between [200, 5000] (max possible: 40000). Planning parameters are - map size: 200×200,$T_{train} = 1100$, $T_{test} = 20000$. Data sizes are: train(200), test(100), validation(70). NONAMEparameters are - $k : 50$, $\beta_0 = 0.7$. NONAME, CEM and QL are run for $N : 15$ iterations. SL uses $m : 600$.

reasons about timesteps individually.

7.7 Experiments on search based planning

In this section, we extensively evaluate our approach on a set of search based planning problems for 2D planning on synthetic problems and more realistic 4D nonholonomic path planning problems encountered by UAVs flying at various speed regimes. We choose a wide variety of world distributions ranging from simple and intuitive environments, chosen to highlight the importance of exploiting environment structure in motion planning, to complex, heterogeneous environments for analyzing scalability and robustness. We also present closed loop results on a UAV flying outdoors at high speeds.

Our implementation is open sourced (`https://goo.gl/YXkQAC`). For details on the problem, baseline approaches, dataset and other learning details, we refer the reader to Choudhury et al. [2017a].

7.7.1 Analysis of results

Fig. 7.12 shows the normalized evaluation cost of all algorithms on various datasets. Snapshots of planning with different heuristics are shown in Fig. 7.13 and Fig. 7.14 (a). Convergence of different learning algorithms are shown in Fig. 7.14 (b). We present a set of key observations to summarize these results.

O 1. SAIL has a consistently competitive performance across all datasets.

Fig. 7.12 shows that SAIL learns a better search policy than any other baseline across all but one environments. It maintains performance from homogeneous to heterogeneous environments.

O 2. SAIL has faster convergence than all learning baselines.

Fig. 7.14 (b) shows that on the 'Forest' dataset, SAIL converges by 6^{th} iteration, while CEM

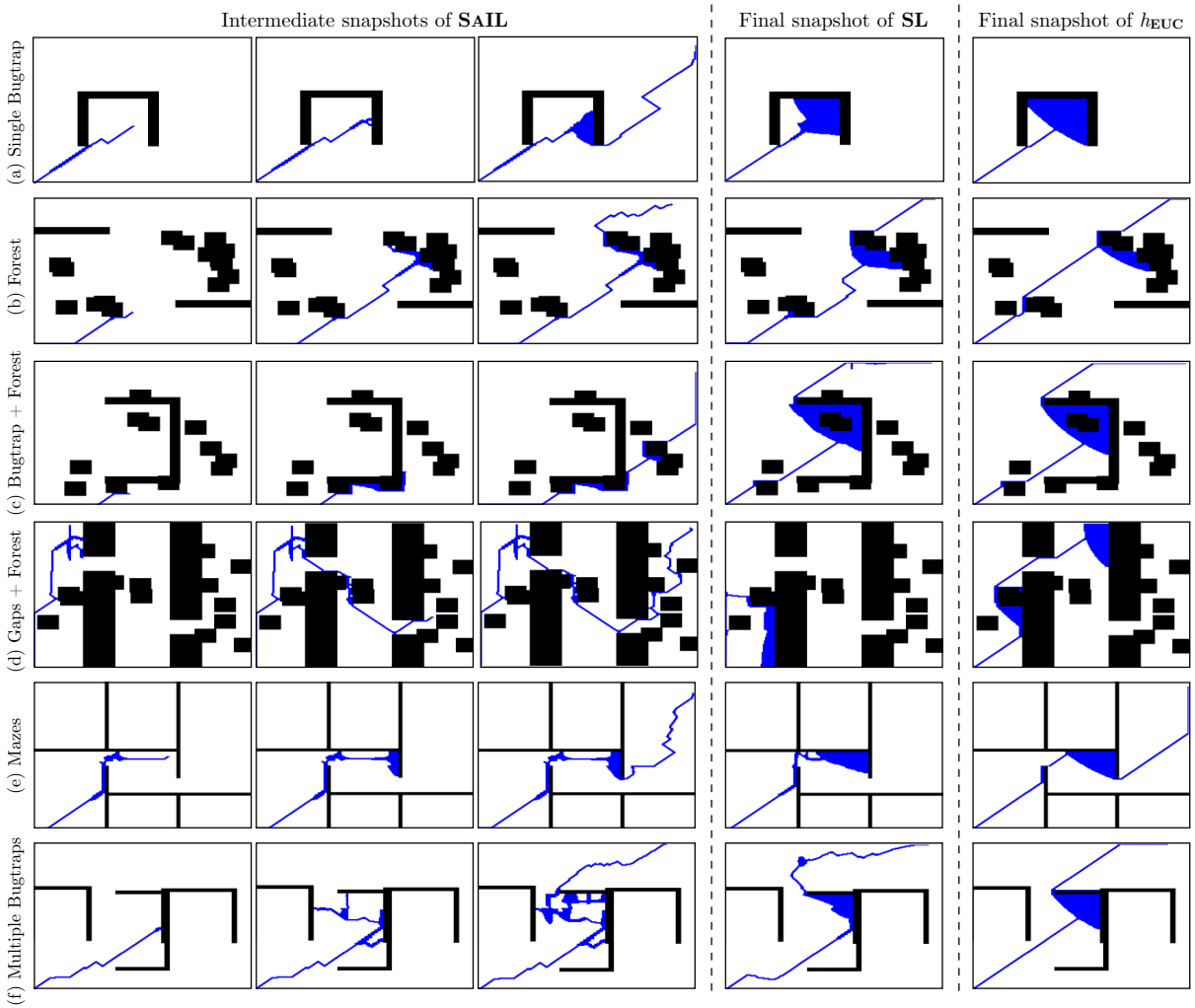


Figure 7.13: Evolution of search frontier (expanded(blue), invalid(black), unexpanded(white)) of SAIL compared with final snapshot of supervised learning (SL) and h_{EUC} across all environments. SAIL expands far less states.

takes 12 and QL does not converge. SAIL also converges quickly (by the 8^{th} iteration) across datasets.

O 3. SAIL is able to detect and escape local minima.

A classic case in motion planning is the bugtrap (Fig. 7.1 (b)) which traps a greedy search in a local minimum. Fig. 7.13 (a) and Fig. 7.13 (f) shows that when trained on such distributions, SAIL is able to detect these artifacts and smartly escape them by exploring in different directions.

O 4. SAIL is able to exploit the relative configuration of obstacles and environment structure.

In a maze world with rectilinear hallways (Fig. 7.13 (e)), SAIL learns to quickly find a wall and then concentrate the search along the axes. In Fig. 7.13 (d), SAIL focuses only on regions where there is a high probability of a gap and skids along obstacles otherwise.

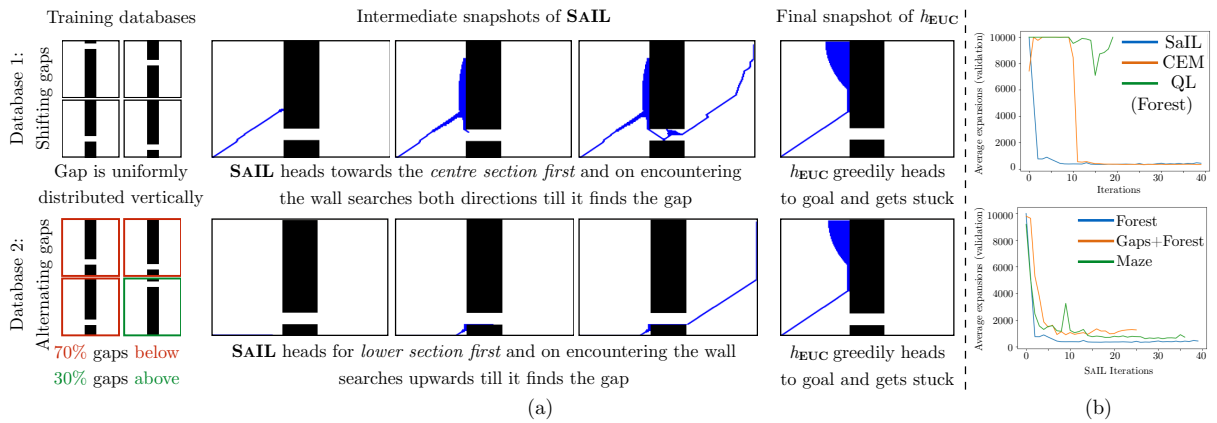


Figure 7.14: (a) SAIL learns to adapt to different environment distributions by directing search to areas where it expects to find gaps. Note SAIL does not have information about the entire environment, only the explored part. (b) On the 'Forest' dataset, SAIL converges faster that CEM and QL to a good policy. SAIL also converges consistently to a good policy across environments 'Gaps', 'Gaps+Forest', 'Maze.'

7.7.2 Case study A: Adaptive behaviour of SaIL

We take a closer look at the behaviour of SAIL in response to a change in the distribution of worlds that it is being trained on $P(\phi)$. Consider the scenario illustrated in Fig. 7.14 (a). We create two datasets. Both datasets have a wall in the middle of the environment, with a gap in the wall. For dataset 1, the gap can occur uniformly randomly along the y-axis. For dataset 2, the gap either occurs with 70% probability at the bottom and 30% probability at the top.

For dataset 1, SAIL learns to approach the centre of the environment first and then search along the wall till it finds a gap. This is in response to the fact that the gap can occur anywhere and hence this is a cost efficient strategy. Contrast this to a greedy search that get stuck expanding states near the top of the wall.

For dataset 2, SAIL learns to approach the bottom of the environment first and then search along the wall. This is in response to the gaps occurring at the bottom of the wall. The greedy search is non responsive to the change in distribution and gets stuck expanding states near the top again.

7.7.3 Case study B: Helicopter path planning

An important application of heuristic learning is to speed up high dimension search. An application of particular relevance to us is an autonomous helicopter [Choudhury et al., 2014]. A class of environment in which the helicopter has to plan in is a canyon like environment. Since the system moves at a speed of $30m/s$, it has to produce a plan in real-time (within $200ms$) otherwise it risks reaching states from which collision is inevitable.

We use SAIL to learn a heuristic that guides search in such environments. We collect a dataset by generating canyons using a parametric distribution as showing in Fig. 7.15 (a). A lattice is created. As a baseline, we run A* with Dubins distance as the heuristic on this problem. As shown in Fig. 7.15 (b), this ends up expanding a large number of vertices (2531). This is

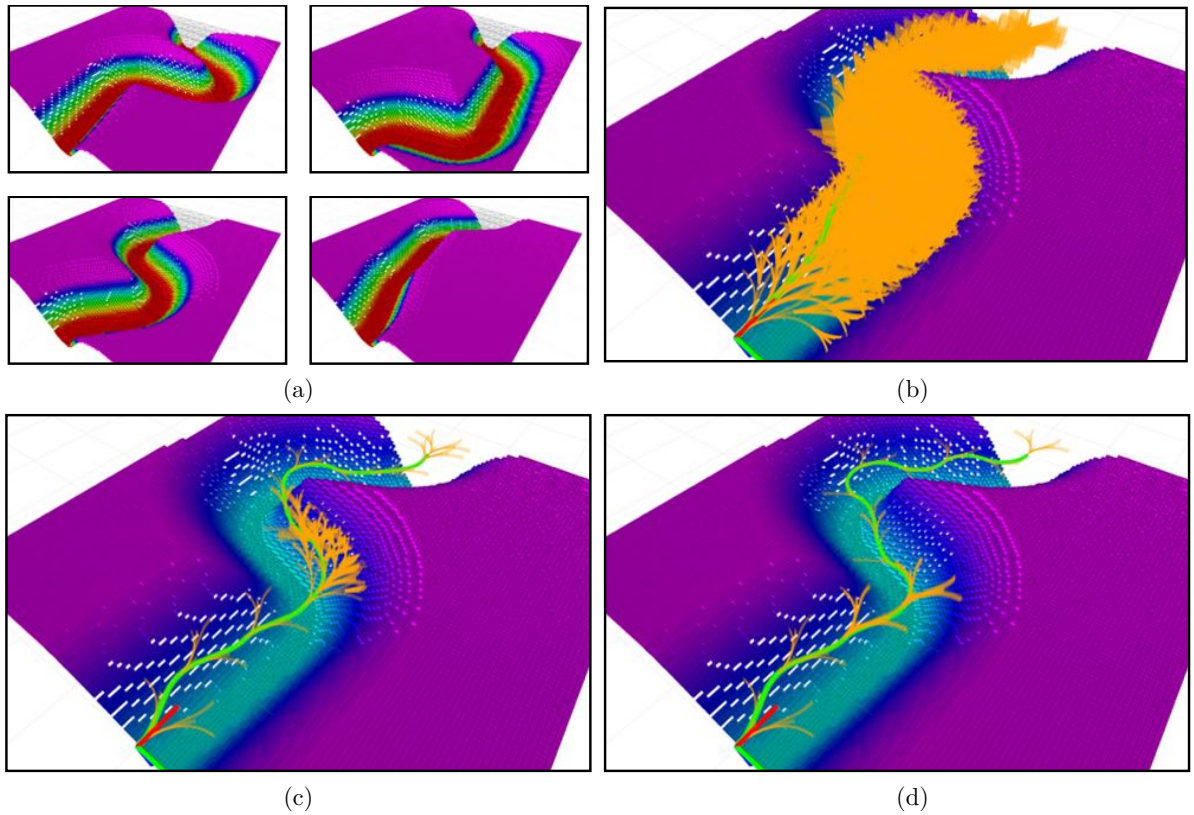


Figure 7.15: Experiments on path planning for an autonomous helicopter in a canyon environment. The environment is motivated from planning challenges as described in Choudhury et al. [2014]. (a) Dataset of canyon-like environments generated by a parametric distribution. (b) The search tree from A* with Dubins distance heuristic on a test environment. The start point is shown by the axes. The expanded edges are shown in yellow. The planned path is shown in green. A* expands 2531 vertices and takes 7000ms. (c) The search tree for greedy search with Dubins distance heuristic. It expands 142 vertices and takes 500ms. Note that most of the wasted expansions are where the tree tries to search through the canyon wall (d) SAIL expands only 18 vertices and takes 100ms. It hugs the canyon wall till it reaches the goal.

because the Dubins distance is not the optimal cost to do. The under-estimation of this heuristic results in a large number of vertices being expanded and hence a long planning time (7000ms).

We also run a greedy search using the Dubins distance as a heuristic. We see that for these kind of environments, greedy search performs pretty good - the number of vertices expanded is 142 and planning time is 500ms. However, the greedy search expends search effort trying to search for a tunnel through the canyon.

SAIL has much better performance than either of these baselines. It is able to learn a heuristic that expands only 18 vertices with a search time of 100ms. The features used by SAIL are minimalistic and are described in Choudhury et al. [2017a]. Among those features are the Dubins distance to the goal and the direction vector to the nearest obstacle. By examining the search tree produced by SAIL, we observe that it learns a trade-off between following the Dubins distance heuristic and not expanding states that are pointing into the canyon wall (as such states would not result in a feasible path eventually).

7.7.4 Case study C: Quadrotor path planning

We also applied this approach to a real quadrotor which has to navigate in an environment at high speed $5m/s$ while avoiding no fly zones. No fly zones can result from areas that a UAV cannot fly over because of risks to property or from other vehicles in the area. These no fly zones can be arbitrary in complexity thus creating artifacts such as a maze as shown in Fig 7.16.

We create a dataset of such mazes by means of a parametric distribution as shown in Fig 7.16 (a). We give a time budget of $1000ms$ for planners to solve the problem. A* with Dubins heuristic is unable to solve the problem in the time limit as shown in Fig 7.16 (b). This is because the Dubins distance vastly under-estimates the distance to the goal in this environment. A* expands 1910 states before being terminated.

Greedy search with Dubins heuristic is able to find a path after 661 expansions within the time budget (in $400ms$). The remaining time is spent relaxing the path found. The greedy behaviour is beneficial in this environment because it results in a wall following like behaviour. However, the algorithm wastes search effort expanding states perpendicular to the wall which would lead to inevitable collision.

SAIL outperforms both algorithms by finding a path in 180 expansions (in $120ms$). The remaining time is spent relaxing the path. As can be seen for the search graph, it focuses on expanding paths perpendicular to the wall. It learns to not expand vertices that point into the wall since the oracle shows the cost to go of such nodes to be ∞.

We also evaluated SAIL on board a DJI M100 quadrotor equipped with a TX2 computer. We created a synthetic maze with no fly zones and commanded the robot to fly through it (Fig 7.16 (e-f)). SAIL is able to find a path expanding a sparse number of vertices. As the robot follows the path, the algorithm is able to consistently replan and find a path consistently without expanding too many states (Fig 7.16 (g)).

7.8 Discussion and future work

We presented a novel data-driven imitation learning framework to learning planning policies. Our approach trains a policy to imitate a clairvoyant oracle that has full information about the world and can compute optimal planning decisions. We examined two problem domains - informative path planning and search based planning. We evaluated our approach in both these domains and showed that the learnt policy can outperform state-of-the-art approaches. We now discuss a set of relevant questions and directions for future work.

Q 1 (Applicability of Framework)**.** When does this framework lead to good policies? What are some failure cases?

MDP framework provides an elegant way of posing problems where the complete state of the problem space is known. The value of an action for a given state in an MDP is given by equation 7.13.

$$Q_t^{\pi}(s,a) = R(s,a) + \mathbb{E}_{s' \sim P(s'|s,a)} \left[V_{t-1}^{\pi}(s') \right] \tag{7.13}$$

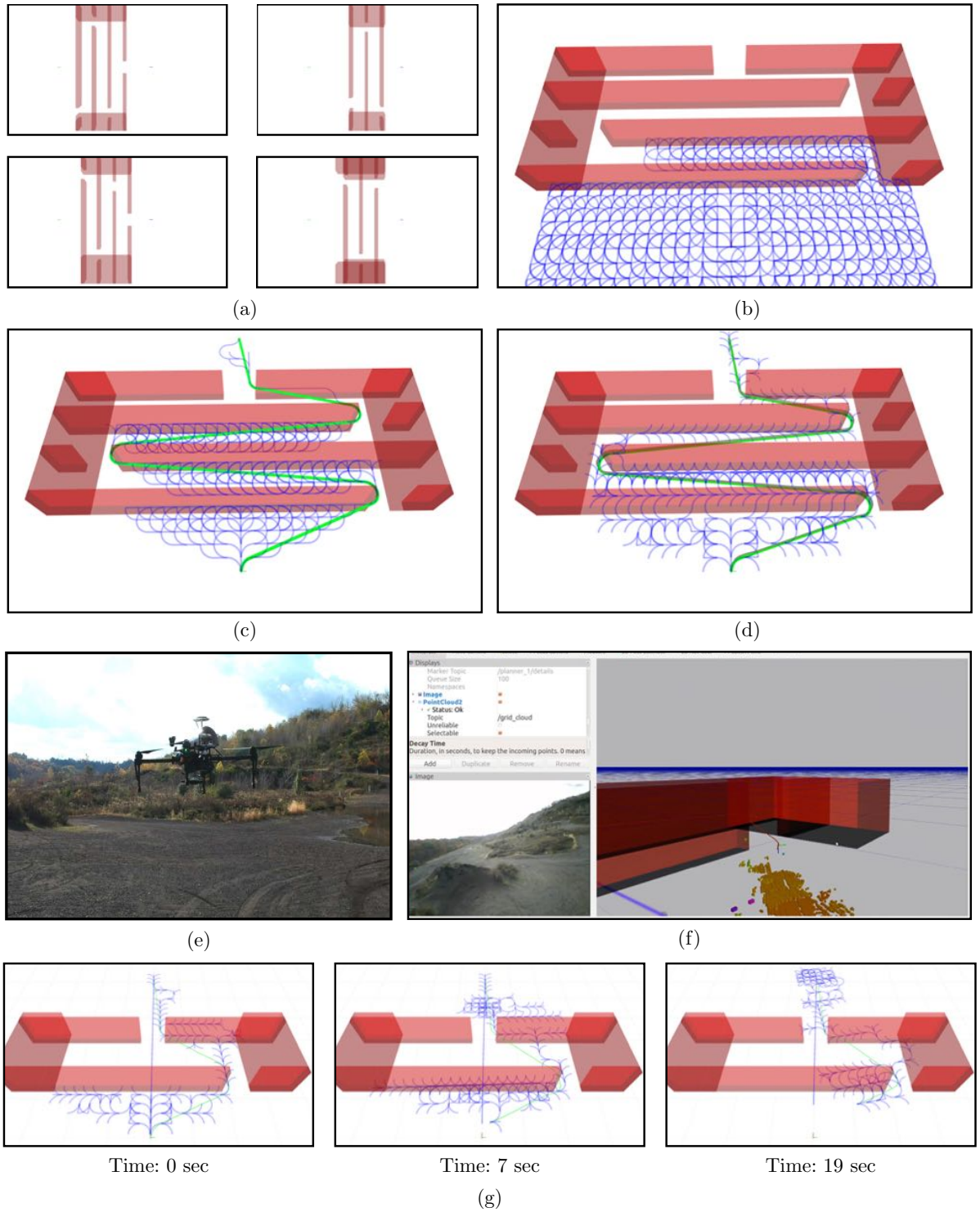
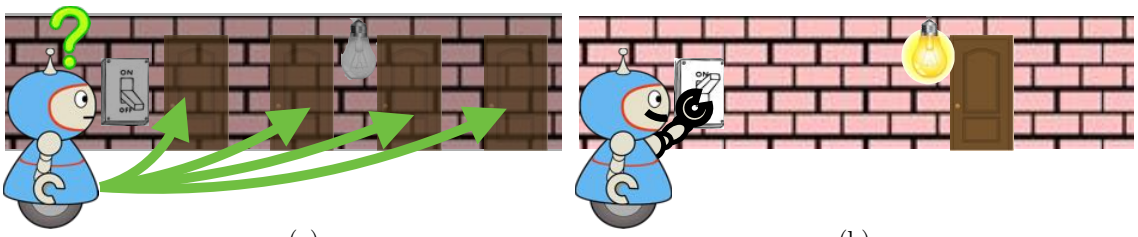


Figure 7.16: Experiments on path planning for a real quadrotor flying at high speed $5m/s$ while avoiding no fly zones that represent a maze like scenario. (a) A dataset of mazes created from a parametric distribution (b) The search graph of A* on the environment. It expands 1910 states in the $1000ms$ time budget without finding a path (c) The greedy search with Dubins distance expands 661 vertices and takes $400ms$. The remaining time is used to relax the path shown in green. (d) SAIL outperforms both and finds a path by expanding only 180 vertices in $120ms$. (e) The DJI M100 used for our experiments (f) An experiment where SAIL is running onboard the robot. A set of no fly zones is created and the robot has to fly through it. The robot view and onboard imagery is shown (g) A time lapse of the search tree as the robot replans while performing the mission. We can see that the search tree remains sparse through out and SAIL is always able to find a path.



(a) (b)

Figure 7.17: The robot in a dark room problem. The robot is uncertain about the location of a door and the only way to collapse that uncertainty is to pull a light switch. (a) A clairvoyant oracle is not incentivized to flip the switch and hence the robot does not learn to collapse uncertainty (b) The optimal POMDP policy would be to flip the switch and then head for the door.

$$V_t^{\pi}(s) = \sum_{i=t}^{T} \mathbb{E}_{s_t \sim P(s_i|\pi,i,s)} \left[R\left(s_i, \pi(s_i)\right) \right] \tag{7.14}$$

The optimal MDP policy maximizes the expected cumulative reward, i.e $\pi^*(s_t) \in \arg\max_{\pi \in \Pi} V_t^{\pi}(s_t)$.

However, there are 2 major challenges that POMDP solvers face-

- Computing the expectation over the state space. Since the state space of most of the problems worth solving is large, computing an expectation over such state space needs a large number, making it expensive to evaluate online.
- Keep track of evolving uncertainty about the state space over the planning horizon.

Our approach solves the first challenge through data driven techniques - the MDP solvers are used over sampled MDP problems to train a policy on the expected distribution of problems. The hallucinating oracle is similar in nature to a QMDP algorithm [Littman et al., 1995], an effective approximate solution to POMDPs, which takes the best action on the current posterior. However, while QMDP requires maintaining an explicit posterior, our framework does not. QMDP has been shown to be very successful where explicit information gathering behaviour is not required [Javdani et al., 2015, Koval et al., 2014] - the belief collapses irrespective of the action. Hence this optimization assumes a fixed belief and does not account for evolving belief over time, (which is challenge 2 for POMDP's). This implies there is no motivation for the MDP solver and hence the learnt policy to change the belief.

These kind of methods work quite well in POMDP problems where the required changes in belief can be attained by actions that are rewarding as well. This is very apt in the problem we address - as the set of actions are constrained to candidate nodes in the open list, no single action is very informative. It suffices to expand the best node under the current belief and continue to update the belief as the open list evolves. And there exists no action that is not rewarding while reducing the uncertainty. We note that this is not true for all learning in planning paradigms. For example, when learning to collision check [Choudhury et al., 2017b], a policy that actively reduces uncertainty about the world is effective.

To illustrate the failure case, we present a simple scenario as shown in Fig 7.17. We have a 'trapped robot' whose task is to escape from a room, i.e. it gets a reward for escaping and penalization for staying in the room. The room is dark, i.e. the robot cannot observe the location of the door. It can perform actions such as moving in the room. It can also perform an action of flicking on the light switch. On performing such an action, it receives an observation containing the location of the door. An optimal POMDP policy would always choose this action, collapse uncertainty about the door location and subsequently head straight for the door. However, imitation of clairvoyant oracles do not provide such behaviours. The oracle, at training, always guides the robot towards the door to maximize reward and is not incentivized to flip the light switch. The policy learns a blind search pattern which takes a long time to find the door.

For such POMDP problems, one way forward would be to incentivize the oracle at train time to reduce the uncertainty as suggested by the POMDP-lite approach [Chen et al., 2016a]. While POMDP-lite quantifies uncertainty reduction as L1-norm of the belief change, this can be hard to compute for the space of world maps. Using approximations to this belief change would be an interesting direction of future work.

Q 2 (Anytime Search)**.** How can we incorporate solution cost in addition to search effort in this framework?

While our framework ignores the cost of a solution, we note that finding feasible solutions quickly is the core motivation of a number of high dimensional planning problems which have historically resorted to sampling based approaches [Kuffner and LaValle, 2000]. Hence, one can apply our framework to such problems to produce potentially faster solutions. We also note that when planning on locally connected lattices for geometric planning problems, minimizing the number of expansions generally leads to near-optimal solutions (unit cost for each valid edge).

However, if we really cared about near optimal solutions, the framework of Multi-heuristic A* (MHA*) [Aine et al., 2016] can be easily adopted. In such a framework, any heuristic function [Narayanan et al., 2015] can be used in tandem with an anchored search which uses an inflated admissible heuristic. Hence we can simply replace our `Search` function with MHA*.

The bi-objective criteria of solution cost and search effort is best reasoned about in the paradigm of *anytime planning*. In this paradigm, an algorithm traces out the *pareto-frontier* [Choudhury et al., 2016b] - finds a feasible solution quickly and iteratively improves it. In this paradigm, SAIL trains a heuristic that displays a behaviour we would expect in the first iteration. A direction of future work would be to learn *anytime heuristics* that minimize search effort initially to and solution cost eventually.

Q 3 (Learning to Sample)**.** Can we generalize this framework to sampling based planners?

The SAIL framework defines `Search` in a very general way - the underlying implicit graph can also be a tree and the expansion operation can be a local steering operation akin to the framework of EST [Hsu et al., 1999b]. The oracle design is an open question - a plausible oracle is growing a backward tree from the goal and using a k-NN value function approximator. Another paradigm to consider is when the `Expand` operation is a call to a *sampler*. For example, the framework in Randomized A* (RA*) [Diankov and Kuffner, 2007] proceeds by selecting a node of the search tree using some criteria and sampling around it.

Recently, Ichter et al. [2017] proposed a framework for learning sampling distributions from optimal paths during training by using a conditional variational auto-encoder (CVAE). However, in this framework sampling and planning are decoupled, i.e. the sampling policy learns a good stationary distribution from which samples are generated and provided to the planner. Hence the planner does not adapt during the planning cycle. Such a stationary distribution can be very hard to learn as directly predicting the optimal path requires conditioning on a lot of information about the environment.

SAIL can be extended to learn sampling policies that address this problem. The CVAE can condition on the state of the search (similar to the feature vector used by SAIL). The labels can be obtained by a backward tree from the goal grown during training. The iterative learning process of SAIL will ensure that the CVAE is trained on the distribution of search state actually encountered rather than simply using the optimal path.

8

Bayesian Active Edge Evaluation

Previously, in Chapter 7, we formulated learning planning policies as sequential decision making. We wanted to learn policies that inferred the utility of a decision based on the history of decisions and actions. We showed how such policies can be effectively trained via imitation learning of clairvoyant oracles. The approach was tractable as it did not require explicitly modelling uncertainty about the world. However, the policies we learnt were passive - they did not actively choose decisions to reduce uncertainty about the world, which in turn would aid them in imitating the oracles.

In this Chapter, we examine learning policies that *actively infer the underlying structure* of the valid configuration space during planning in order to find solutions with minimal effort. To make the problem tractable, we investigate planning on explicit graphs and model uncertainty as function of edges on the graph. We motivate the problem of actively evaluating edges in Section 8.1. We provide some background in Section 8.2. We then describe the formal problem in Section 8.3 and show how its equivalent to the Bayesian active learning paradigm of decision region determination. We describe an active learning algorithm, DIRECT, that solves the problem in Section 8.4. We also analyze a special case where edges are indpendent random variables and describe an algorithm, BISECT, to solve it in Section 8.5. We present our approach on combining these two algorithms to solve the edge evaluation problem in Section 8.6. We extensively evaluate our approach on a number of datasets in Section 8.7. We summarize and discuss potential future directions in Section 8.8.

8.1 Introduction

A widely-used approach for solving robot motion-planning problems is the construction of graphs, where vertices represent robot configurations and edges represent potentially valid movements of the robot between these configurations. The main computational bottleneck is collision checking which is manifested as expensive edge evaluations. For example, in robot arm plan-

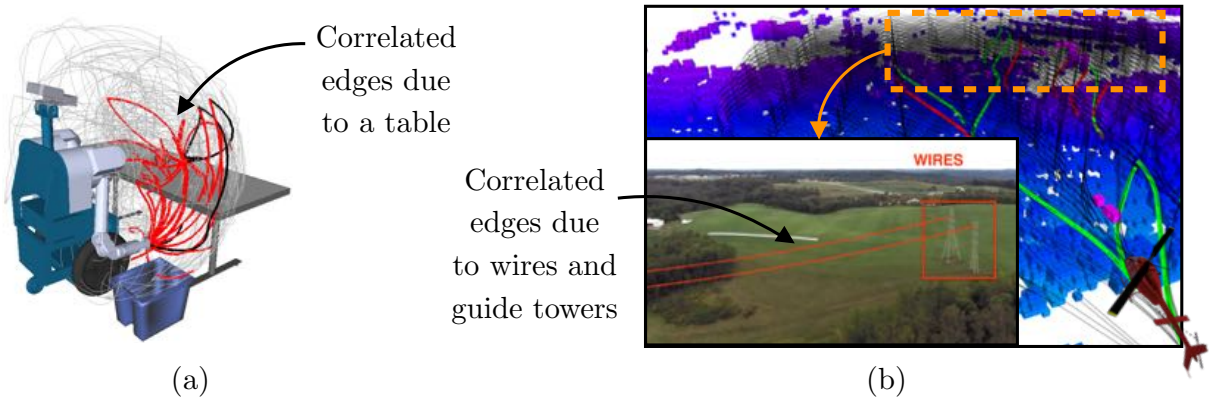


Figure 8.1: Real world planning problems where edges are correlated. In such cases, we can infer the structure of the world from outcomes of edge evaluations. (a) The presence of a table in robotic arm planning correlates neighbouring edges (courtesy Dellin and Srinivasa [2016]). (b) The presence of wires and guide-towers in helicopter planning correlates corresponding edges.

ning [Dellin et al., 2016] (Fig. 8.1(a)), evaluation requires expensive geometric intersection computations. In autonomous helicopter planning [Choudhury et al., 2014] (Fig. 8.1(b)), evaluation requires expensive reachability volume verification of the closed loop system. State-of-the-art planning algorithms [Dellin and Srinivasa, 2016] deal with expensive evaluation by resorting to *laziness* - they first compute a set of unevaluated paths quickly, and then evaluate them sequentially until a valid path is found.

However, these methods do not reason about the utility of an edge. Edges are not alike in value - some are *important*, others are *informative*. Important edges have a lot of good paths flowing through them. Informative edges, on being evaluated, affect the likelihood of other neighboring edges being valid. Hence, we wish to characterize and leverage this utility to do more than passive laziness. This motivates us to ask the question

> Can we enable planners to actively probe edges in the graph in order to quickly find feasible paths?

8.1.1 Motivating examples

We explore two problem settings - planning in environments with varying implicit structure and search on generalized binomial graphs. We describe scenarios where these settings arise naturally and make the case for active edge evaluation approaches in both.

Planning in structured environments

Consider the problem of planning for a robot arm doing a pick and place task on a cluttered table top as shown in Fig. 8.1 (a). The presence of a table in the robot arm workspace implicitly correlates edges in front of the robot. Similarly consider the problem of an autonomous helicopter flying over terrain with hills and power-lines as shown in Fig. 8.1 (b). The presence of power-lines implicitly correlates a horizontal strip of edges near the ground.

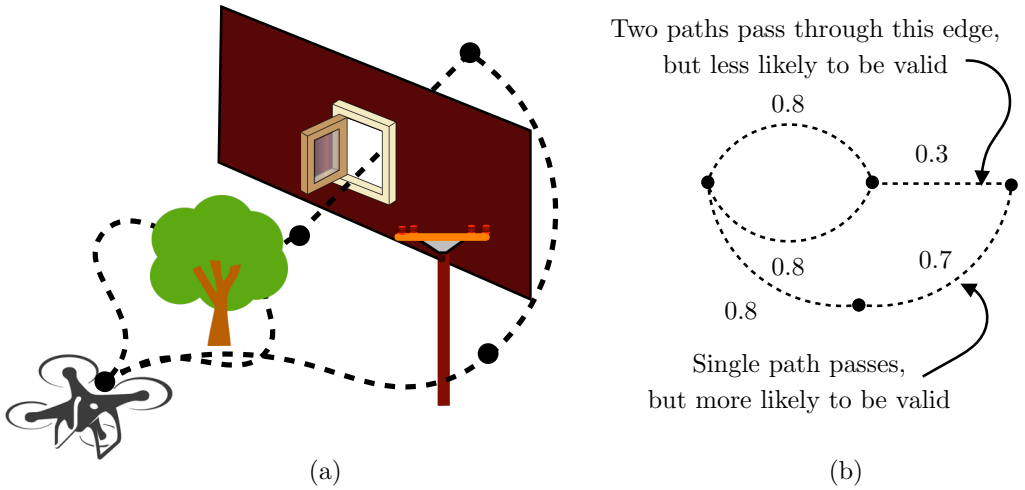


Figure 8.2: Planning on Generalized Binomial Graphs (GBG) where each edge has an independent probability of being valid. (a) A scenario where a UAV has to fly either through a window or around the wall. We consider a simple graph with 5 edges and 3 likely paths. To determine if an edge is valid or not, we have to do expensive collision checking. However we can guess the likelihood easily with a simpler model (b) The likelihood of each of these edges. The edge evaluation policy must reason about these likelihoods as well as how many paths flow through the edge.

The underlying observation for these problems is that *edges in a graph are implicitly correlated*. Evaluating such edges provides valuable information about the feasibility likelihood of other edges which in turn can be used to infer the feasibility likelihood of a path.

Lets take a closer look at the arm planning problem in Fig. 8.1 (a). If two edges passing through the front of the robot is invalid, the planner should infer all edges between them are likely to be invalid. It should then evaluate paths on either flank and leave the central edges untouched. We wish to compute such a policy that judiciously chooses edges to evaluate by reasoning about likely worlds in which the robot operates.

Search on generalized binomial graphs

There are situations where we only have independent likelihoods of an edge being valid. This can be due to insufficient data for capturing correlations between edges. This can also arise from assumptions about the world due to imperfect sensing. A third possibility is that we can use a simpler and cheaper model of the world to get these probabilities, and evaluating the true status of edges requires expensive computations. Graphs where each edge has an independent likelihood of being valid is known as a Generalized Binominal Graph (GBG) [Frieze and Karoński, 2015].

Consider such a scenario where an UAV is flying in an urban environment as shown in Fig. 8.2(a). Evaluating each edge requires precisely checking the geometry of the UAV, inflated by execution uncertainty with a high fidelity model of the world (a point cloud). On the other hand, we can easily obtain likelihoods of each edge being valid by using a coarse model of the world and assuming a point robot. Hence we have the GBG shown in Fig. 8.2(b).

Which edge should we evaluate? One candidate edge has two paths passing through it. Hence

if its valid, it increases the chances of finding at least one valid path. However, since that edge passes through a window, it is likely to be invalid and may result in a wasted evaluation. On the other hand, we have an edge that belongs to only one path but is likely to be valid. We want a principled way to reason about this trade-off. In other words, we want to actively decide which edge to evaluate to increase the likelihood of a path being feasible.

8.1.2 Key ideas

Our key insight is that this problem is equivalent to the Bayesian active learning problem of *decision region determination (DRD)* [Chen et al., 2015a, Javdani et al., 2014a] - given a set of tests (edges), hypotheses (worlds), and regions (potential paths), the objective is to select a sequence of tests that drive uncertainty into a single decision region. The DRD problem has one important distinction from the general active learning problem [Dasgupta, 2004] - we only need to know enough about the world to ascertain if a path is feasible.

To solve the DRD problem in our context, we need to address two issues:

(a) Enumeration of all possible worlds.

(b) Solving the DRD problem in general is NP-hard [Javdani et al., 2014a].

Fortunately, Chen et al. [2015a] provide an algorithm, DIRECT, to address (b) by maximizing an objective function that satisfies *adaptive submodularity* [Golovin and Krause, 2011] - a natural diminishing returns property that endows greedy policies with near-optimality guarantees.

However, DIRECT requires (a) to be solved, i.e. requires an exhaustive training database of worlds. Since DIRECT operates on a realizability assumption, it can easily terminate without finding a solution when the test world is not in its training database. Explicitly enumerating all possible worlds is impractical even as an offline operation - a graph with E edges can induce $\mathcal{O}\left(2^E\right)$ possible worlds.[1]

We address (a) by examining the DRD problem when edges are *independent*. We proposed an efficient near-optimal algorithm BISECT that has linear complexity $\mathcal{O}(E)$. BISECT reasons about the exhaustive set of worlds without ever explicitly enumerating them by leveraging the independence assumption.

Our key idea is to combine the two approaches. We sample a finite database of worlds and apply DIRECT offline on this database to compute a decision tree of edges to evaluate. At test time we execute the tree. When we reach a leaf node, we have either solved the problem or we have narrowed the problem down to a set of 'tail worlds' outside of DIRECT's domain, i.e. low probability worlds that do not appear in the sampled database. We then run BISECT, which implicitly reasons about this set of 'tail worlds', and accept the performance loss due to the independence assumption.

8.1.3 Contributions

We make the following contributions:

[1] A typical graph, $|E| : 10000$, will need 2^{10000} bits of storage!

1. We show an equivalence between the optimal edge evaluation problem and the decision region determination problem (Section 8.3).

2. We develop BiSECt, a near-optimal algorithm for the special case of Bernoulli tests, which selects tests in $\mathcal{O}(E)$ instead of $\mathcal{O}\left(2^E\right)$ (Section 8.5).

3. We propose a framework to combine two DRD algorithms, DiRECt and BiSECt, that near-optimally solves the decision region problem, overcomes issues pertaining to finite databases and can be executed efficiently online (Section 8.6).

4. We demonstrate the efficacy of our approach on a spectrum of planning problems for mobile robots, manipulators, autonomous full-scale helicopters (Section 8.7).

8.2 Background

8.2.1 Lazy edge evaluation

The computational bottleneck in motion planning varies with problem domain and that has led to a plethora of planning techniques [LaValle, 2006]. When vertex expansions are a bottleneck, A* [Hart et al., 1968] is optimally efficient while techniques such as partial expansions [Yoshizumi et al., 2000] address graph searches with large branching factors. However, we examine the problem class that is of particular importance in robotics - expensive edge evaluation. This is primarily because evaluation is performed by querying an underlying representation of the world that is built online and requires expensive geometric intersection computation.

The problem class we examine, that of expensive edge evaluation, has inspired a variety of 'lazy' approaches. The LazyPRM algorithm [Bohlin and Kavraki, 2000] only evaluates edges on the shortest path while FuzzyPRM [Nielsen and Kavraki, 2000] evaluates paths that minimize probability of collision. The Lazy Weighted A* (LWA*) algorithm [Cohen et al., 2015] delays edge evaluation in A* search and is reflected in similar techniques for randomized search [Choudhury et al., 2016a, Gammell et al., 2015, Hauser, 2015]. An approach most similar in style to ours is the LazyShortestPath (LazySP) framework [Dellin and Srinivasa, 2016] which examines the problem of which edges to evaluate on the shortest path. Instead of the finding the shortest path, our framework aims to efficiently identify a feasible path in a library of 'good' paths. The Anytime Edge Evaluation (AEE*) framework [Narayanan and Likhachev, 2017] also deals with a similar problem however it makes an independent edge assumption. Finally, there is a lot of work on modelling belief over configuration spaces [Burns and Brock, 2005a, Choudhury et al., 2016b, Huh and Lee, 2016, Pan et al., 2012]. Using such models in DRD would be interesting future work.

8.2.2 Bayesian active learning

We draw a novel connection between motion planning and optimal test selection which has a wide-spread application in medical diagnosis [Kononenko, 2001] and experiment design [Chaloner and Verdinelli, 1995]. Optimizing the ideal metric, decision theoretic value of information [Howard, 1966], is known to be $\mathrm{NP}^{\mathrm{PP}}$ complete [Krause and Guestrin, 2009]. For

hypothesis identification (known as the Optimal Decision Tree (ODT) problem), Generalized Binary Search (GBS) [Dasgupta, 2004] provides a near-optimal policy. For disjoint region identification (known as the Equivalence Class Determination (ECD) problem), EC^2 [Golovin et al., 2010] provides a near-optimal policy. When regions overlap (known as the Decision Region Determination (DRD) problem), HEC [Javdani et al., 2014a] provides a near-optimal policy. The DIRECT algorithm [Chen et al., 2015a], a computationally more efficient alternative to HEC, forms the basis of our approach.

8.3 Problem formulation

We introduce the problem for identifying feasible paths. We show equivalence with Bayesian active learning. We also examine a special case of independent edges which will be useful in the approach.

8.3.1 The Feasible Path Identification problem

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an explicit graph that consists of a set of vertices $\mathcal{V}$ and edges $\mathcal{E}$. Given a pair of start and goal vertices, $(v_s, v_g) \in \mathcal{V}$, a search algorithm computes a path $\xi \subseteq \mathcal{E}$ - a connected sequence of valid edges.

The search is performed on an underlying world ϕ which corresponds to a particular configuration of obstacles. To ascertain the validity of an edge e, the algorithm queries the world ϕ which returns a binary outcome - valid (1) or invalid (0). We address applications where edge evaluation is expensive, i.e., the computational cost $c(e)$ of evaluating an edge is significantly higher than regular search operations [Dellin and Srinivasa, 2016].

Note that the search algorithm is free to evaluate any edge $e \in \mathcal{E}$. This is made possible since we have an explicit graph as opposed to the implicit graph considered in Section 7.2.2 where edges are created dynamically during the search process. Using an explicit graph allows us to easily leverage a prior on the distribution over worlds $P(\phi)$. We represent this prior as a set of outcome vectors $\Phi = \{\phi_1, \ldots, \phi_n\}$ sampled from $P(\phi)$, where $\phi_i \in \{0, 1\}^{\mathcal{E}}$. This set Φ can easily be used to compute the likelihood of an edge e being valid as well as the posterior likelihood conditioned on the outcome of other edges.

We make a simplification to the problem - from that of search to that of identification. Instead of searching $\mathcal{G}$ online for a path, we frame the problem as identifying a valid path from a library of 'good' candidate paths $\Xi = (\xi_1, \xi_2, \ldots, \xi_m)$. The candidate set of paths Ξ is constructed offline, while being cognizant of $P(\phi)$, and can be verified to ensure that all paths have acceptable solution quality when valid.

We are now ready to define the *Feasible Path Identification* problem. We wish to design an adaptive edge selector $\texttt{Select}(\phi)$ which is a decision tree that operates on a world ϕ, selects an edge for evaluation and branches on its outcome. By abuse of notation, we define the total cost of edge evaluation for such a selector on a world ϕ to be $c(\texttt{Select}(\phi))$.

Problem 10 (Feasible Path Identification (FEAS-PATH))**.** Given a distribution of worlds, $P(\phi)$,

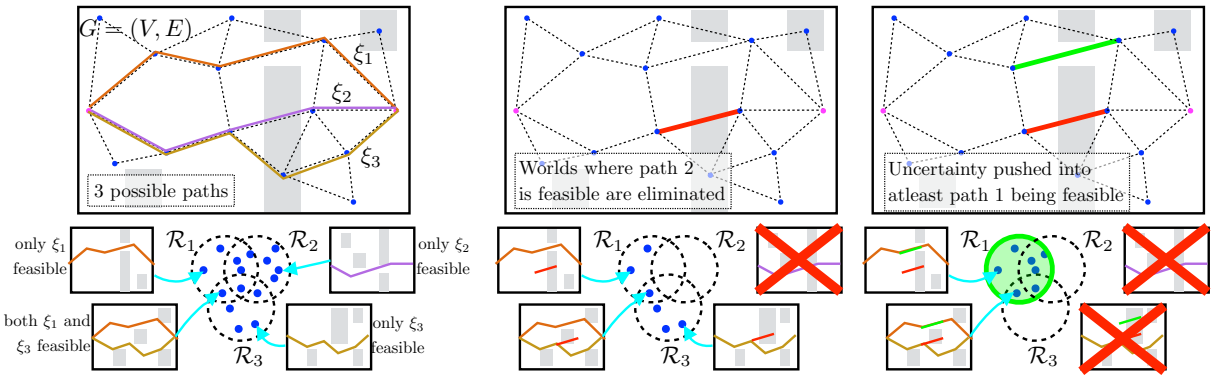


Figure 8.3: Equivalence between the feasible path identification problem and the decision region determination problem. A plausible world is equivalent to hypothesis (as shown by the blue dots in the lower row). A path ξ_i is equivalent to a region $\mathcal{R}_i$ over valid hypotheses where the path is feasible. A collision check is equivalent to a test whose outcome is valid (green) or invalid (red). Tests eliminate hypotheses and the algorithm terminates when uncertainty is pushed into a region ($\mathcal{R}_1$) and the corresponding path (ξ_1) is determined to be valid.

find an edge selector $\texttt{Select}(\phi)$ that minimizes the expected cost required to find a valid path.

$$\min \ \mathbb{E}_{\phi \in P(\phi)} \left[c(\texttt{Select}(\phi)) \right] \ \text{s.t} \ \forall \phi, \exists \xi_i \in \Xi \ : \ P(\xi_i \mid \texttt{Select}, \phi) = 1 \tag{8.1}$$

8.3.2 The Decision Region Determination problem

We will now define the Bayesian active learning problem of *Decision Region Determination* while highlighting equivalence with the feasible path identification problem. Let $\mathcal{H} = \{h_1, \ldots, h_n\}$ be a set of "hypotheses", each of which is analogous to a world. We have a prior distribution $P(h)$ on this set. A "test" $t \in \mathcal{T}$ is performed by querying a corresponding edge $e \in \mathcal{E}$ for evaluation, which returns a binary outcome $x \in \{0, 1\}$ denoting if an edge is valid or not. Thus each hypothesis can be considered a function, $h : \mathcal{T} \to \{0, 1\}$, mapping tests to corresponding outcomes. The cost of performing a test is $c(t)$. A path $\xi_i \in \Xi$ corresponds to a set of worlds on which that path is valid. Hence each path $\xi_i \in \Xi$ corresponds to a "decision region" $\mathcal{R}_i \subseteq \mathcal{H}$ over the space of hypotheses. Let $\{\mathcal{R}_i\}_{i=1}^m$ be the set of "decision regions" corresponding to Ξ.

For a set of tests $\mathcal{A} \subseteq \mathcal{T}$ that are performed, let the observed outcome vector be denoted by $\mathbf{x}_\mathcal{A}$. Let the version space $\mathcal{H}(\mathbf{x}_\mathcal{A})$ be the set of hypotheses consistent with outcome vector $\mathbf{x}_\mathcal{A}$, i.e. $\mathcal{H}(\mathbf{x}_\mathcal{A}) = \{h \in \mathcal{H} \ \mid \ \forall t \in \mathcal{A}, h(t) = \mathbf{x}_\mathcal{A}(t)\}$.

We define a policy π as a mapping from the current outcome vector $\mathbf{x}_\mathcal{A}$ to the next test to select. A policy terminates when at least one region is valid, or all regions are invalid. Let h be the underlying world on which it is evaluated. Denote the outcome vector of a policy π as $\mathbf{x}_\mathcal{A}(\pi, h)$. The expected cost of a policy π is $c(\pi) = \mathbb{E}_{h \sim P(h)} \left[c(\mathbf{x}_\mathcal{A}(\pi, h)) \right]$ where $c(\mathbf{x}_\mathcal{A})$ is the cost of all tests $t \in \mathcal{A}$.

Problem 11 (Decision Region Determination (DRD))**.** Given a distribution of hypotheses, $P(h)$, the objective is to compute a policy with minimum expected cost that ensures at least one region is valid, i.e.

$$\pi^* \in \arg\min_\pi \ c(\pi) \ \text{s.t} \ \forall h, \exists \mathcal{R}_d \ : \ P(\mathcal{R}_d \mid \mathbf{x}_\mathcal{A}(\pi, h)) = 1 \tag{8.2}$$

Note that we can cast Problem FEAS-PATH to Problem DRD by setting $\mathcal{E} = \mathcal{T}$ and $\Xi = \{\mathcal{R}_i\}_{i=1}^m$. That is, driving uncertainty into a region is equivalent to identification of a valid path (Fig. 8.3). This casting enables us to leverage efficient algorithms with near-optimality guarantees for motion planning.

8.3.3 Special case: The DRD problem with Independent Bernoulli Tests

We now examine the special case scenario where we are planning on a *Generalized Binomial Graph (GBG)* [Frieze and Karoński, 2015]. In this scenario, the edges in the graph are independent Bernoulli random variables. Generally, edges in this graph are correlated, due to the implicit structure of the world. However, in many situations, measuring such correlations is challenging, particularly when there is insufficient data. Assuming independent edges is a common simplification [Burns and Brock, 2005a, Choudhury et al., 2016b, Dellin and Srinivasa, 2016, LaValle, 2006, Narayanan and Likhachev, 2017].

Similar to Section 8.3.2, we can define an equivalent problem - *the Decision Region Determination problem with Independent Bernoulli Tests*. We define a set of tests $\mathcal{T} = \{1, \ldots, n\}$, where the outcome of each test is a Bernoulli random variable $X_t \in \{0, 1\}$, $P(X_t = x_t) = \theta_t^{x_t}(1-\theta_t)^{1-x_t}$. We define a set of hypotheses $h \in \mathcal{H}$, where each is an outcome vector $h \in \{0, 1\}^{\mathcal{T}}$ mapping all tests $t \in \mathcal{T}$ to outcomes $h(t)$. We define a set of regions $\{\mathcal{R}_i\}_{i=1}^m$, each of which is a subset of tests $\mathcal{R} \subseteq \mathcal{T}$. A region is determined to be valid if all tests in that region evaluate to true, which has probability $P(\mathcal{R}) = \prod_{t \in \mathcal{R}} P(X_t = 1)$.

We define a policy π as a mapping from observation vector $\mathbf{x}_\mathcal{A}$ to tests. A policy terminates when it shows that at least one region is valid, or all regions are invalid. Let $\mathbf{x}_\mathcal{T} \in \{0, 1\}^{\mathcal{T}}$ be the ground truth - the outcome vector for all tests. Denote the observation vector of a policy π given ground truth $\mathbf{x}_\mathcal{T}$ as $\mathbf{x}_\mathcal{A}(\pi, \mathbf{x}_\mathcal{T})$. The expected cost of a policy π is $c(\pi) = \mathbb{E}_{\mathbf{x}_\mathcal{T}}[c(\mathbf{x}_\mathcal{A}(\pi, \mathbf{x}_\mathcal{T})]$ where $c(\mathbf{x}_\mathcal{A})$ is the cost of all tests $t \in \mathcal{A}$.

Problem 12 (Decision Region Determination with Independent Bernoulli Tests (BERN-DRD))**.** Given a Bernoulli distribution on ground truth $\mathbf{x}_\mathcal{T}$, $P(h)$, the objective is to compute a policy with minimum expected cost that ensures at least one region is valid, i.e.

$$\pi^* \in \arg\min_{\pi} \; c(\pi) \text{ s.t } \forall \mathbf{x}_\mathcal{T}, \exists \mathcal{R}_d \; : \; P(\mathcal{R}_d \mid \mathbf{x}_\mathcal{A}(\pi, \mathbf{x}_\mathcal{T})) = 1 \tag{8.3}$$

8.4 The Decision Region Edge Cutting algorithm (DiRECt)

In order to solve Problem DRD, we adopt the framework of *Decision Region Edge Cutting (*DIRECT*)* [Chen et al., 2015a]. The intuition behind the method is as follows - as tests are performed, hypotheses inconsistent with test outcomes are pruned away. Hence, tests should be incentivized to push the probability mass over hypotheses into a region as fast as possible. Chen et al. [2015a] derive a surrogate objective function that not only provides such an incentive, but also exhibits the property of adaptive submodularity [Golovin and Krause, 2011] - greedily maximizing such an objective results in a near-optimal policy.

DIRECT uses a key result from Golovin et al. [2010] who address the *Equivalence Class Determination* (ECD) problem - a special case of the DRD problem (8.2) when *regions are*

disjoint. Let $\{\mathcal{R}_1, \ldots, \mathcal{R}_m\}$ be a set of disjoint regions, i.e, $\mathcal{R}_i \cap \mathcal{R}_j = 0$ for $i \neq j$. Golovin et al. [2010] provide an efficient yet near-optimal approach for solving ECD in their EC^2 algorithm. The EC^2 algorithm defines a graph $\mathcal{G}_{\text{EC}} = (\mathcal{V}_{\text{EC}}, \mathcal{E}_{\text{EC}})$ where the nodes are hypotheses and edges are between hypotheses in different decision regions $\mathcal{E}_{\text{EC}} = \cup_{i \neq j} \{\{h, h'\} \mid h \in \mathcal{R}_i, h' \in \mathcal{R}_j\}$. The weight of an edge is defined as $w(\{h, h'\}) = P(h)P(h')$. An edge is said to be 'cut' by an observation if either hypothesis is inconsistent with the observation. Hence a test t with outcome x_t is said to cut a set of edges $\mathcal{E}_{\text{EC}}(x_t) = \{\{h, h'\} \mid h(t) \neq x_t \vee h'(t) \neq x_t\}$. The aim is to cut all edges by performing test while minimizing cost.

EC^2 employs a weight function over regions, $w_{\text{EC}}(\{\mathcal{R}_i\}) = \sum_{i \neq j} P(\mathcal{R}_i)P(\mathcal{R}_j)$. Naively, computing the total edge weight requires enumerating all pairs of regions. However, we can compute this efficiently in linear complexity as $w_{\text{EC}}(\{\mathcal{R}_i\}) = \frac{1}{2}\left((\sum_i P(\mathcal{R}_i))^2 - \sum_i P(\mathcal{R}_i)^2\right)$. EC^2 defines an objective function $f_{\text{EC}}(\mathbf{x}_{\mathcal{A}})$ that measures the ratio of the original weight of subregions $\mathcal{R}_i$ and the weight of pruned subregions $\mathcal{R}_i \cap \mathcal{H}(\mathbf{x}_{\mathcal{A}})$, i.e.

$$f_{\text{EC}}(\mathbf{x}_{\mathcal{A}}) = 1 - \frac{w_{\text{EC}}(\{\mathcal{R}_i\} \cap \mathcal{H}(\mathbf{x}_{\mathcal{A}}))}{w_{\text{EC}}(\{\mathcal{R}_i\})} \tag{8.4}$$

EC^2 uses the fact that $f_{\text{EC}}(\mathbf{x}_{\mathcal{A}})$ is *adaptive submodular* [Golovin and Krause, 2011] to define a greedy algorithm. Let the expected marginal gain of a test be $\Delta_{f_{\text{EC}}}(t \mid x) = \mathbb{E}_{x_t}\left[f_{\text{EC}}(\mathbf{x}_{\mathcal{A}\cup\{t\}}) - f_{\text{EC}}(\mathbf{x}_{\mathcal{A}}) \mid \mathbf{x}_{\mathcal{A}}\right]$. EC^2 greedily selects a test $t^* \in \arg\max_t \frac{\Delta_{f_{\text{EC}}}(t \mid \mathbf{x}_{\mathcal{A}})}{c(t)}$.

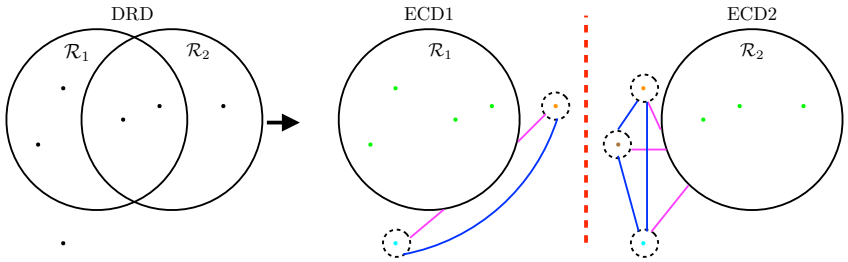


Figure 8.4: The DRD problem split into 'one region versus all' ECD problems by the DIRECT algorithm

We now return to the general DRD problem where regions are not disjoint. DIRECT reduces the DRD problem with m regions to m instances of the ECD problem. Each ECD problem is a 'one region versus all'. ECD problem i is defined over the following disjoint regions: the first region is $\mathcal{R}_i$ and the remaining regions are singletons containing only one hypothesis $h \notin \mathcal{R}_i$. The EC^2 objective corresponding to this problem is $f^r_{\text{EC}}(\mathbf{x}_{\mathcal{A}})$. The key idea is that *solving any one ECD problem solves the DRD problem*. The DIRECT algorithm then combines them in a *Noisy-OR* formulation by defining the following combined objective

$$f_{\text{DRD}}(\mathbf{x}_{\mathcal{A}}) = 1 - \prod_{r=1}^{m}(1 - f^r_{\text{EC}}(\mathbf{x}_{\mathcal{A}})) \tag{8.5}$$

DIRECT uses the fact that $f_{\text{DRD}}(\mathbf{x}_{\mathcal{A}})$ is also *adaptive submodular* to greedily select a test $t^* \in \arg\max_t \frac{\Delta_{f_{\text{DRD}}}(t \mid \mathbf{x}_{\mathcal{A}})}{c(t)}$.

We state the near optimality property of DIRECT and refer the reader to Chen et al. [2015a] for proofs and other associated lemmas.

Theorem 8.1 (Near Optimality of DIRECT). Let m be the number of regions, $p^h_{\min}$ the minimum prior probability of any hypothesis, π_{DRD} be the greedy policy and π^* with the optimal policy. We have

$$c(\pi_{DRD}) \leq c(\pi^*)(2m \log \frac{1}{p^h_{\min}} + 1) \tag{8.6}$$

8.5 The Bernoulli Subregion Edge Cutting algorithm (BiSECt)

We now address Problem BERN-DRD. We follow the framework of *Decision Region Edge Cutting (*DIRECT*)* [Chen et al., 2015a] by creating separate sub-problems for each region, and combining them. For each sub-problem, we provide a modification to EC^2 which is simpler to compute when the distribution over hypotheses is non-uniform, while providing the same guarantees. Unfortunately, naively applying this method requires $\mathcal{O}\left(2^{\mathcal{T}}\right)$ computation per sub-problem. For the special case of independent Bernoulli tests, we present a more efficient *Bernoulli Subregion Edge Cutting (*BISECT*)* algorithm, which computes each subproblem in $\mathcal{O}(\mathcal{T})$ time. We briefly review the algorithm and refer the reader to Choudhury et al. [2017c] for details.

8.5.1 A simple subproblem: One region versus all

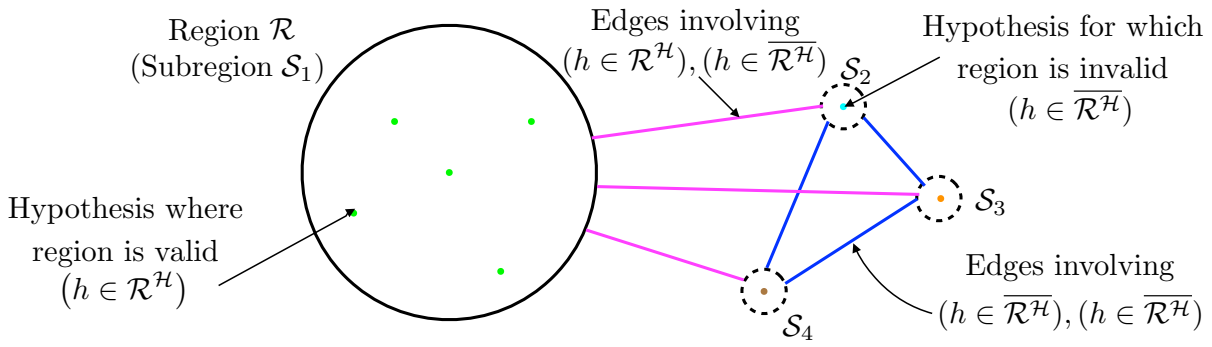


Figure 8.5: The 'one region versus all' ECD problem. The region $\mathcal{R}^{\mathcal{H}}$ is shown as a circle encompassing a set of consistent hypothesis h (green dots). Hypothesis for which the region is not valid lie outside the circle (dots in colors other than green). The objective is to compute an efficient policy to either force the probability mass in the region $\mathcal{R}^{\mathcal{H}}$ or determine the *unique* hypothesis $h \in \overline{\mathcal{R}^{\mathcal{H}}}$.

We will now define a simple subproblem whose solution will help in addressing the BERN-DRD problem. We define the 'one region versus all' subproblem as follows - given a *single region*, the objective is to either push the entire probability mass of the version space on a region or collapse it on a single relevant hypothesis. We view a region $\mathcal{R}$ as a version space $\mathcal{R}^{\mathcal{H}} \subseteq \mathcal{H}$ consistent with its constituent tests. We define this subproblem over a set of *disjoint subregions* $\mathcal{S}_i$. Let the hypotheses in the target region $\mathcal{R}^{\mathcal{H}}$ be $\mathcal{S}_1$.

We refer to hypothesis region $\mathcal{R}^{\mathcal{H}}$ as subregion $\mathcal{S}_1$ as shown in Fig.8.5. Every other hypothesis $h \in \overline{\mathcal{R}^{\mathcal{H}}}$ is defined as its own subregion $\mathcal{S}_i$. Determining which subregion is valid falls under the

framework of *Equivalence Class Determination* (ECD), (a special case of the DRD problem) and can be solved efficiently by the EC^2 algorithm (Golovin et al. [2010]).

The EC^2 algorithm

As described in Section 8.4, the EC^2 algorithm [Golovin et al., 2010] solves this problem by creating a graph cut problem. It defines a graph $\mathcal{G} = (\mathcal{V}_{\text{EC}}, \mathcal{E}_{\text{EC}})$ where the nodes are hypotheses and edges are between hypotheses in different decision regions $\mathcal{E}_{\text{EC}} = \cup_{i \neq j} \{\{h, h'\} \mid h \in \mathcal{S}_i, h' \in \mathcal{S}_j\}$. The weight of an edge is defined as $w(\{h, h'\}) = P(h)P(h')$. The weight function over sub-regions is:

$$w_{\text{[Golovin et al., 2010]}}(\{\mathcal{S}_i\}) = \sum_{i \neq j} P(\mathcal{S}_i)P(\mathcal{S}_j) \tag{8.7}$$

When hypotheses have uniform weight, this can be computed efficiently for the 'one region versus all' subproblem. Let $P(\overline{\mathcal{S}_1}) = \sum_{i>1} P(\mathcal{S}_i)$:

$$w_{\text{[Golovin et al., 2010]}}(\{\mathcal{S}_i\}) = P(\mathcal{S}_1)P(\overline{\mathcal{S}_1}) + P(\overline{\mathcal{S}_1})\left(P(\overline{\mathcal{S}_1}) - \frac{1}{|\mathcal{H}|}\right) \tag{8.8}$$

EC^2 defines an objective function $f_{\text{EC}}(\mathbf{x}_{\mathcal{A}})$ that measures the weight of edges cut. This is the difference between the original weight of subregions $\mathcal{S}_i$ and the weight of pruned subregions $\mathcal{S}_i \cap \mathcal{H}(\mathbf{x}_{\mathcal{A}})$, i.e. $f_{\text{EC}}(\mathbf{x}_{\mathcal{A}}) = w_{\text{[Golovin et al., 2010]}}(\{\mathcal{S}_i\}) - w_{\text{[Golovin et al., 2010]}}(\{\mathcal{S}_i\} \cap \mathcal{H}(\mathbf{x}_{\mathcal{A}}))$.

EC^2 uses the fact that $f_{\text{EC}}(\mathbf{x}_{\mathcal{A}})$ is *adaptive submodular* (Golovin and Krause [2011]) to define a greedy algorithm. Let the expected marginal gain of a test be $\Delta_{f_{\text{EC}}}(t \mid x) = \mathbb{E}_{x_t}\left[f_{\text{EC}}(\mathbf{x}_{\mathcal{A}\cup\{t\}}) - f_{\text{EC}}(\mathbf{x}_{\mathcal{A}}) \mid \mathbf{x}_{\mathcal{A}}\right]$. EC^2 greedily selects a test $t^* \in \arg\max_t \frac{\Delta_{f_{\text{EC}}}(t \mid \mathbf{x}_{\mathcal{A}})}{c(t)}$.

An alternative to EC^2 on the 'one region versus all' problem

For non-uniform prior the quantity (8.8) is more difficult to compute. We modify this objective slightly, adding self-edges on subregions $\mathcal{S}_i, i > 1$ as shown in Fig. 8.5, enabling more efficient computation while still maintaining the same guarantees:

$$\begin{aligned} w_{\text{EC}}(\{\mathcal{S}_i\}) &= P(\mathcal{S}_1)(\sum_{i\neq 1} P(\mathcal{S}_i)) + (\sum_{i\neq 1} P(\mathcal{S}_i))(\sum_{j\geq i} P(\mathcal{S}_j)) \\ &= P(\mathcal{S}_1)P(\overline{\mathcal{S}_1}) + P(\overline{\mathcal{S}_1})^2 \\ &= P(\mathcal{R}^{\mathcal{H}})P(\overline{\mathcal{R}^{\mathcal{H}}}) + P(\overline{\mathcal{R}^{\mathcal{H}}})P(\overline{\mathcal{R}^{\mathcal{H}}}) \\ &= P(\overline{\mathcal{R}^{\mathcal{H}}})(P(\mathcal{R}^{\mathcal{H}}) + P(\overline{\mathcal{R}^{\mathcal{H}}})) \\ &= 1 - \prod_{i\in\mathcal{R}} \theta_i \end{aligned} \tag{8.9}$$

Similarly we can compute $w_{\mathrm{EC}}(\{\mathcal{S}_i\} \cap \mathcal{H}^{\mathcal{R}}(\mathbf{x}_{\mathcal{A}}))$

$$\begin{aligned}
&w_{\mathrm{EC}}(\{\mathcal{S}_i\} \cap \mathcal{H}^{\mathcal{R}}(\mathbf{x}_{\mathcal{A}})) \\
&= P(\mathcal{S}_1 \cap \mathcal{H}^{\mathcal{R}}(\mathbf{x}_{\mathcal{A}}))P(\overline{\mathcal{S}_1} \cap \mathcal{H}^{\mathcal{R}}(\mathbf{x}_{\mathcal{A}})) + P(\overline{\mathcal{S}_1} \cap \mathcal{H}^{\mathcal{R}}(\mathbf{x}_{\mathcal{A}}))^2 \\
&= P(\mathcal{R} \cap \mathcal{H}^{\mathcal{R}}(\mathbf{x}_{\mathcal{A}}))P(\overline{\mathcal{R}} \cap \mathcal{H}^{\mathcal{R}}(\mathbf{x}_{\mathcal{A}})) + P(\overline{\mathcal{R}} \cap \mathcal{H}^{\mathcal{R}}(\mathbf{x}_{\mathcal{A}}))P(\overline{\mathcal{R}} \cap \mathcal{H}^{\mathcal{R}}(\mathbf{x}_{\mathcal{A}})) \\
&= P(\overline{\mathcal{R}} \cap \mathcal{H}^{\mathcal{R}}(\mathbf{x}_{\mathcal{A}}))(P(\mathcal{R} \cap \mathcal{H}^{\mathcal{R}}(\mathbf{x}_{\mathcal{A}})) + P(\overline{\mathcal{R}} \cap \mathcal{H}^{\mathcal{R}}(\mathbf{x}_{\mathcal{A}}))) \\
&= \left(1 - \prod_{i \in (\mathcal{R} \cap \mathcal{A})} \mathbb{I}(X_i = 1) \prod_{j \in (\mathcal{R} \setminus \mathcal{A})} \theta_j\right) \left(\prod_{k \in \mathcal{R} \cap \mathcal{A}} \theta_k^{\mathbf{x}_{\mathcal{A}}(k)} (1 - \theta_k)^{1 - \mathbf{x}_{\mathcal{A}}(k)}\right)^2
\end{aligned} \tag{8.10}$$

Using (8.9) and (8.10) we can express $f_{\mathrm{EC}}(\mathbf{x}_{\mathcal{A}}) =$

$$\begin{aligned}
&= 1 - \frac{w_{\mathrm{EC}}(\{\mathcal{S}_i\} \cap \mathcal{H}^{\mathcal{R}}(\mathbf{x}_{\mathcal{A}}))}{w_{\mathrm{EC}}(\{\mathcal{S}_i\})} \\
&= 1 - \frac{\left(1 - \prod_{i \in (\mathcal{R} \cap \mathcal{A})} \mathbb{I}(X_i = 1) \prod_{j \in (\mathcal{R} \setminus \mathcal{A})} \theta_j\right) \left(\prod_{k \in \mathcal{R} \cap \mathcal{A}} \theta_k^{\mathbf{x}_{\mathcal{A}}(k)} (1 - \theta_k)^{1 - \mathbf{x}_{\mathcal{A}}(k)}\right)^2}{1 - \prod_{i \in \mathcal{R}} \theta_i}
\end{aligned} \tag{8.11}$$

Lemma 8.2. The expression $f_{\mathrm{EC}}(\mathbf{x}_{\mathcal{A}})$ is strongly adaptive monotone and adaptive submodular.

Proof. See Choudhury et al. [2017c] □

Improvement in runtime from exponential to linear

For non-uniform priors, computing (8.7) is difficult. The naive approach is to compute all hypothesis and assign them to correct subregions and then compute the weights. This has a runtime of a runtime of $\mathcal{O}\left(2^{\mathcal{T}}\right)$. However, our expression (8.11) can be computed in $\mathcal{O}(\mathcal{T})$. This is because of the simplifications induced by the independent bernoulli assumption. Since we have to repeat this computation every iteration of the algorithm, we can reduce this to $\mathcal{O}(1)$ through memoization. If we memoize $\left(1 - \prod_{i \in (\mathcal{R} \cap \mathcal{A})} \mathbb{I}(X_i = 1) \prod_{j \in (\mathcal{R} \setminus \mathcal{A})} \theta_j\right)$, we can incrementally update it every time a test t is evaluated. We also need to memoize $\left(\prod_{k \in \mathcal{R} \cap \mathcal{A}} \theta_k^{\mathbf{x}_{\mathcal{A}}(k)} (1 - \theta_k)^{1 - \mathbf{x}_{\mathcal{A}}(k)}\right)^2$ and update it incrementally.

8.5.2 Solving the original DRD problem using BiSECt

We now return to the Problem Bern-DRD where we have multiple regions $\{\mathcal{R}_1, \ldots, \mathcal{R}_m\}$ that can overlap and the goal is to push the probability into one such region. Similar to DiRECt (Chen et al. [2015a]), we apply BiSECt to solve the problem. We can now evaluate the DRD

objective in (8.5) using (8.11)

$$\begin{aligned}
&f_{\mathrm{DRD}}(\mathbf{x}_{\mathcal{A}}) \\
&= 1 - \prod_{r=1}^{m}(1 - f^{r}_{\mathrm{EC}}(\mathbf{x}_{\mathcal{A}})) \\
&= 1 - \prod_{r=1}^{m}\left(\frac{\left(1 - \prod_{i\in(\mathcal{R}_r\cap\mathcal{A})}\mathbb{I}\left(X_i = 1\right)\prod_{j\in(\mathcal{R}_r\setminus\mathcal{A})}\theta_j\right)\left(\prod_{k\in\mathcal{R}_r\cap\mathcal{A}}\theta_k^{\mathbf{x}_{\mathcal{A}}(k)}(1-\theta_k)^{1-\mathbf{x}_{\mathcal{A}}(k)}\right)^{2}}{1 - \prod_{i\in\mathcal{R}_r}\theta_i}\right)
\end{aligned} \tag{8.12}$$

The vanilla version of BISECT selects from candidate tests $\mathcal{T}_{\mathrm{cand}}$ that contains only tests belonging to active regions that have not already been evaluated

$$\mathcal{T}_{\mathrm{cand}} = \left\{\bigcup_{i=1}^{m}\left\{\mathcal{R}_i \;\middle|\; P(\mathcal{R}_i|\mathbf{x}_{\mathcal{A}}) > 0\right\}\right\} \setminus \mathcal{A} \tag{8.13}$$

We now examine the BISECT test selection rule which can be simplified as

$$\begin{aligned}
t^* &\in \arg\max_{t\in\mathcal{T}_{\mathrm{cand}}} \frac{\Delta_{f_{\mathrm{DRD}}}(t \mid \mathbf{x}_{\mathcal{A}})}{c(t)} \\
&\in \arg\max_{t\in\mathcal{T}_{\mathrm{cand}}} \frac{\mathbb{E}_{x_t}\left[f_{\mathrm{DRD}}(\mathbf{x}_{\mathcal{A}\cup\{t\}}) - f_{\mathrm{DRD}}(\mathbf{x}_{\mathcal{A}}) \mid \mathbf{x}_{\mathcal{A}}\right]}{c(t)} \\
&\in \arg\max_{t\in\mathcal{T}_{\mathrm{cand}}} \frac{1}{c(t)}\mathbb{E}_{x_t}\left[\prod_{r=1}^{m}\left(1 - \prod_{i\in(\mathcal{R}_r\cap\mathcal{A})}\mathbb{I}\left(X_i = 1\right)\prod_{j\in(\mathcal{R}_r\setminus\mathcal{A})}\theta_j\right)\right. \\
&\quad \left. - \left(\prod_{r=1}^{m}\left(1 - \prod_{i\in(\mathcal{R}_r\cap\mathcal{A}\cup t)}\mathbb{I}\left(X_i = 1\right)\prod_{j\in(\mathcal{R}_r\setminus\mathcal{A}\cup t)}\theta_j\right)\right)\left(\theta_t^{x_t}(1-\theta_t)^{1-x_t}\right)^{2\sum_{k=1}^{m}\mathbb{I}(t\in\mathcal{R}_k)}\right]
\end{aligned} \tag{8.14}$$

Theorem 8.3. Let m be the number of regions, $p^h_{\min}$ the minimum prior probability of any hypothesis, π_{DRD} be the greedy policy and π^* with the optimal policy. Then $c(\pi_{DRD}) \leq c(\pi^*)(2m\log\frac{1}{p^h_{\min}} + 1)$.

Proof. See Choudhury et al. [2017c] □

We now discuss the complexity of computing the marginal gain at each iteration. We have to cycle through n tests. For each tests, we only have to cycle through regions which it impacts. Let η be the maximum number of regions that any test belongs to. For every region, we need to do an $O(1)$ operation of calculating the change in probability. Hence the complexity is $O(n\eta)$. Note that this can be faster in practice by leveraging lazy methods in adaptive submodular problems (Golovin and Krause [2011]).

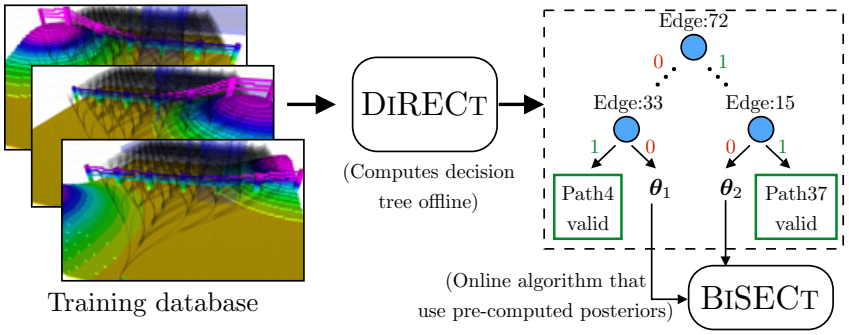


Figure 8.6: The overall approach framework. A training database is created by randomly sampling worlds from a generative model, collision checking the edge of the graph on each such world and creating a library of paths. The algorithm DIRECT is invoked to compute a decision tree offline. Each node of the tree contains the index of the edge to evaluate and branches on the outcome. The leaf node i of the tree correspond either to a feasible path existing or the number of consistent worlds dropping below a threshold fraction η. In the latter case, the bias vector $\boldsymbol{\theta}_i$ is stored. At test time, the tree is executed till a leaf node i is reached. If the problem is unsolved at that point, the BISECT algorithm is invoked with $\boldsymbol{\theta}_i$ as bias term.

8.5.3 Adaptively constraining test selection to most likely region

We observe in our experiments that the surrogate (8.12) suffers from a slow convergence problem - $f_{\text{DRD}}(\mathbf{x}_{\mathcal{A}})$ takes a long time to converge to 1 when greedily optimized. This can be attributed to the curvature of the function. To alleviate the convergence problem, we introduce an alternate candidate selection function that assigns to $\mathcal{T}_{\text{cand}}$ the set of all tests that belong to the most likely region $\mathcal{T}_{\text{maxP}}$. We hence forth denote the constraint as MAXPROBREG. It is evaluated as follows

$$\mathcal{T}_{\text{maxP}} = \left\{ \underset{\mathcal{R}_i=(\mathcal{R}_1,\mathcal{R}_2,\ldots,\mathcal{R}_m)}{\arg\max} P(\mathcal{R}_i|\mathbf{x}_{\mathcal{A}}) \right\} \setminus \mathcal{A} \tag{8.15}$$

Applying the constraint in (8.15) leads to a dramatic improvement for any test selection policy which we explore theoretically and empirically in Choudhury et al. [2017c].

The complexity reduces since we only have to visit states belonging to the most probable path. Finding the most probable path is an $O(m)$ operation. Let l be the maximum number of tests in a region. Hence the complexity of gain calculation is $O(l\eta)$. The total complexity is $O(l\eta + m)$.

8.6 Approach

8.6.1 Overview

Fig. 8.6 shows an overview of our approach. We sample a finite database of worlds to create a training dataset. We employ a greedy yet near-optimal algorithm DIRECT [Chen et al., 2015a] to solve the DRD problem. DIRECT chooses decisions to prune of inconsistent worlds from the database till it can ascertain if a path is valid. The decisions of DIRECT can be compactly stored in the form of a decision tree which is computed offline. At test time, the tree is executed till the leaf node is reached. At this point, either the problem is solved or the

Algorithm 14: DiRECt $(\mathcal{H}_{\text{act}}, \mathbf{R}, \mathbf{X}, \mathbf{c})$

```
for t ∈ 𝒯 do
    Δ(t) ← 0;
    for x_t ∈ {0, 1} do
        ℋ_cond ← {h ∈ ℋ_act | X(h, t) = x_t} ;          ▷ Prune hyp
        p ← |ℋ_cond| / |ℋ_act| ;                        ▷ Probability of outcome
        Δ(t) ← Δ(t) + p GainDRD(ℋ_cond, R);
    Δ(t) ← Δ(t) / c(t);
return arg max_{t∈𝒯} Δ(t);
```

Algorithm 15: GainDRD $(\mathcal{H}', \mathbf{R})$

```
v ← 1;
for i ∈ {1, ..., m} do
    v ← v ( WeightEC(ℋ', R, i) / WeightEC(ℋ, R, i) ) ;   ▷ Gain from each ECD
return v;
```

Algorithm 16: WeightEC $(\mathcal{H}', \mathbf{R}, i)$

```
a ← Σ_{h∈ℋ'} R(h, i) ;                                  ▷ Number of hyp in region
b ← |ℋ| − a ;                                           ▷ Remaining hyp
return 1/(2|ℋ|²) ((a + b)² − a² − b)
```

fraction of consistent worlds drops below a threshold η, i.e. it is likely that the test world is not in the database. In the latter case, we invoke another DRD algorithm, BiSECt. BiSECt implicitly reasons about the exhaustive set of $O(2^{\mathcal{E}})$ worlds and does this efficiently by assuming edges are independent. BiSECt is invoked with a bias vector of edge likelihoods $\boldsymbol{\theta}$ computed from the remaining consistent worlds in DiRECt. The combined behaviour of the framework is as follows - the tree makes a set of evaluations to quickly collapse the posterior on to a set of candidate paths, while BiSECt completes the episode being guided by the obtained posterior. We describe each component of the framework in the remaining subsections.

8.6.2 Creating an offline decision tree using DiRECt

We now provide a pseudo-code for the DiRECt algorithm described in Section 8.4. Algorithm 14 describes the DiRECt policy. $\mathcal{H}_{\text{act}}$ is the set of active hypotheses which have remained consistent so far with test outcomes. $\mathbf{R} \in \mathbb{R}^{n \times m}$ is a binary membership matrix where $\mathbf{R}(h, r) = 1$ if $h \in \mathcal{R}_r$. $\mathbf{X} \in \mathbb{R}^{n \times |\mathcal{T}|}$ is the test outcome matrix where $\mathbf{X}(h, t) = h(t)$. $\mathbf{c} \in \mathbb{R}^{|\mathcal{T}| \times 1}$ is a vector of test costs. Algorithm 14 computes the expected gain for each test by computing $\mathcal{H}_{\text{cond}}$, the set of hypotheses conditioned on test outcomes, and picks the best test. Algorithm 15 computes the DRD gain for $\mathcal{H}_{\text{cond}}$ by taking a product of individual ECD gains. Algorithm 16 calculates the weight of the i^{th} ECD problem. The computational complexity of Algorithm 14 is $\mathcal{O}(|\mathcal{T}|\, mn)$. Speedups can be obtained by lazily evaluating gains and using graph coloring to reduce the number of ECD

Algorithm 17: BISECT $(\{\mathcal{R}_i\}_{i=1}^m, \boldsymbol{\theta}, \mathcal{A})$

```
while (∄R_i, P(R_i|x_A) = 1) and (∃R_i, P(R_i|x_A) > 0) do
    T_cand ← SelectCandTestSet(x_A) ;        ▷ Using either (8.13) or (8.15)
    t* ← SelectTest(T_cand, θ, x_A) ;        ▷ Using (8.14)
    A ← A ∪ t*;
    x_t* ← Evaluate(t*) ;                    ▷ Observe outcome for selected test
```

problems [Chen et al., 2015a].

DIRECT needs access to the entire training database which can be prohibitive at runtime for storage and computational reasons. We circumvent this problem by computing a decision tree offline using DIRECT and storing it. The nodes of the tree encode which edge to evaluate. The tree branches on the outcome of the evaluation. Note that the depth of the tree is bounded by $\log_2(n)$ as all leaf nodes must be consistent with the training database. The size is further bounded by the fact that the tree terminates on a leaf node when the uncertainty has been pushed onto one region.

8.6.3 Executing BiSECt from the leaf node

As discussed in Section 8.1, it is impractical to have a database large enough to encompass all possible worlds that can arise at test time. Hence, if we reach the leaf node of the tree and the problem is still unsolved, we need to execute an online algorithm that can run to completion by reasoning over the exhaustive set of worlds. We use BISECT as described in Algorithm. 17.

BISECT needs as input a bias vector which corresponds to the independent likelihood of an edge being free. Since DIRECT has made a set of decisions to collapse the posterior, albeit on a finite database, we wish to use this to inform BISECT. We do this by growing the DIRECT decision tree only till the version space $\mathcal{H}_\eta$ drops below a fraction η of consistent worlds, i.e. $|\mathcal{H}_\eta| \leq \eta\,|\mathcal{H}|$. This is then used to create a bias vector $\boldsymbol{\theta}$ with a mixture term to ensure non-zero support for all plausible worlds. The bias term for a test t is

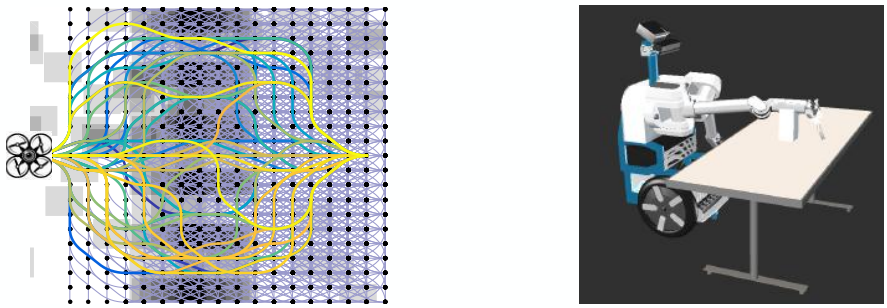
$$\boldsymbol{\theta}(t) = \alpha\,\frac{1}{|\mathcal{H}_\eta|}\sum_{h\in\mathcal{H}_\eta}\mathbf{X}(h,t) + (1-\alpha)\,0.5 \tag{8.16}$$

Using $\boldsymbol{\theta}$ leads to a more informed BISECT as compared to directly invoking BISECT from the beginning using a bias vector computed from the training database.

8.7 Experiments

We evaluated our approach on a collection of datasets spanning a spectrum of motion planning applications that range from simplistic yet insightful 2D problems to more realistic high dimension problems as encountered by a helicopter or a robot arm (Fig. 8.7). The autonomous helicopter dataset in particular is our target application.

A typical dataset is constructed as follows. The robot dynamics information is used to create an explicit graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a start and goal vertex. A dataset of n worlds is sampled from a designed generative model. Each edge is evaluated on each world to create a test outcome



(a) (b)

Figure 8.7: In addition to 2D planning, we also collect data sets of (a) Nonholonomic planning on state lattice for UAVs (b) 7D robot arm planning

matrix $\mathbf{X} \in \mathbb{R}^{n \times |\mathcal{T}|}$. A library of paths is created by solving for $k-$shortest paths on the dataset and sub-sampling it to maintain a size of m. This is then used to create a binary membership matrix $\mathbf{R} \in \mathbb{R}^{n \times m}$ encoding the validity of a path on a world. 10% of the data is used for test, remainder for training. The algorithms work with these abstract representations and do not need access to application specific details.

Our primary baseline is BISECT [Choudhury et al., 2017b] which treats each edge as independent Bernoulli random variables (i.e. averages $\mathbf{X}$ along each column to use as bias). We additionally use high performing baselines from Choudhury et al. [2017b] which were competitive with BISECT, i.e the MAXPROBREG version of MAXTALLY, SETCOVER and MVOI. We add to this the LAZYSP algorithm [Dellin and Srinivasa, 2016] which operates on the original graph $\mathcal{G}$. We also introduce a new algorithm LAZYSPSET which is restricted to the library of paths Ξ.

Our implementation is open sourced for MATLAB (`https://github.com/sanjibac/matlab_learning_collision_checking`). For details on the problem, baselines, dataset and other learning details, we refer the reader to Choudhury et al. [2017b].

8.7.1 Analysis of results

Fig. 8.9 shows the evaluation cost of all algorithms on various datasets normalized w.r.t DIRECT +BISECT. The two numbers are lower and upper 95% confidence intervals - hence it conveys how much fractionally poorer algorithms are w.r.t our approach. The best performance on each dataset is highlighted. Fig. 8.9 shows a comparison of algorithms on certain datasets. We present a set of observations to interpret these results.

O 1. DIRECT +BISECT has a consistently competitive performance across all datasets.

Fig. 8.9 shows on 17 datasets, DIRECT is at par with the best - on 8 of those it is exclusively the best.

O 2. DIRECT is more effective on environments with spatial correlation.

Fig. 8.8 shows that datasets such as TwoWall, MovingWall, Maze and Baffle are more struc-

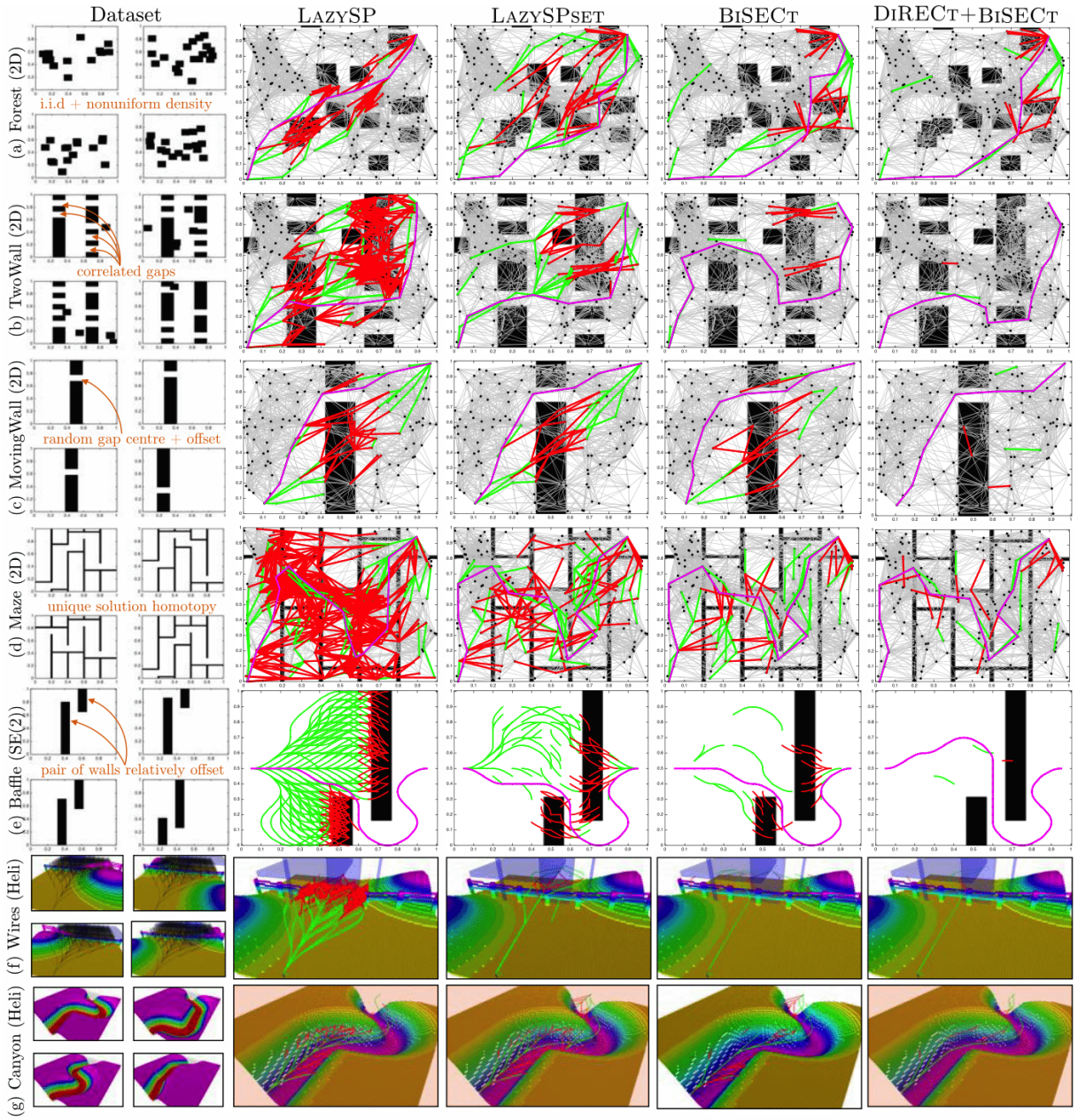


Figure 8.8: Comparison of LazySP, LazySPSet, BiSECt and DiRECt +BiSECt on a selection of datasets. 4 samples from each dataset is shown. The final performance of all algorithms on a test problem is shown: valid edges checked (green) and invalid edges checked (red).

tured. For example in the Maze dataset, there are 5 hallways with one interconnecting passage. DiRECt is able to locate this passage with a few checks and has better performance than BiSECt which assumes independence between edges. On the other hand the Forest dataset has less spatial correlation and BiSECt performs comparably (has an upper margin of 0.20). Similar phenomemnon was observed in 7D arm planning between Clutter (less correlation) and

	LazySP	LazySPSet	MaxTally	SetCover	MVoI	BiSECT	DiRECT + BiSECT
2D Geometric Planning: Variation across environments							
Forest	(10.90, 18.48)	(1.84, 3.02)	(0.17, 0.40)	(0.14, 0.51)	(0.30, 0.55)	(0.014, 0.20)	(0.00, 0.00)
OneWall	(7.47, 16.01)	(0.30, 0.71)	(0.00, 0.30)	(0.08, 0.34)	(0.09, 0.36)	(−0.06, 0.22)	(0.00, 0.00)
TwoWall	(21.54, 26.68)	(0.00, 0.21)	(0.20, 0.92)	(0.12, 0.58)	(0.31, 0.56)	(0.00, 0.53)	(0.00, 0.00)
MovingWall	(1.33, 3.01)	(1.00, 1.54)	(0.43, 1.17)	(0.35, 0.91)	(−0.03, 0.57)	(0.11, 0.92)	(0.00, 0.00)
Baffle	(7.86, 11.26)	(2.30, 3.83)	(0.33, 1.06)	(0.36, 0.74)	(0.26, 0.89)	(0.11, 0.55)	(0.00, 0.00)
Maze	(14.39, 19.66)	(1.16, 1.81)	(0.12, 0.34)	(0.00, 0.17)	(0.41, 0.87)	(0.44, 0.76)	(0.00, 0.00)
Bugtrap	(7.40, 8.57)	(2.74, 3.53)	(0.51, 0.84)	(−0.12, 0.54)	(−0.12, 0.53)	(0.43, 0.91)	(0.00, 0.00)
2D Geometric Planning (Baffle): Variation across path library size							
m : 200	(8.67, 11.20)	(1.73, 2.34)	(0.32, 0.73)	(0.38, 0.89)	(0.47, 1.17)	(−0.03, 0.58)	(0.00, 0.00)
m : 977	(7.20, 10.10)	(1.35, 2.92)	(0.24.0.38)	(0.31, 0.63)	(0.20, 0.79)	(0.03, 0.34)	(0.00, 0.00)
SE(2) Nonholonomic Path Planning: Variation across environments							
OneWall	(2.22, 4.18)	(0.15, 0.57)	(0.16, 0.48)	(−0.11, 0.07)	(0.00, 0.28)	(−0.07, 0.12)	(0.00, 0.00)
MovingWall	(−0.14, 0.23)	(−0.14, 0.15)	(0.24, 0.49)	(0.13, 0.41)	(0.000.36)	(0.10, 0.54)	(0.00, 0.00)
Baffle	(7.74, 10.48)	(2.88, 4.81)	(1.86, 3.21)	(1.35, 2.32)	(0.70, 1.47)	(1.14, 1.70)	(0.00, 0.00)
Bugtrap	(3.75, 6.51)	(2.27, 4.69)	(0.22, 0.52)	(0.05, 0.43)	(0.26, 0.55)	(0.12, 0.44)	(0.00, 0.00)
Autonomous Helicopter Path Planning: Variation across environments							
Wires	(17.42, 75.85)	(1.15, 3.08)	(0.55, 0.96)	(0.00, 0.25)	(−0.08, 0.08)	(0.08, 0.23)	(0.00, 0.00)
Canyon	(0.73, 1.27)	(1.41, 2.00)	(0.15, 0.52)	(0.07, 0.40)	(0.43, 0.72)	(0.06, 0.47)	(0.00, 0.00)
7D Arm Planning: Variation across environments							
Clutter	(0.49, 1.08)	(0.09, 0.57)	(−0.04, 0.05)	(0.00, 0.13)	(0.10, 0.32)	(0.00, 0.10)	(0.00, 0.00)
Table+Clutter	(0.94, 1.84)	(−0.22, 0.17)	(0.06, 0.51)	(0.05, 0.27)	(0.11, 0.46)	(0.06, 0.36)	(0.00, 0.00)

Figure 8.9: Normalized cost (with respect to our approach) of different algorithms on different datasets (lower and upper bounds of 95% C.I.)

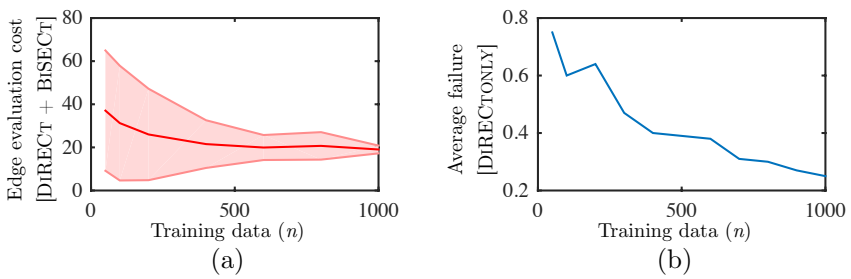


Figure 8.10: (a) Mean and variance of edge evaluation cost of DiRECT +BiSECT with increasing training size. (b) The average failure (to indentify a feasible path) rate when only using DiRECT (without BiSECT).

Table+Clutter (more correlation) datasets.

O 3. DiRECT +BiSECT improves in performance with more data.

Fig. 8.10(a) shows that both mean and variance reduce as the size of the dataset is increased. This is not only due to DiRECT having better realizability, but also due to BiSECT having a more accurate bias term.

O 4. BiSECT is essential as a post-processing step

We defined an algorithm, DiRECTonly that runs DiRECT to completion and randomly returns a path from the consistent set of paths, i.e. the a path DiRECT believes should be feasible. Fig. 8.10(b) shows the failure rate of DiRECTonly with training size, i.e. the returned path being infeasible. The plot shows the failure does not go to zero. BiSECT is essential to reason about the remaining paths and in which order to check edges to ascertain which path is

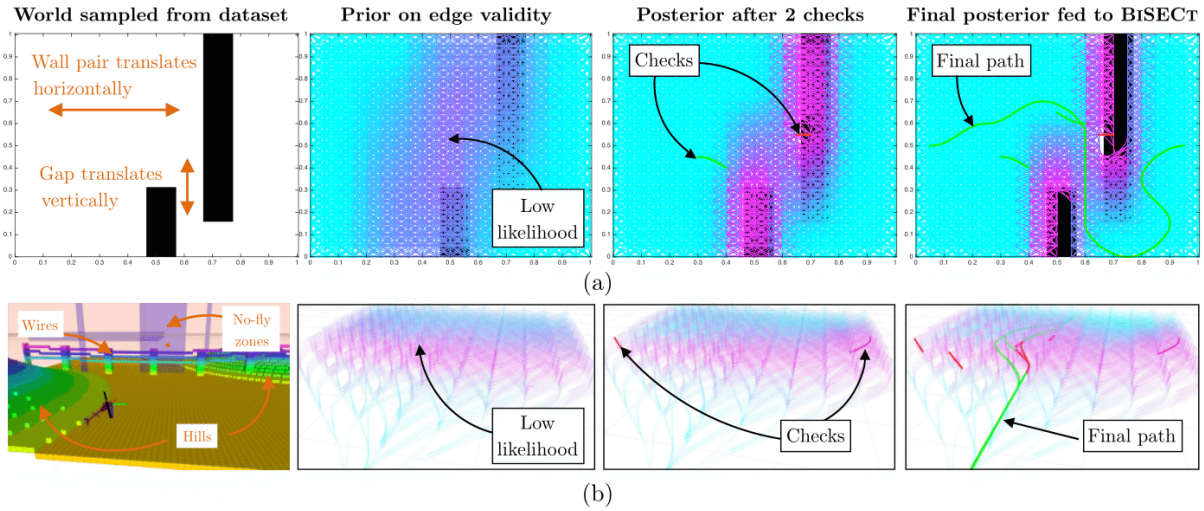


Figure 8.11: DiRECt performs edges evaluation to collapse the uncertainty about the validity of a path. (a) An example from the Baffle dataset for SE(2) nonholonomic path planning. Here two walls occur in a pair forcing the path to maneuver through the gap. The prior shows only a general location where obstacles are likely to occur. After 2 checks, DiRECt is able to locate the gap. The resultant posterior allows BiSECt to finish off the episode. (b) A realistic example from the Wires dataset for autonomous helicopter path planning. Here the helicopter is flying over a terrain that may have powerlines. The terrain also has natural obstacles such as hills. Presence of other aircrafts and no-fly zones also require avoidance. The prior shows a band of low likelihood region that corresponds to the presence of the wires. After 2 checks, DiRECt is able to infer the location of obstacles on either flank. The resultant posterior allows BiSECt to focus on the centre region and find a path.

free.

8.7.2 Case study A: Roles played by DiRECt and BiSECt

We take a closer look at the Baffle dataset for SE(2) path planning as shown in Fig. 8.11(a). The combination of the narrow gap between two walls and the curvature constraint of the robot makes this a challenging problem as shown by the performance of baseline LazySP in Fig. 8.8(e). Also note that BiSECt too struggles on this problem. Returning back to Fig. 8.11(a), we see that the prior over edge validity is not informative enough for BiSECt to find the gap. As DiRECt proceeds to collision check edges, it is quickly able to localize the gap between the two walls. Interestingly, it is *relatively uncertain about the actual vertical location of the wall* - this is reflective of DiRECt judiciously reducing uncertainty only enough to make a region valid (i.e to know if a candidate path would be feasible). The posterior is much more informative for BiSECt which is able to easily find a feasible path.

8.7.3 Case study B: Autonomous helicopter avoiding power-lines

We now closely examine planning problems encountered by an autonomous helicopter which is our target application. The helicopter navigates in a receding horizon fashion(for details refer to Chapter 9). It is equipped with a laser scanner that scans the world to build a model of obstacles and free space. The system is required to plan around detected obstacles as it performs various

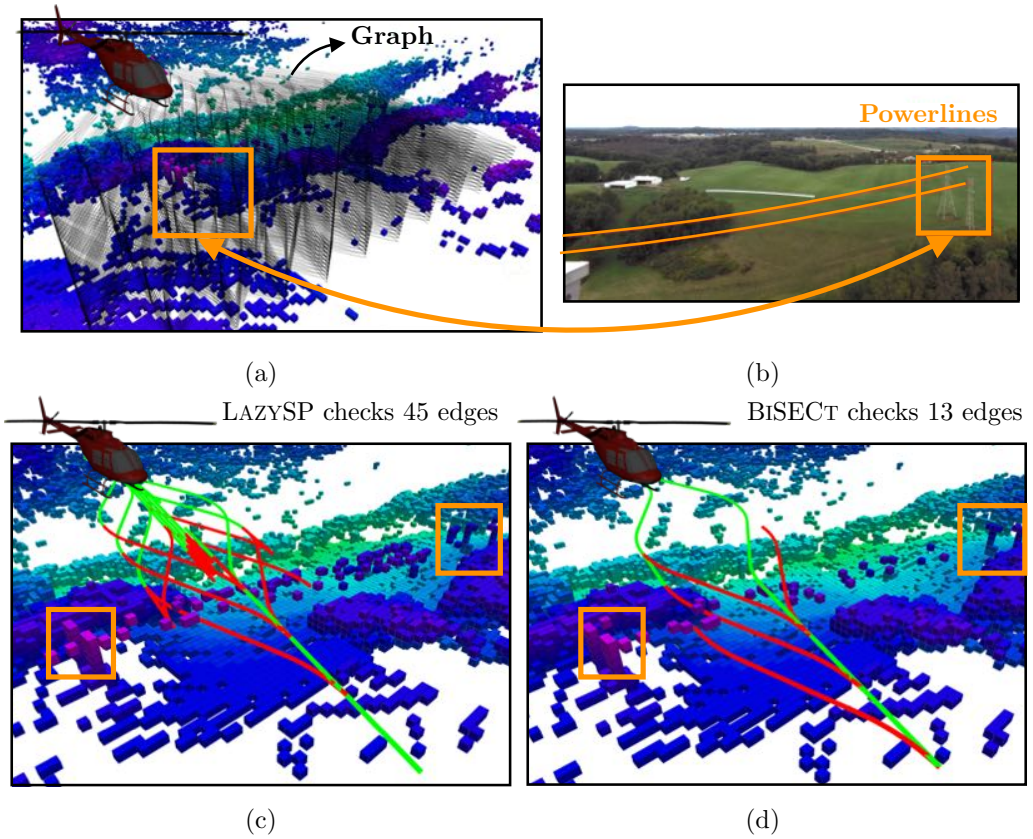


Figure 8.12: Application of active edge evaluation policies on real flight test data from an autonomous helicopter. (a) The world is sensed by the onboard laser and modelled as an occupancy grid. Since wires are thin structures, they are detected fairly late. (b) The camera view from onboard the helicopter. (c) LazySP evaluates 45 edges till it finds a feasible path. This is because it does not leverage any prior on the edge validity (d) Our approach (BiSECT) checks only 13 edges till it finds a feasible path.

missions. A particularly difficult problem is dealing with power-lines as the system comes in to land. The system has limitations on how fast it can ascend / descend. As a result, it does not suffice to simply move up as the planner must ensure that the system can reach the goal point.

We hand design a parametric distribution to simulate different configurations of power-lines along with other commonly occuring obstacles like no-fly-zones (see Fig. 8.8(f)). A close up of a situation is shown in Fig. 8.11(b) along with a visualization of edge likelihood.

We first evaluate our approach in a simulated situation shown in Fig. 8.11(b). We have a helicopter flying in a terrain with hills on either side and power-line in the middle. There are also no-fly-zones which correspond to unsafe areas such as radio towers, other aircrafts, etc. The environment has an implicit structure due the the power-lines, and the presence of hills that invalidate a large number of edges simultaneously. As DiRECT proceeds to collision check edges, it is quickly able to ascertain presence of hills in the two flanks and a gap in the centre. BiSECT uses this posterior to focus along the centre and find a path. In comparison LazySP evaluates much more edges as shown in Fig. 8.8(f).

We also evaluate our approach on real flight data collected from a full scale autonomous

helicopter. The situation is simpler in comparison to our previously simulated example. Fig. 8.12 shows the scenario. There are power-lines that are detected fairly late, so the planner has to ascertain how to both ascend and maneuver laterally to safely avoid obstacle. LazySP evaluates a lot of edges that intersect with the wire. BiSECt evaluates a small number of edges till it finds a feasible path.

8.8 Discussion and future work

We addressed the problem of identification of a feasible path from a library while minimizing the expected cost of edge evaluation given priors on the likelihood of edge validity. We showed that this problem is equivalent to the Bayesian active learning paradigm of Decision Region Determination (DRD) where the goal is to select tests (edges) that drive uncertainty into a single decision region (a valid path). We presented an algorithm DiRECt that efficiently solves the DRD problem. We developed a new algorithm BiSECt that solves the DRD problem under the special assumption of independent tests. We proposed an approach that combines these two algorithms, DiRECt and BiSECt, to efficiently solve our original problem. We validated our approach on a spectrum of problems against state of the art heuristics and showed that it has a consistent performance across datasets. These results demonstrate the efficacy of leveraging prior data to significantly reduce collision checking effort.

We now discuss some insights and directions for future work.

Q 1 (Generalization to All Paths)**.** Can we reason about set of all paths without explicitly enumerating them?

Consider the DRD problem on the set of all simple paths. Explicitly reasoning about such a set is expensive as the size of the set can be exponential in the number of edges in the graph. An alternate method is to directly reason about the utility of an edge conditioned on the distribution over all possible paths implicitly. This would involve a set of approximations such as the probability of a path $P(\xi)$ being valid is the product of the constituent edges e_i being valid. The goal would be to use this approximation to express the objective in (8.5) in terms of the probability of each edge. Dellin and Srinivasa [2016] examine a special case of this problem under some restrictive assumptions. They show that if one assumes a fully connected bi-directional graph, where validity of edge is proportional to edge weight, the partition function can be computed and updated exactly. It would be interesting to investigate if this idea can be applied in our framework.

An alternate technique to reasoning about all paths is by replacing the fixed library with a planner that generates a set of candidate paths Ξ. This can be framed as a two player game where the objective of the policy π is to find a feasible path in the set. The objective of the planner is to come up with a set of paths such that at least one is likely to be feasible. It would be interesting to see if the guarantees in our framework can be extended to this procedure of decoupling path library generation from verification.

Q 2 (Incorporate Solution Quality)**.** What is an optimal policy for identifying the shortest path in a set of candidate paths?

This is a fundamental question that requires contrasting the decision region *determination* problem with a new problem - the decision region *elimination* (DRE) problem. The DRE problem follows the same notation as our framework with a key difference in the objective

$$\pi^* \in \arg\min_{\pi} \; c(\pi) \text{ s.t } \forall h, \forall \mathcal{R}_d \; : \; P(\mathcal{R}_d \mid \mathbf{x}_{\mathcal{A}}(\pi, h)) = 0 \tag{8.17}$$

This can be shown to be equivalent to the stochastic set cover problem for which a greedy algorithm can be shown to have a $\log(n)$ near-optimality bound.

The decision region elimination problem plays a key role in verifying if a path is the shortest by requiring all shorter paths to be eliminated. This can be used to derive an algorithm for obtaining the shortest path in a set. For example consider the following strategy: We start with the P shortest paths in the library and invoked the DRE problem. When the library is eliminated, we consider the next P paths and repeat. This process terminates when we reach a set when the DRE algorithm terminates without eliminating the set (say on the $k+1$ iterations).

The downside to such a strategy is that it can be myopic in comparison to a policy that in hindsight could have eliminated all Pk paths. On the other hand, even though the DRD policy is not incentivized to identify the optimal solution, if can efficiently identify a feasible path from a set which is useful is compressing the upper bound on the length of the shortest path.

This motivates us to frame the *shortest path identification* in the following manner. Let $L(\mathcal{R}_i)$ be the length of the path ξ_i corresponding to $\mathcal{R}_i$. We wish to find a policy

$$\begin{aligned} \pi^* \in \arg\min_{\pi} \; c(\pi) \text{ s.t } \forall h, \exists \mathcal{R}_d \; : \; P(\mathcal{R}_d \mid \mathbf{x}_{\mathcal{A}}(\pi, h)) = 1 \; , \\ \forall \mathcal{R}_i \; , \; L(\mathcal{R}_i) \geq L(\mathcal{R}_d) \; : \; P(\mathcal{R}_i \mid \mathbf{x}_{\mathcal{A}}(\pi, h)) = 0 \end{aligned} \tag{8.18}$$

A possible direction would be to combine the attributes of DRD and DRE to arrive at a solution for (8.18).

Q 3 (Dealing with Data Starvation)**.** Can we leverage machine learning oracles that can predict a set of plausible worlds?

There has been a significant advancement in the area of using deep generative models such as Generative Adversarial Networks [Radford et al., 2015] and Variational AutoEncoders [Walker et al., 2016]. These tools have enjoyed a lot of success in the computer vision community in being able to generate plausible high dimensional vectors such as images. One can leverage such tools in planning for predicting plausible worlds.

We wish to train oracles that can predict a set of plausible worlds $\{\phi_1, \ldots, \phi_n\}$. Our framework can use this set online as the set of hypothesis. Furthermore, the oracles can be trained to produce such worlds conditioned on the outcome of the edge evaluations. This would alleviate the data starvation issue and allow the edge evaluation to perpetually reason about possible worlds in rolling buffer fashion.

Part IV

Bridging Theory and Practice

9

A Unified Planning Architecture for UAVs

In the previous chapters, we focused mainly on the algorithmic aspects of adaptive motion planning. In this chapter, we cast the spotlight on the problem of how to implement an adaptive planning system in practice. We examine a class of general UAV applications where the system is required to visit a sequence of waypoints, potentially spaced out across long distances. Such applications require the system to fly across different environments while operating at different speed regimes. We propose a general trajectory planning architecture for UAVs dealing with such problems. The architecture is based on a decoupling scheme that enables us to solve such planning problems tractably in real-time. A key component of the architecture is the adaptive planning module - this allows us to use the *same library of planning algorithms* across all UAV classes. We present results on 3 different UAV classes and show how the architecture seamlessly adapts to different planning problem distribution encountered by these systems.

9.1 Introduction

Unmanned rotor-craft have a high demand in applications such as cargo delivery, emergency rescue operations and surveillance due to their dexterity in operating in close vicinity to ground. Such applications typically require the UAV to visit a sequence of waypoints that may stretch across large distances and different environments. Such operations also involve flying at varying speed regimes, varying proximity to obstacles and repeatedly landing / taking off from unprepared sites. Hence, from a motion planning perspective, this problem has several challenging aspects.

The first challenge arises from the varying dynamic constraints encountered by UAVs. The commercial success of such systems depends heavily on their ability to produce high-performance flight profiles that optimize time while strictly adhering to constraints imposed by the control system, flight dynamics and performance charts [Prouty, 1995]. In conjunction with these requirements, these systems must be cognizant to the effect of wind on flight profile [McGee et al.,

2005, Seleck et al., 2013, Techy, 2011]. The dynamic constraints restricts the reachability of the system, i.e. limits the set of feasible trajectories the vehicle can execute. The reachability plays a critical role when the UAV has to avoid obstacles or plan maneuvers to fly through waypoints.

The second challenge arises from the fact that the robot has to guarantee safety while flying through partially known environments. Guaranteeing safety implies planning trajectories that satisfy safety constraints for an infinite time horizon. Hence the partial known aspect makes providing such guarantees challenging. Conservative approaches are inefficient since the missions require the vehicle to proceed as fast as possible. Unlike unmanned ground vehicles, coming to a stop is often not a safe option since rotorcrafts with a high payload have to move to stay airborne.

The final challenge arises from the fact that we wish to generalize across UAV classes. In other words, we want our planning software to transfer across domains without having to re-engineer the motion planners. This is challenging because different UAVs imply different dynamics constraints, different proximity to obstacles and different planning horizons. Even if planning algorithms are parametrized to deal with these variations, tuning such parameters is time consuming.

We present a general trajectory planning architecture for UAVs that deals with these challenges. Our approach is based on a novel 3 stage decoupling scheme - a global nominal trajectory planner, a local trajectory planner for avoiding obstacles and a safety checking executive module to guarantee safety. Each of these modules operate at different resolutions, different time scales and examine complimentary aspects of the problem. The global planner is responsible for producing a nominal trajectory that has guarantees with respect to the overall mission such as ensuring the vehicle can stay in a designated safe flight corridor, respect speed limits along segments and can feasibly transition between corridors in presence of wind. It is computationally expensive and is invoked once per mission. The local planner is responsible for locally repairing the trajectory to avoid sensed obstacles. This runs periodically ($\sim$ 1 Hz) to ensure obstacles are avoided shortly after they are sensed.

Finally, an executive module ensures safety for an infinite time horizon by only executing trajectories that have at least one evasive maneuver. An evasive maneuver is a set of safe states for an infinite time horizon (typically a loiter pattern that strictly lies in space identified as free by the sensor). The executive module is tasked with the job of checking if such maneuvers exist, and if not either slowing down the trajectory or in the worst case executing the evasive maneuver. The executive module reasons within the sensor horizon range and runs at the highest frequency ($\sim$ 10 Hz).

9.2 Problem formulation

We now present the trajectory planning problem that we wish to address. The overall block diagram for the system is shown in Fig. 9.1. We first describe the input and output to the trajectory planner module. We then describe the constraints and objective of the planner.

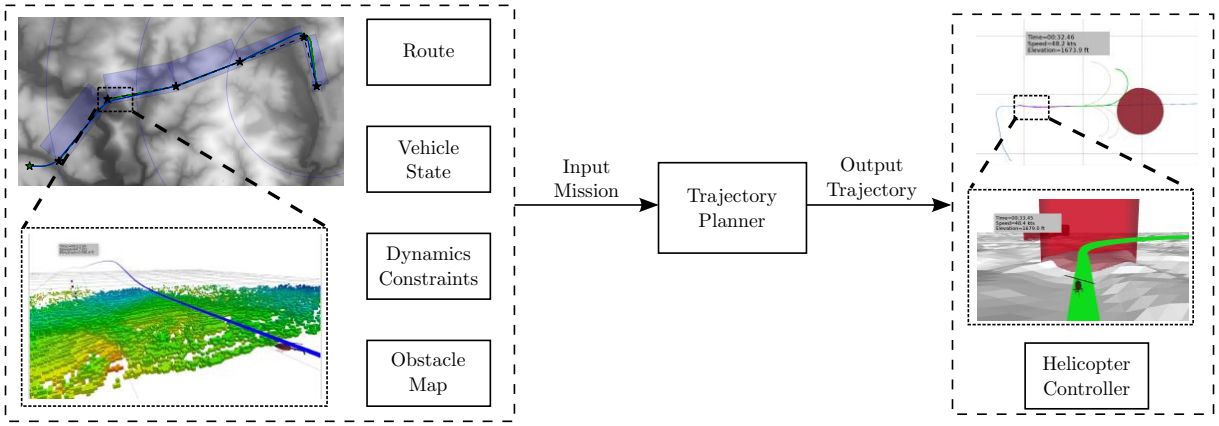


Figure 9.1: Overview of the input and output to the trajectory planner. The input is a mission. The first component of a mission is a route as shown in the top left figure. The route comprises of waypoints (black stars) and safety corridors (blue rectangles). The input mission also consists of the vehicle state, dynamic constraint and a representation of the world as an occupancy map (botton left figure). The output of the module is a trajectory. The top right figure shows such a trajectory (green) avoiding obstacles such as NFZs (red cylinder). The trajectories is also ensured to be guaranteed safe by verifying the existence of evasive maneuvers (orange segments).

9.2.1 Input and output

The input to the trajectory planner is a *mission* that fully specifies the intended behaviour and constraints of the system. This mission is typically created by a ground control operator but also can be generated autonomously by a mission planner. It consists of 4 main components:

1. *Route*: A route is a sequence of 3D waypoints that represents the locations that the system has to visit. Associated with each pair of waypoints is a desired speed and margins for a flight corridor (left, right, up, down). Hence, these numbers define a sequence of *safe flight corridor* that the system has to stay in, as well desired speeds that the system should achieve. It is not required for the system to actually fly through the waypoints. By controlling the width of the flight corridors, the vehicle can be restricted in a specified airspace - a critical requirement for UAVs. On the other hand, the corridor may contain obstacles such as terrain, other aircrafts or no-fly-zones that the system has to avoid. An illustration of a route is shown in Fig. 9.2

2. *Vehicle state*: This represents the current configuration of the system (pose and velocity) which serves as the initial boundary value for the trajectory planner. This is either the current pose of the system or the reference pose being tracked by the controller. It is expected that the trajectory planner returns a path that is constrained to pass through this state. Location on the trajectory the helicopter is tracking + pose. we plan from here

3. *Dynamics constraints*: This is a set of equality and inequality constraints that the trajectory must satisfy. These constraints are a combination of the actuator limits of the control system (e.g. maximum acceleration, maximum roll rate), constraints derived from

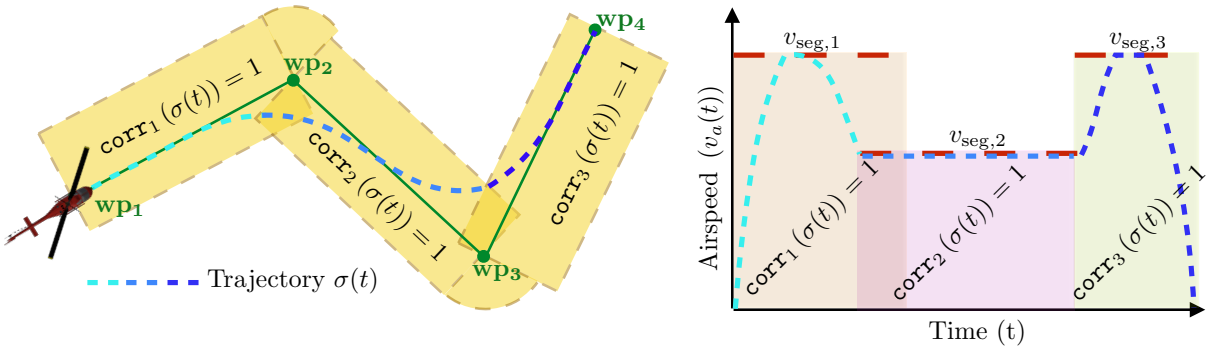


Figure 9.2: Illustrations of route constraints. (a) A route which defines a set of 3 safe corridors (yellow rectangles) $\texttt{corr}_1, \texttt{corr}_2, \texttt{corr}_3$. A feasible trajectory $\sigma(t)$ must lie inside all corridors. (b) A route also defines velocity constraints along the segments. The air speed of the trajectory $v_a(t)$ must not exceed constraint velocities $v_{\text{seg},1}, v_{\text{seg},2}, v_{\text{seg},3}$.

performance charts (e.g. torque limits, height-velocity limits) and constraints due to environmental factors (e.g wind).

4. *Obstacle map*: This is a map of the world that is being constructed online by the perception system. The trajectory is expected to stay clear of terrain obstacles by sufficient clearance.

The output of the planner is a time parameterized trajectory $\sigma(t) = \{x(t), y(t), z(t), \psi(t)\}$. The domain of the trajectory is $[0, t_f]$. This trajectory is sent to the control system as reference. We represent the trajectory as a spline, i.e. a sequence of polynomials. We choose the order to be 5 to ensure that it is sufficiently differentiable for the controller to obtain feed-forward terms.

9.2.2 Constraints

The trajectory $\sigma(t)$ computed by the planner has to satisfy a set of equality constraints $\mathcal{F}(\sigma) = 0$, inequality constraints $\mathcal{H}(\sigma) \leq 0$ and lie in valid space $\sigma(t) \in \Sigma_{\text{valid}}$. We briefly describe the different categories of constraints and refer the reader to Appendix A for formal definitions.

1. *Dynamics constraints*: We will abstract these as $\mathcal{F}_{\text{dyn}}(\sigma) = 0$ and $\mathcal{H}_{\text{dyn}}(\sigma) \leq 0$. These constraints correspond to ensuring that the trajectory satisfy a set of differential equations (a unicycle model), that derivatives are bounded (e.g. velocity, bank angle) and the configuration of the robot is within acceptable regimes according to specified performance charts (e.g. $\dot{z}(t)$ is bounded according to torque limits).

2. *Route constraints*: We will abstract these as $\mathcal{F}_{\text{route}}(\sigma) = 0$ and $\mathcal{H}_{\text{route}}(\sigma) \leq 0$. These correspond to ensuring the trajectory satisfies boundary constraints and stays within the flight corridor.

3. *Safety constraints*: We will abstract these as $\sigma(t) \in \Sigma_{\text{valid}}$. This corresponds to the trajectory being sufficiently clear of obstacles and no-fly-zones.

9.2.3 Objective

The objective function $J(\sigma)$ that the planner tries to minimize is the total traversal time, i.e $J(\sigma) = t_f$

9.2.4 Trajectory planning problem

Combining these constraints and objective, we have the following the trajectory planning problem

Problem 13 (Trajectory Planning Problem)**.** The trajectory planning problem is then formally defined as the search for the trajectory, σ^*, that minimizes a given cost function, while satisfying boundary and trajectory wide constraints

$$\begin{aligned} \min_{\sigma \in \Sigma} \quad & J(\sigma) \\ \text{s.t} \quad & \sigma(0) = s \\ & \sigma(t_f) = g \\ & \mathcal{F}(\sigma) = 0 \\ & \mathcal{H}(\sigma) \leq 0 \\ & \sigma(t) \in \Sigma_{\text{valid}} \\ & \forall t \in [0,\ t_f] \end{aligned} \tag{9.1}$$

9.3 Trajectory planning architecture

We now present an overview of our approach. The key idea is to decompose the overall problem into 3 cascaded sub-problems, each of which can be solved in a tractable fashion. Each of these sub-problems operate at different resolutions, different time-periods and focus on different aspects of the optimization problem.

Fig. 9.3 illustrates the 3 stage decomposition. We highlight each stage of the decomposition, explaining the rationale along the way

1. *Route guide trajectory* : The first sub-problem involves processing the entire route and dynamics constraints and solving for a global trajectory which we call the *route-guide*. This is the most computationally expensive operation as it has to reason about the mission as a whole and compute not only an overall shape that is feasible but also a velocity profile. However, since this procedure has to be done once a mission, it is given a time budget of 20 seconds to compute such a trajectory. Existence of a solution implies that under nominal conditions, the mission is feasible. In other words, if there were no obstacles or no-fly-zones, the ideal trajectory to execute would be this trajectory.

2. *Receding horizon trajectory* : The second sub-problem is concerned with obstacle and no-fly-zone avoidance. Since avoidance must be done in real-time, such a module has to run at a higher frequency (1 Hz). Hence such a module only computes a trajectory up to a finite

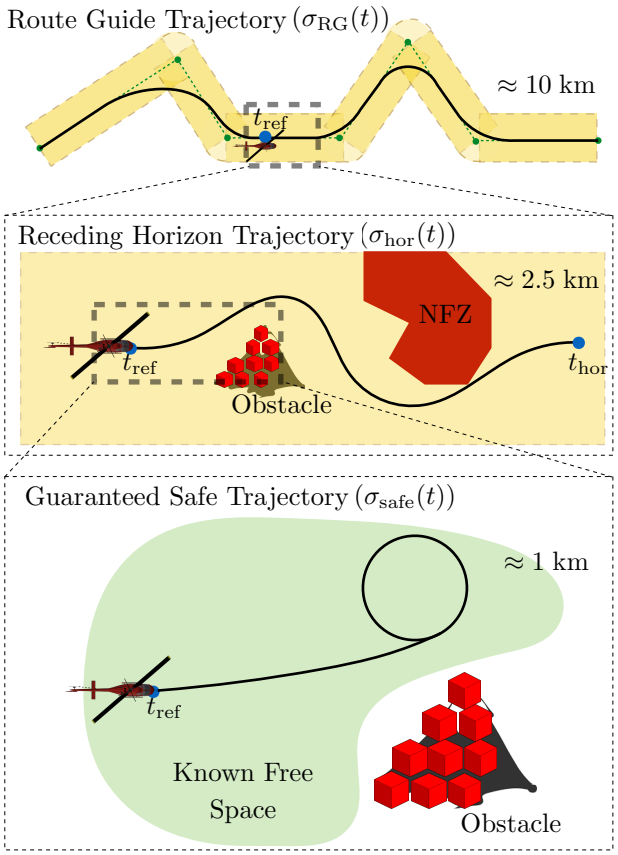


Figure 9.3: The 3 stage decomposition of the trajectory planning problem. The decomposition is illustrated for an autonomous helicopter application. The first stage takes as input the entire route and dynamics constraints and computes a route guide trajectory $\sigma_{\mathrm{rg}}(t)$. The trajectory is ensured to be dynamically feasible and lie in the safe flight corridor. The second stage takes as input the route guide trajectory and obstacle constraints and repairs the trajectory locally upto a finite horizon to produce $\sigma_{\mathrm{hor}}(t)$. The horizon is typically a few kilometers. The final stage takes as input the computed trajectory and ensures that it is guaranteed to be safe. It does this by ensuring that there exists an evasive maneuver that lies in known free space. It then appends this evasive maneuver and to the end of the trajectory to produce a safe trajectory $\sigma_{\mathrm{safe}}(t)$. Since the system re-plans, the evasive maneuver is never really executed unless the motion planner enters a scenario where it is unable to replan for a feasible path.

horizon to alleviate computational burden. The horizon is set to be double the sensor range to ensure that the system only reasons about obstacles that it has seen and no-fly-zones in the vicinity. Under nominal situations the receding horizon trajectory simply imitates the route guide trajectory. When obstacles appear, this trajectory avoids obstacles within the horizon while eventually connecting back to the route guide trajectory.

3. *Guaranteed safe trajectory*: The third sub-problem is concerned with guaranteeing safety. As the robot flies in a partially known world, it discovers obstacles on the fly. Even though the previous sub-problem is tasked with computing avoidance maneuvers, it is often the case that no such maneuvers exist. Hence, a safety module has to ensure that robot can perpetually exist in a set of states that will never be in collision. Safety can be guaranteed

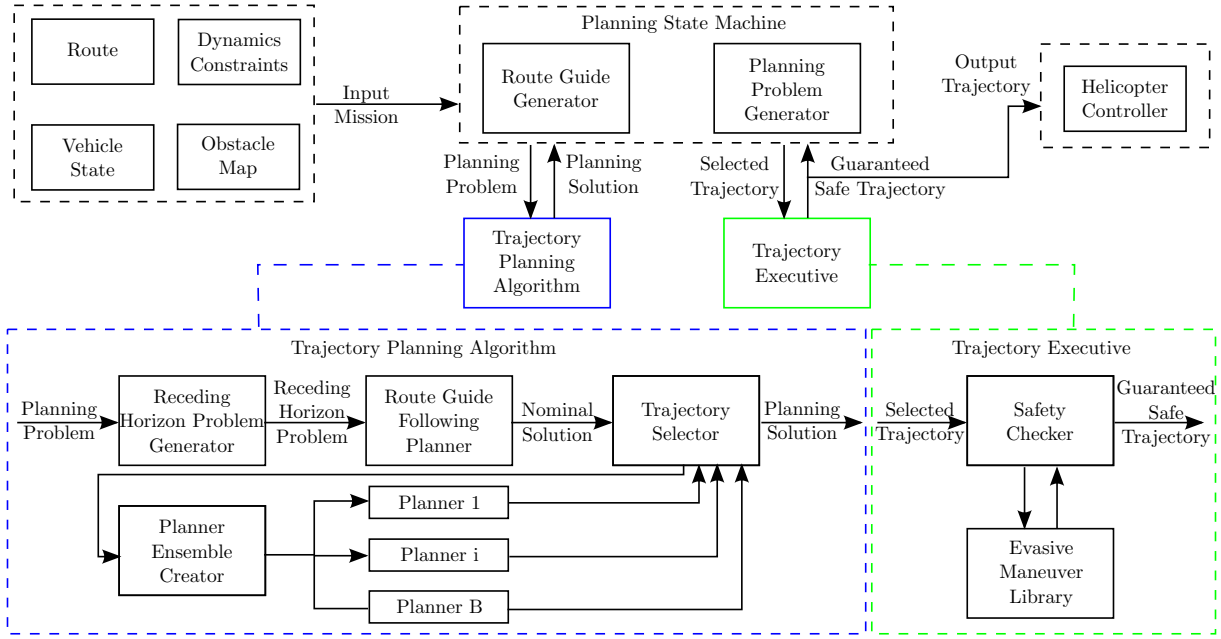


Figure 9.4: The trajectory planner block diagram. It consists of 3 main modules - the planning state machine, the trajectory planning algorithm and the trajectory executive.

by checking if an evasive maneuver can be attached to the planned trajectory. The evasive maneuver is a trajectory that can perpetually keep the robot in known free space. Hence this module has to reason at the sensor range, but must do so at a much higher frequency (10 Hz).

Each of these 3 decompositions are handled by 3 modules in the trajectory planner. The block-diagram of our trajectory planning architecture is shown in Fig. 9.4. We will now briefly describe these modules.

9.3.1 Module 1: Planning state machine

The role of the state machine is to process the input mission and create a planning problem for the trajectory planning module to solve. It does so by invoking two sub-modules - the *route guide generator* and the *planning problem creator*.

Route guide generator

A *route guide* is a complete trajectory from start to goal of the mission that respects the dynamics constraints and the route constraints. Under nominal conditions, i.e. in the absence of obstacles or no-fly-zones, this trajectory is the optimal desired trajectory we wish to follow. Computing the route guide is the first step in our decoupling based approach. Once the route guide is computed, the trajectory planning modules need only be concerned with following the route-guide as closely as possible. Fig. 9.3 shows an example of a route guide trajectory generated from a mission.

The route guide is computed by a *route guide generator* once per mission. This implies that the route guide generator can take much longer than a normal trajectory planner (up to 20 s). However, computing a route guide is a difficult non-convex multi-scale optimization problem. We will discuss how we solve such an optimization problem in Section 9.4.

Planning Problem Generator

A *planning problem* is a complete list of specifications sent to the trajectory planning module. It is defined by the following tuple $\Gamma_{\text{pp}} = (s, g, \mathcal{F}, \mathcal{H}, \Sigma_{\text{valid}}, \sigma_{\text{rg}})$. We describe each component briefly

1. s - The start pose and velocity that the trajectory must satisfy
2. g - The terminal touchdown point (set of touchdown points) coming from the mission specification
3. $\mathcal{F}$ and $\mathcal{H}$ - The set of equality and inequality constraints as defined in Section 9.2.
4. Σ_{valid} - The set of valid states.
5. σ_{rg} - The route guide trajectory

The *planning problem generator* prepares this tuple and sends it along to the trajectory planning algorithm.

9.3.2 Module 2: Trajectory planning algorithm

The *trajectory planning algorithm* module takes a planning problem as input. It then tries to follow the route guide trajectory upto a horizon. If such a solution violates safety constraints, it tries to compute a feasible trajectory upto the finite horizon that satisfies all constraints including safety constraints such as obstacle avoidance and no-fly-zone avoidance. Fig. 9.3 shows an example of a receding horizon trajectory σ_{hor}.

This module consists of two sub-modules: *receding horizon problem generator* which generates receding horizon problems and *receding horizon planning* which is tasked with computing the trajectory.

Receding horizon problem generator

This sub-module takes the whole planning problem as input and decides a horizon to plan to. It then uses this horizon to splice the route guide trajectory. This spliced route guide trajectory is then used to create a new receding horizon planning problem. Algorithm 18 describes this process.

Receding horizon planning

This submodule takes as input the receding horizon planning problem and produces a trajectory till the horizon. The general approach is as follows: the module computes a nominal trajectory

Algorithm 18: CreateRHPP$(\Gamma_{\text{pp}}, t_{\text{hor}})$

$(s, g, \mathcal{F}, \mathcal{H}, \Sigma_{\text{valid}}, \sigma_{\text{rg}}) \leftarrow$ `Unpack` (Γ_{pp});
$t_{\text{proj}} \leftarrow$ `Project` (s, σ_{rg});
$t^{+}_{\text{hor}} \leftarrow$ `Propagate` $(\sigma_{\text{rg}}, t_{\text{proj}}, \delta_{\text{hor}})$;
if $t^{+}_{\text{hor}} \geq \min\left(t_{\text{hor}} + \varepsilon_{\text{hor}}, t_f\left(\sigma_{\text{rg}}\right)\right)$ **then**
  $t_{\text{hor}} \leftarrow t^{+}_{\text{hor}}$;
$\sigma^{+}_{\text{rg}} \leftarrow$ `Chop` $(\sigma_{\text{rg}}, t_{\text{proj}}, t_{\text{hor}})$;
$\Gamma_{\text{rhpp}} \leftarrow \left(s, g, \mathcal{F}, \mathcal{H}, \Sigma_{\text{valid}}, \sigma^{+}_{\text{rg}}\right)$;
return $(\Gamma_{\text{rhpp}}, t_{\text{hor}})$;

$\sigma_{\text{nom}}(t)$ that follows the route guide. If such a trajectory satisfies all constraints (including safety), it is selected as a solution. If $\sigma_{\text{nom}}(t)$ is invalid, trajectory planning algorithms are invoked.

The approach to solving such problems is the adaptive ensemble approach described in Chapter 5. The goal of the planning algorithms is to plan minimal time paths to the horizon point while satisfying constraints. A trajectory selector then selects the best trajectory returned by the ensemble. Algorithm 19 illustrates the trajectory selection algorithm.

Algorithm 19: TrajectorySelector$(\Gamma_{\text{rhpp}}, \sigma_{\text{nom}}, \Sigma_{\text{ens}}, \sigma_{\text{sol}})$

$(s, g, \mathcal{F}, \mathcal{H}, \Sigma_{\text{valid}}, \sigma_{\text{rg}}) \leftarrow$ `Unpack` (Γ_{rhpp});
if $\mathcal{F}\left(\sigma_{\text{nom}}\right) = 0$ **and** $\mathcal{H}\left(\sigma_{\text{nom}}\right) \leq 0$ **then**
  $\sigma_{\text{sol}} \leftarrow \sigma_{\text{nom}}$;
else
  $\Sigma_{\text{ens}} \leftarrow \Sigma_{\text{ens}} \bigcup \sigma_{\text{sol}}$;
  $\Sigma^{+}_{\text{ens}} \leftarrow$ `FilterIllegal` $(\Sigma_{\text{ens}}, \Gamma_{\text{rhpp}})$;
  $\sigma_{\text{sol}} \leftarrow \underset{\sigma \in \Sigma^{+}_{\text{ens}}}{\arg\min} J\left(\sigma\right)$ s.t $\mathcal{F}\left(\sigma\right) = 0, \mathcal{H}\left(\sigma\right) \leq 0$;
return σ_{sol};

9.3.3 Module 3: Trajectory executive

The role of this module is to guarantee safety of the system. It does so by taking as input the finite horizon trajectory computed by the planning algorithm and checking whether it is a *guaranteed safe trajectory*. It does so by ensuring that there exists an evasive (or emergency) maneuver that originates from the trajectory. An evasive maneuver is an infinite horizon trajectory that lies in the known free state space. Usually such trajectories are loiter patterns (for helicopters) or hover commands (for quadrotors). Fig. 9.3 shows a guaranteed safe trajectory $\sigma_{\text{safe}}(t)$. It initially is identical to $\sigma_{\text{hor}}(t)$, but as it approaches the sensor horizon, it diverges to form a loiter pattern. Note that as long as the planner keeps re-planning safe trajectories, the system will never execute the evasive maneuver portion of the trajectory. At this junction, a pertinent concern is whether such a method of decoupling leads to deadlocks. To answer this question, we examine the interaction between nested stages.

1. *Deadlock between route guide and finite horizon trajectory*: The only scenario when a deadlock can potentially occur is when an obstacle appears such that no finite horizon

trajectory exists. We overcome this scenario by ensuring that the goal point on the route guide is always in a feasible location sufficiently far away from obstacles. Hence as long as the flight corridors are wide enough, the trajectory planners should be able to find a path.

2. *Deadlock between finite horizon trajectory and guaranteed safe trajectory*: The only scenario where a deadlock occurs is where the trajectory planner produces a solution to which no evasive maneuvers can be attached. The executive alleviates the situation by slowing down the trajectory. If that does not resolve the deadlock, the system enters the evasive maneuver and the sensor keeps on scanning and increasing the known free space. This increases the chance of the planned trajectory being verified to be safe.

9.4 Global planning via route guide generation

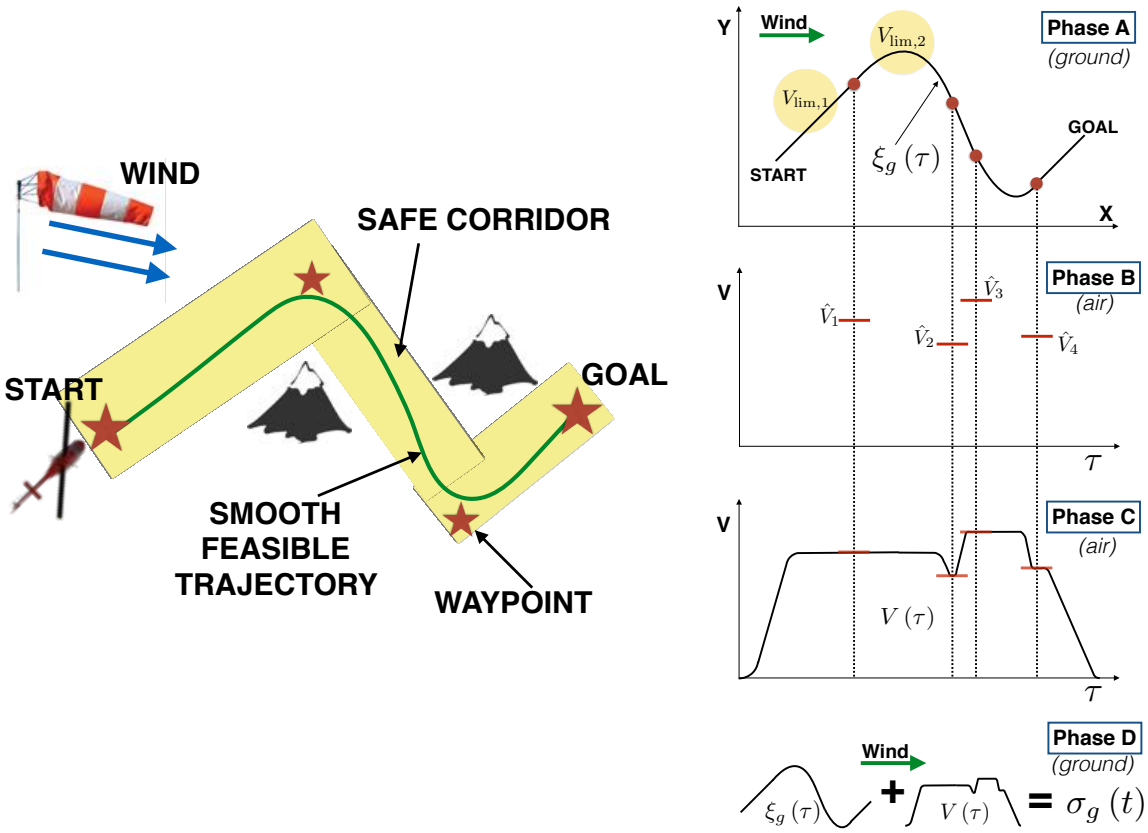


Figure 9.5: Overview of the route guide generation algorithm. The algorithm takes as input a route - a set of waypoints (red star), a set of flight corridors (yellow rectangle). It also takes as input dynamic constraints and a wind vector. It then solves for a feasible time optimal trajectory with a four stage decomposition. Phase A solves for a path in ground frame. Phase B solves for limits on the airspeed at certain keyframes. Phase C solves for an airpseed that satisfies these limits. Phase D combines the path and velocity profile to find a trajectory in ground frame.

We briefly describe the procedure for creating a route guide trajectory. The route guide

trajectory is concerned with satisfying the dynamics constraints and the route constraints while ignoring the obstacle and safety constraints. Hence the trimmed down optimization problem is as follows

$$\begin{aligned} \min_{\sigma} \quad & J(\sigma) \\ \text{s.t} \quad & \sigma(0) = s \\ & \sigma(t_f) = g \\ & \mathcal{F}_{\text{route}}(\sigma) = 0 \\ & \mathcal{H}_{\text{route}}(\sigma) \leq 0 \\ & \mathcal{F}_{\text{dyn}}(\sigma) = 0 \\ & \mathcal{H}_{\text{dyn}}(\sigma) \leq 0 \end{aligned} \tag{9.2}$$

A challenging aspect of this problem is the interplay between the route constraints and the dynamic constraints that depend on the wind. Consider the problems faced when planning in a moving reference frame. Planning a dynamically feasible path in this frame results in a drifting ground frame path that might violate the safe flight corridor. On the other hand, if planning is done in the ground frame, the dynamics constraints are no longer stationary and varies along the path. In addition to wind, the other main challenges are satisfying nonholonomic constraints due to vehicle dynamics and scaling to large distances. This results in a complex multi-resolution non-convex planning problem.

To make the solution-search tractable, we decouple the optimization problem into a path optimization and a velocity profile optimization [Bobrow et al., 1985, Choset, 2005, hwan Jeon et al., 2013]). The optimization approach, summarized in Fig 9.5 proceeds in 4 stages:

1. Phase A: *Path Optimization*. This phase solves for a path that is *guaranteed to be feasible* (in terms of dynamics and route constraints) for a range of constrained velocity profiles. The path is parameterized as a sequence of *sections* which are either straight lines or arcs. The optimizer solves for each section i independently along with a corresponding V_i, such that the section is feasible for any velocity profile limited by V_i. In the interest of time-optimality, the objective of this optimizer is to maximize the velocity limit V_i while keeping the total arclength of the section small.

2. Phase B: *Velocity Optimization*. This phase optimizes velocity at specific *control points* at the end of the sections to minimize time. By ignoring jerk constraints at this stage and assuming a trapezoidal velocity profile, we are able to solve this optimization very efficiently.

3. Phase C: *Velocity Spline Fitting*. This phase solves for smooth velocity splines that introduce jerk limits.

4. Phase D: *Ground Frame Trajectory Repair*. This phase combines the path from Phase A with the velocity profile from Phase C to yield the final ground frame trajectory.

We refer the reader to Dugar et al. [2017a] and Dugar et al. [2017b] for details on this approach.

9.5 Local planning via an adaptive ensemble

9.5.1 Overview

The goal of the receding horizon trajectory planning algorithm is to solve the original trajectory planning problem defined in Problem 9.1 upto a finite horizon. The horizon is then receded and the module requested to replan. To alleviate myopic shortcomings due to reasoning in a finite horizon, the planner is tasked with trying to follow the route guide closely. Hence in the absence of obstacles, the trajectory planning algorithm employs a nominal planning approach that minimizes deviation from the route guide trajectory.

When the nominal trajectory is infeasible, the planner has to compute an *avoidance maneuver*. To solve such avoidance maneuvers tractably in real-time, we employ the expert planner approach described in Chapter 4. This allows us to import the black-box adaptive planner framework from Chapter 5.

9.5.2 Solving nominal planning problems

We define the nominal planning problem as minimizing the deviation from the route guide trajectory upto a horizon. This can be expressed as the following optimization problem.

$$\begin{aligned} \min_{\sigma} \quad & \int_0^{t_f} ||\sigma(t) - \sigma_{\text{rg}}(t)|| \, \text{dt} \\ \text{s.t} \quad & s = (\sigma(0), \dot{\sigma}(0), \ddot{\sigma}(0)) \\ & ||\sigma(t_f) - \sigma_{\text{rg}}(t_f(\sigma_{\text{rg}})||_2 \leq \varepsilon_{\text{tol}} \\ & \mathcal{F}_{\text{dyn}}(\sigma) = 0 \\ & \mathcal{H}_{\text{dyn}}(\sigma) \leq 0 \end{aligned} \tag{9.3}$$

This optimization problem can be solved by the dynamics projection operator as described in Appendix B.

9.5.3 Solving avoidance planning problems

When the solution to the nominal planning problem (9.3) violates safety constraints, an *avoidance planning problem* is created. For these problems, it is no longer beneficial to minimize deviation from the route guide. Instead the planners are asked to compute time optimal trajectories to shortcut to the finite horizon goal as fast as possible. This leads to the following

optimization problem.

$$\begin{aligned} \min_{\sigma} \quad & J(\sigma(t)) \\ \text{s.t} \quad & s = (\sigma(0), \dot{\sigma}(0), \ddot{\sigma}(0)) \\ & ||\sigma(t_f) - g||_2 \leq \varepsilon_{\text{tol}} \\ & \mathcal{F}(\sigma) = 0 \\ & \mathcal{H}(\sigma) \leq 0 \\ & \sigma(t) \in \Sigma_{\text{valid}} \\ & \forall t \in [0,\ t_f] \end{aligned} \tag{9.4}$$

The problem described in (9.4) now follows the template of the kinodynamic planning problem described in Chapter 4, (4.1). This allows us to employ the ensemble of expert planner approach.

We use the exact framework described in Chapter 5. We have a library of expert planners for UAVs which remains the same irrespective of the UAV platform. The dynamics projection filter that is applied to project a path into a feasible trajectory changes with different UAVs in order to respect the dynamics. The meta-planner extracts context such as the speed at which the system is moving, the density of obstacles at various locations to decide which ensemble to choose. Details on the library of planners are described in Appendix C. Details on the feature extraction is described in Appendix D.

9.6 Guaranteeing safety via trajectory executive

We briefly describe the verification process for ensuring a trajectory is guaranteed safe. A trajectory is deemed to be guaranteed safe if there exists at least one evasive / emergency maneuver that can be executed from the trajectory. An evasive maneuver is a trajectory that is safe for an infinite time horizon. To satisfy this criteria, the evasive maneuver must completely come to a stop inside *known free space*. It can also enter a periodic motion (such as a fixed loiter pattern) inside the space. The known free space is constructed online as the sensor scans the world. Hence, safety must be checked for online.

The trajectory executive module has a sub-module - the evasive maneuver library. The library comprises of a diverse set of evasive maneuvers. The details for constructing such a set is described in Arora et al. [2015]. If the library is queried with an input candidate trajectory, it cycles through the maneuvers, and attempts to find a connection point to the main trajectory such that the maneuver lies in known free space. It tries to find the connection point as far in the future as possible.

The other submodule in the executive is the safety checker which is illustrated in Fig. 9.6. The safety checker is tasked with the decision making aspect of the executive. It takes as input a candidate trajectory from the trajectory planning module. It queries the library to find a maneuver for the trajectory. If such a maneuver exists, the trajectory is deemed safe. The maneuver is stitched to the trajectory and the new trajectory is sent to the controller. This is to ensure that if the planning module were to suddenly terminate, the vehicle does not execute

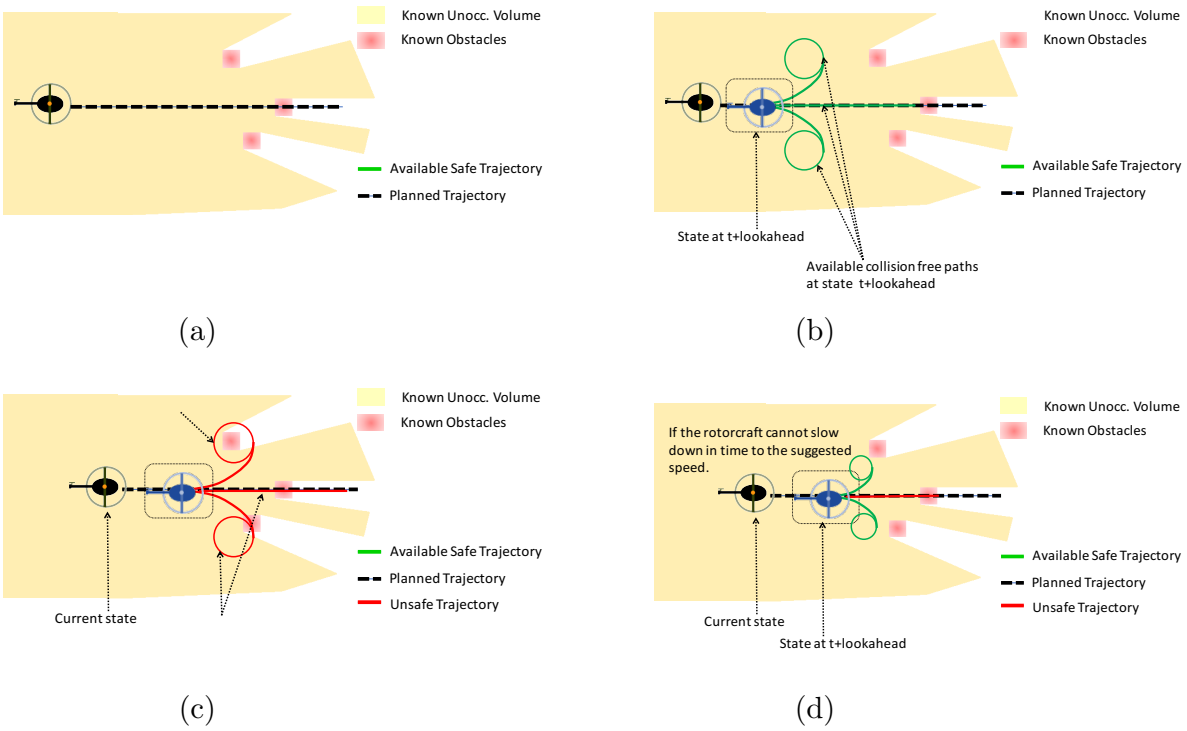


Figure 9.6: The executive ensures trajectories are guaranteed safe. (a) The motion planner sends a planned trajectory to the executive (b) The safety checker queries an emergency maneuver library to ensure maneuvers exist from the trajectory that lie entirely in known free space. The maneuver is attached to the trajectory and the guaranteed safe trajectory is passed to the control system (c) Eventually the safety checker encounters a case where the library is unable to find a maneuver for the trajectory (d) The safety checker tries to slow down the vehicle until the library is successful. If not, the safety checker reverts to the previous guaranteed safe trajectory.

an unsafe motion.

If the library was unable to find a maneuver, the safety checker tries to slow down the trajectory and re-query the library. Often, this is successful since slowing down the system allows it to make sharper turns and find loiter patterns that are in known free space. In the event that the candidate trajectory cannot be transformed to be guaranteed safe, the executive reverts to the previous safe trajectory which should exist by construction. If the planner is unable to produce a feasible trajectory on re-planning, the system eventually executes the evasive maneuver portion of the trajectory.

9.7 Flight test summary

The primary application that we focused on was autonomous cargo delivery by a full scale helicopter [Choudhury et al., 2014]. The goal of the project was to develop a complete autonomy system that enabled the system to fly long missions from take-off to touchdown, avoid obstacles, fly smoothly like a pilot and ensure safety at all times. The trajectory architecture evolved to satisfy these diverse set of needs. We additionally validated this architecture on two other systems

	Full-scale helicopter	**Large hexarotor**	**Small quadrotor**
Flight hours	$>$ 700 hours	$>$ 6 hours	$>$ 6 hours
Top speed	60 m/s	10 m/s	5 m/s
Max distance flown in 1 mission	$>$ 48 km	$>$1 km	$>$ 0.5 km
Sensing horizon	≈1000 m	≈100 m	≈ 15 m
Planning framework	Hand designed ensemble Learnt static ensemble	Hand designed ensemble Learnt dynamic ensemble	Hand designed ensemble Learnt dynamic ensemble
Obstacles avoided	Mountains, towers, trees, wires, no-fly-zones	Pylons, shipping containers	Mound, trees, ridges, no-fly-zones

Figure 9.7: Summary of evaluation of planning architecture on 3 distinct UAVs

- a large hexarotor, and a small quadrotor - to show the generalizability of the architecture and planners. Fig. 9.7 provides a summary of the flight tests.

We now highlight results to shed a spotlight on the contribution of different elements of the architecture.

9.7.1 Case study A: Obstacle avoidance in different environments

Fig. 9.8 shows a sample of the different kinds of environments the planning algorithm had to operate in. Each of these scenarios favoured different expert planners, thus justifying the need for an ensemble of planners.

Fig. 9.8(a) shows a scenario where the UAV had to fly between a no-fly-zone and a mountain. This created a narrow gap which was solved by `RRT*Tunnel`[1]. Fig. 9.8(b) shows a scenario where the helicopter has to fly around a tower with sufficient clearance. `CHOMP`[1] is able to solve this problem by smoothly optimizing the trajectory around the tower. Fig. 9.8(c) shows planning in an urban environment. `A*`[1] does well on this problem since long edges will generally be in collision. Fig. 9.8(d) shows the UAV avoiding a power-line. This was solved by a precision planner `SingleDetour`[3], which samples points vertically around the wire to find a path.

9.7.2 Case study B: Scaling to long missions

Fig. 9.9 shows how the system can scale to a long mission. The shown mission is $48km$ long. There were occasional no-fly-zones along the way invalidating the route. Because of the decoupling scheme of our architecture, the system was able to solve the mission. The global route guide reasoned about velocity profiles at different legs of the mission and computed a route guide. The planners subsequently followed the route guide whilst avoiding any no fly zones that cropped up. It is important to note that in absence of the decoupling, solving this mission in one shot would have been computationally intractable.

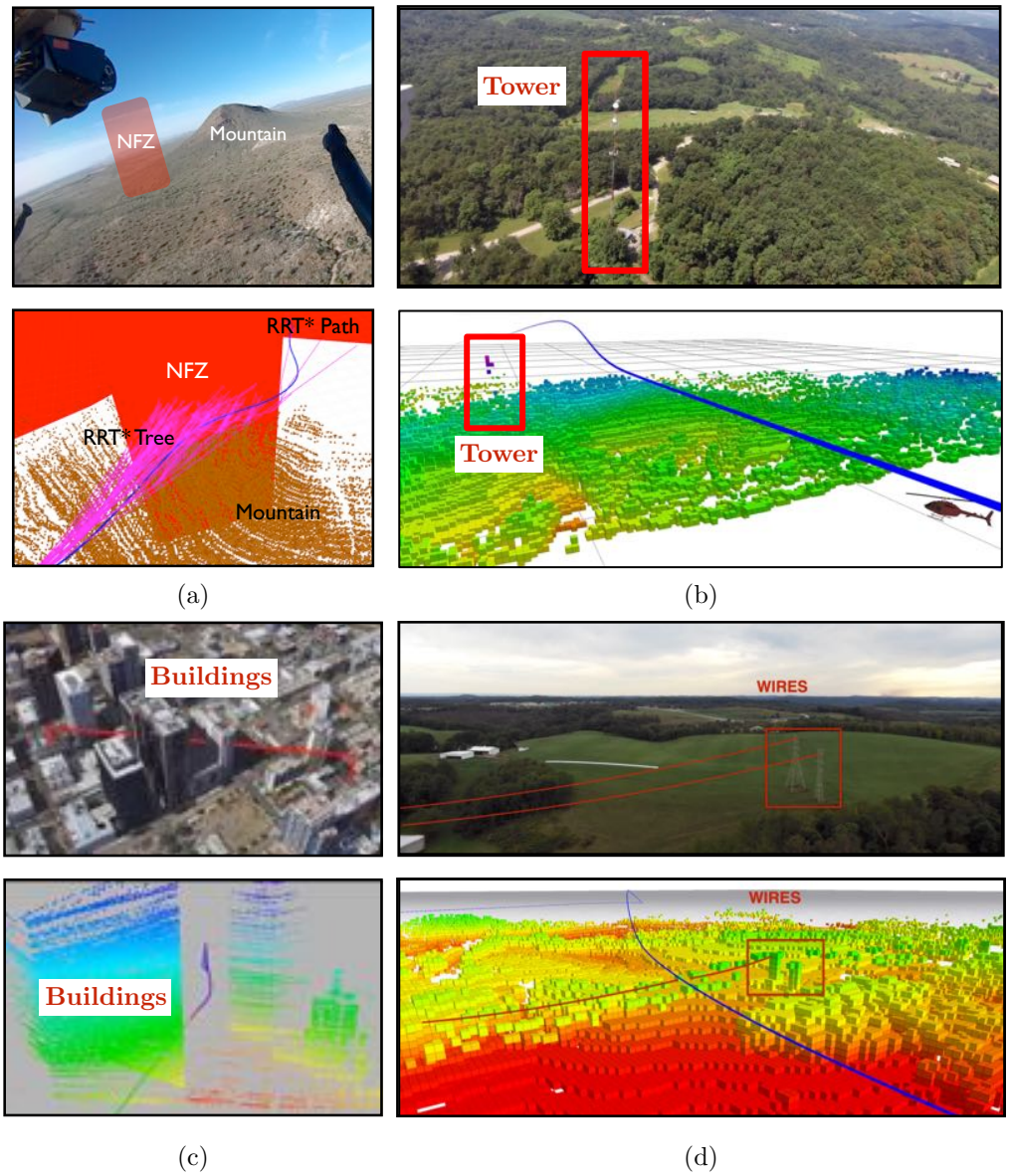


Figure 9.8: Obstacle avoidance in different environment (a) Mountains (b) Tower (c) Urban (d) Power-lines.

9.7.3 Case study C: Interaction between safety and planning

Fig. 9.10 shows how the system guarantees safety when the motion planner is unable to solve the problem. Fig. 9.10 (a) shows the evasive maneuver library. Fig. 9.10 (b) shows the planned trajectory by the motion planner (blue). The safety checker is able to find multiple evasive maneuvers (orange). Moreover, it attaches an evasive maneuver at the end of the trajectory to ensure its guaranteed safe. Fig. 9.10 (c) A popup no-fly-zone appears very near the vehicle. The planner is unable to compute a path. As a result the system eventually executes the evasive maneuver part of the current trajectory. (d) The evasive maneuver moves the system to a state

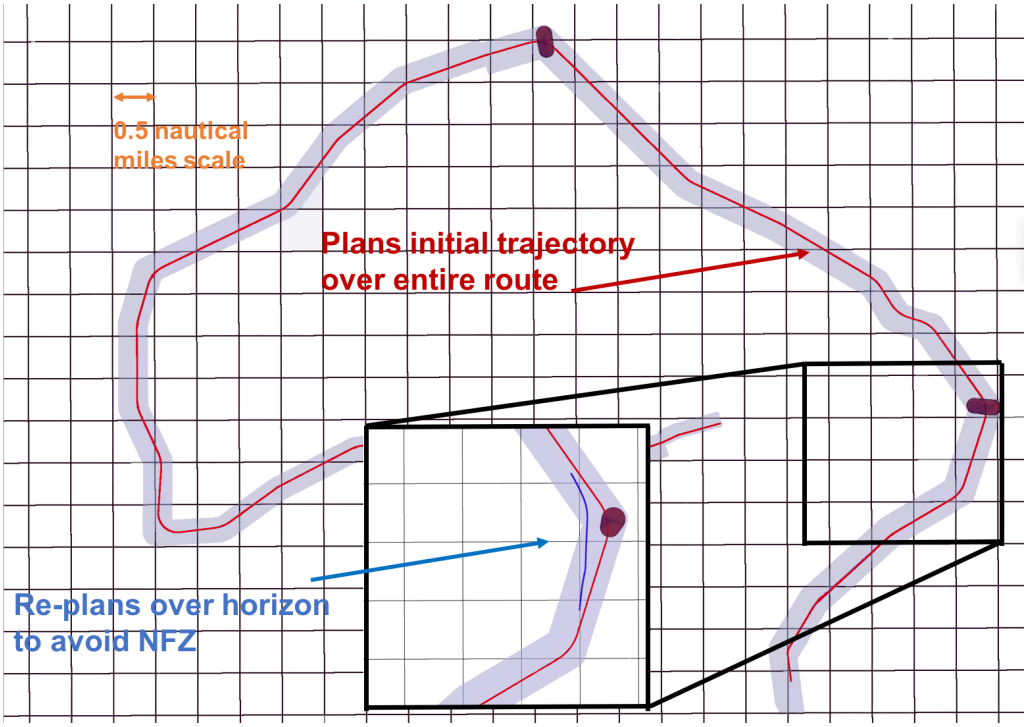


Figure 9.9: Executing a mission 48 km long. The helicopter is also able to avoid obstacles along the way.

from where the planner can find a solution.

9.7.4 Case study D: Generalization across vehicles

Fig. 9.11 shows that the same planning library generalizes across the 3 UAVs on which we tested. The things that changed from system to system are the following

1. The dynamics model and constraints vary
2. The clearance from obstacle vary
3. The route guide generation remains the same as it takes the constraints as input
4. In the expert planner architecture, the DPF changes from smaller vehicles to the helicopter. However, the planner library specified in Appendix C.
5. The feature extraction changes from system to system

Hence we are able to make significant progress towards our goal of data-driven planning by keeping the underlying software layer same, but training new meta-planners for different UAVs / applications.

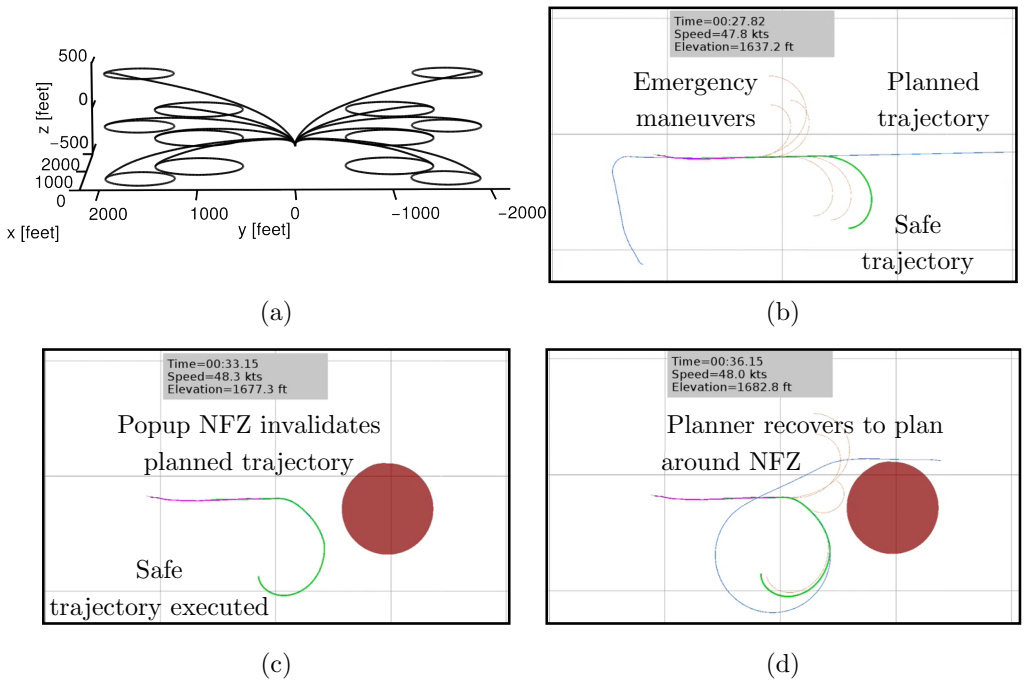


Figure 9.10: Guaranteed safe architecture in operation (a) Evasive maneuver library (b) The planner plans a nominal trajectory which is checked to be safe (c) A no-fly-zone appears, and the safety system rejects the old plan. System eventually executes the evasive maneuver (d) Planner recovers and nominal behaviour resumes.

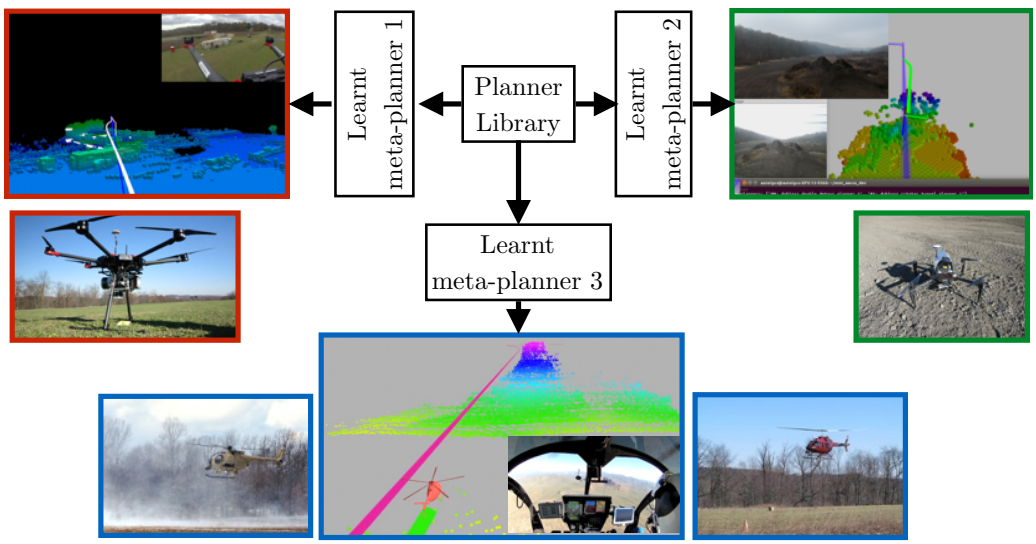


Figure 9.11: Generalization of the same planning software to multiple vehicles.

10

Conclusion

This thesis argues for the proposition that a motion planning system for a mobile robot needs to adapt to the distribution of problems the robot actually encounters to find good solutions within a fixed time budget. The proposed approach builds upon a key observation that planners can employ different search strategies that prioritize searching different regions of the solution space to find such solutions quickly. However, the mapping from the configuration of obstacles in the environment to the effectiveness of a search strategy is not known and can be quite complex. Learning can play an important role here. This thesis presents a framework for learning such mappings. We present results where the proposed framework not only outperforms baseline non-adaptive systems on a wide range of planning datasets, but also enables UAVs to navigate robustly for long durations across different problem instances.

We begin by summarizing the arguments, the algorithmic contributions and the findings in Section 10.1. We present future avenues for research in Section 10.2 that build on the novel connections and observations that this thesis makes.Finally, we offer some concluding remarks in Section 10.3.

10.1 Summary and contributions

To address the problem of adaptive motion planning, this thesis proposes examining three principle challenges. These challenges are arranged in order of increasing adaptivity. We propose a collection of frameworks and algorithms to address each of these challenges. Finally, we also summarize how the developed algorithms work in practice as part of an overall planning architecture for UAVs. Fig. 10.1 shows how the developed algorithms can be unified in a framework.

10.1.1 Challenge 1: Designing effective planning modules

Challenge 1 considers the design of a motion planning system that meets performance requirements for a fixed distribution of planning problems known to a human engineer at design time.

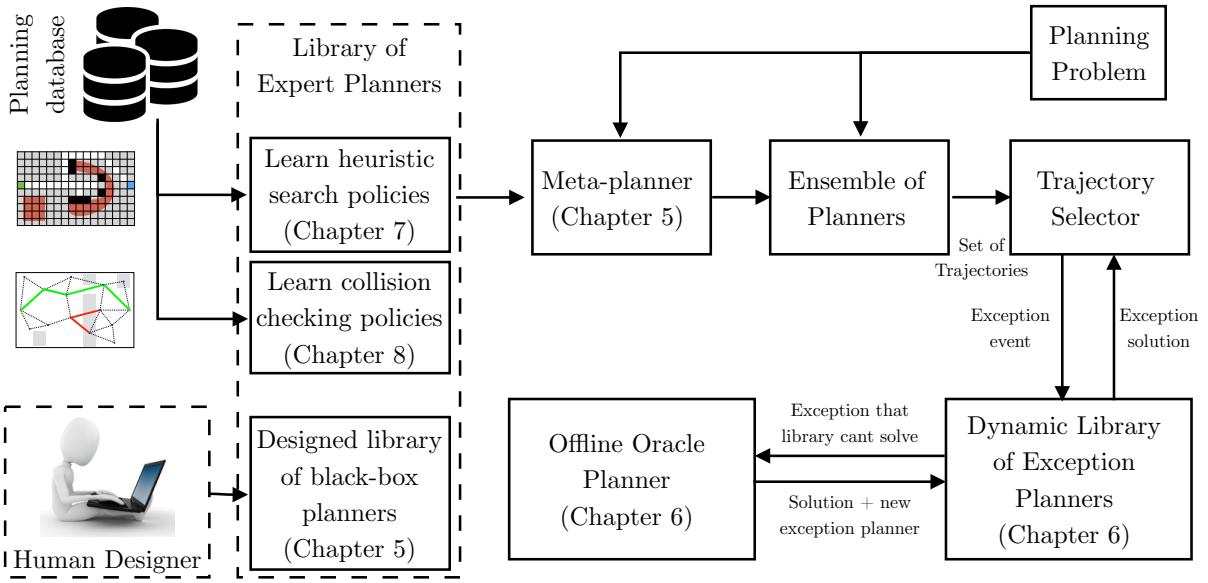


Figure 10.1: A unified framework for adaptive motion planning that uses all algorithms developed in this thesis. We start with a planning problem database. This database can be used to train heuristic search policies (Chapter 7). We can also train edge evaluation policies that operate on a roadmap (Chapter 8). We can train both these policies on subsets of the planning problem database to get different planners. These trained planners can be collected to form a library of expert planners. The human designer can also choose to populate the library with hand-designed planners using techniques described in Chapter 4. We can now train a meta-planner that maps context extracted from the problem to select from the library of planners using techniques described in Chapter 5. Each planner in the ensemble computes a plan. A trajectory selector selects the best plan. If none of the planners solves the problem, the selector queries a library of exception planner to solve the problem. Such a library is created on the fly by querying an oracle for a solution everytime none of the selected planners or library can solve the problem. This is done using techniques presented in Chapter 6.

In contrast to conventional motion planning research which analyzes the worst case complexity of different planning strategies, this complementary line of inquiry seeks to understand the inverse mapping from planning problems to suitable planning strategies.

In Chapter 3, we approached this problem by proposing a framework for assembling path planning algorithms. We showed how we can design implicit graphs to contain edges in regions where we expect a good solution to be found.

In Chapter 4, we employed such a framework to compile an expert planner for a distribution of planning problems. We showed how one can end up with two complimentary planner designs - a general purpose planner that performs reasonably on a lot of problems or a precision planner that performs near-optimally on a small number of problems. We found it hard to design a single threaded planning module on account of the unpredictability of the performance of precision planners. This motivated us to hedge our bests and design an ensemble of expert planners. We proposed a greedy planner design strategy to create an effective ensemble containing planners from both classes.

10.1.2 Challenge 2: Learning for black-box meta-planners

Challenge 2 considers the case when the distribution of planning problems can vary with different missions. The human designer cannot redesign the planning system - the module must exhibit some form of adaptation. To alleviate the burden on the learning component, we assume the learner has access to a good library of planners. To generalize the framework, we assume each of these planners are black-box atomic operations that map a problem to a solution. This challenge requires learning a meta-planner that can select planners from this library to maximize performance on the distribution of problems encountered.

In Chapter 5, we proposed a framework to train a meta-planner to select an ensemble from a library of black-box planners. We proposed a lazy greedy algorithm that leverages trained priors on planner performance to select a static ensemble for a distribution without evaluating every planner. However, there are scenarios where a static ensemble does not suffice. For such settings, we present a framework to greedily train a list of contextual predictors to select an ensemble. We show that such a procedure not only has theoretical guarantees, but leads to an intuitive procedure where each predictor focuses on solving failure cases of the preceding predictors.

In Chapter 6, we examine the online setting where we want a meta-planner to dynamically construct a library of black-box planners created on-the-fly. Such planners are created when an exception event occurs - an event where the motion planner fails to find a solution, the robot comes to a stop and an offline oracle provides an exception planner to solve the problem. We show that the problem of deciding which exception planner to retain is equivalent to the problem of online paging. We adopt a popular algorithm from online paging, Least Recently Used (LRU), to solve the problem. This framework can be used in conjunction with any motion planning system to systematically minimize the number of failure events.

10.1.3 Challenge 3: Learning for white-box adaptive planners

The black-box paradigm is restrictive - no amount of intervention or adaptation is allowed once a planner is selected. It is also computationally expensive - the meaningful context extracted from the problem is not used during planning. Challenge 3 requires the design of a white-box planner that adapts its search strategies as it gain information about the planning problem.

In Chapter 7, we addressed the problem of learning planning policies that map the sequence of decisions and outcomes to decide the next action. We showed that such problems result in large scale POMPDs. We propose a framework to efficiently train such policies via imitation learning of clairvoyant oracles. These are oracles that at train time have full access to the world and can play a good sequence of actions. We evaluate this approach on two domains - learning policies for informative path planning and learning heuristic policies for search based planning.

In Chapter 8, we addressed the problem of learning policies for adaptively evaluating edges on an expensive graph. Contrary to search heuristics, such policies have much more freedom to actively gain information. We show how this problem is equivalent to the Bayesian active learning paradigm of Decision Region Determination (DRD) where the goal is to select tests (edges) that drive uncertainty into a single decision region (a valid path). We derived efficient near-optimal algorithms, DiRECt and BiSECt to solve our problem under various setting.

Finally, in Chapter 9 we describe the overall planning architecture. We present a novel decomposition of a planning problem into low fiedlity route guide trajectory computation, mid fidelity receding horizon trajectory planning and high fidelity safety planning. We present results from 3 distinct UAV platforms where this architecture was evaluated.

10.2 Future directions

The intersection of statistical approaches and motion planning is rich with open problems. We survey a few promising directions that not only investigate improving planning performance but also more meta level questions such as when and how to plan.

10.2.1 Learning good roadmaps

We observed a common phenomenon while trying to learn a heuristic policy as well as a collision checking policy - it is impossible to learn a "perfect" policy that only evaluates the optimal path and nothing else. This is not surprising - both policies operate only on uncovered partial information. Hence they do not have enough information about the world to identify the optimal path at the very onset. However, we empirically observe that such policies eventually uncover enough information to substantially prune the solution space and narrow down the solution to one of several choices. This suggests that it is possible to probe the environment at the very onset and extract enough information to compute a good roadmap. A good roadmap is one that is very likely to contain the optimal (or bounded sub-optimal) solution while being very sparse. If such a roadmap is found, any non-adaptive planning algorithm can be employed to search the roadmap - the sparsity of the roadmap implies that in the worst case all edges can be checked within the computation budget. This suggests the possibility of a clean division of labour between planning and learning - the learning module proposes a roadmap which the planning module searches.

A gateway to the problem is to formulate a criterion for a good roadmap. Consider the following simple problem setting. We have a dense roadmap $\mathcal{G}_{\text{dense}} = (\mathcal{V}, \mathcal{E})$. We also have a database of planning problems $\Omega = \Gamma_1, \ldots, \Gamma_n$. Let $J(\mathcal{G}, \Gamma_i)$ be the cost of the optimal path in any roadmap $\mathcal{G}$ when evaluated on problem Γ_i. We now wish to frame the problem as a roadmap subset selection problem. We wish to find the sparsest roadmap $\mathcal{G} \subseteq \mathcal{G}_{\text{dense}}$ that is a subset of the dense roadmap such that it contains a $(1+\varepsilon)$ optimal path w.r.t to the dense roadmap. The size of a roadmap $|\mathcal{G}| = |\mathcal{E}|$ is the number of edges. This can be framed as the following set selection problem

$$\begin{aligned} \underset{\mathcal{G} \subseteq \mathcal{G}_{\text{dense}}}{\arg\min} \quad & |\mathcal{G}| \\ & J(\mathcal{G}, \Gamma_i) \leq (1+\varepsilon) J(\mathcal{G}_{\text{dense}}, \Gamma_i) \\ & \forall\, i = 1, \ldots, n \end{aligned} \tag{10.1}$$

We can approach the problem in (10.1) by processing each problem Γ_i and producing Ξ_i - the set of all $(1+\varepsilon)$ paths for the problem. Now (10.1) is equivalent to selecting edges from $\mathcal{G}_{\text{dense}}$ until *at least one path* from each path set Ξ_i is *covered*. While this is indeed a combinatorially

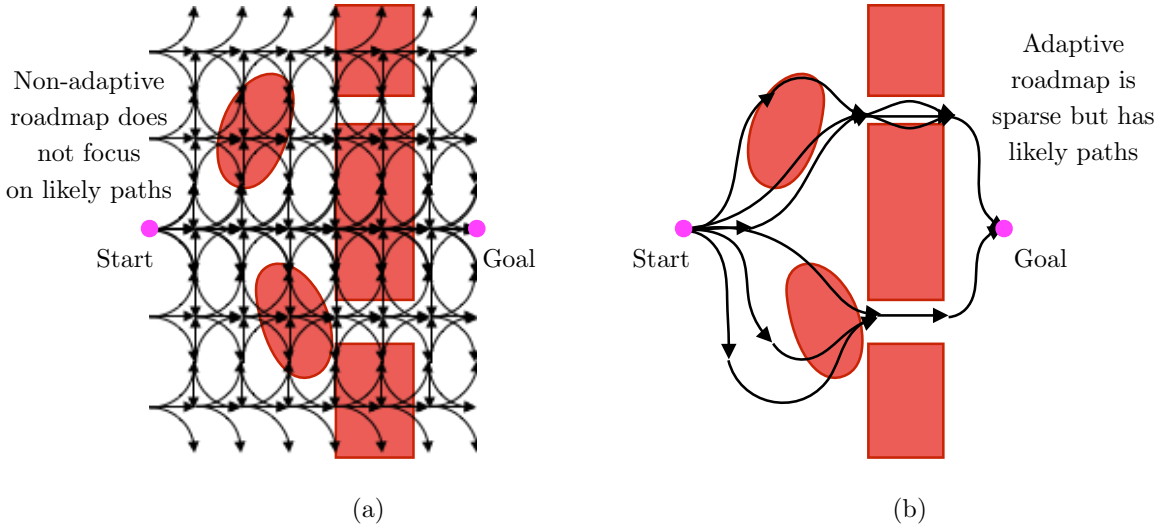


Figure 10.2: (a) A non-adaptive roadmap is agnostic to the problem instance. Hence it has to be very dense in order to ensure completeness. In this case, despite being dense the roadmap does not have a path that is feasible. (b) An adaptive roadmap that extracts a mild amount of context from the problem to focus edges in certain regions. By coarsely sampling the workspace, it only contains paths passing through the gap and circumnavigating obstacles. Note that it does not directly have to predict the optimal path. It has to simply identify a set of good candidate paths.

hard problem, we can construct surrogate objective functions that incentivize selecting edges that belong to a lot of these candidate paths (in the same manner as in Chapter 8). If such objective functions can be shown to be submodular, greedy maximization is near-optimal.

The shortcoming of the formulation in (10.1) is that it tries to find a static roadmap that covers all problems. There might not be a sparse enough roadmap that achieves this. However, we can extract some context from the problem and use it to generate a roadmap. Fig. 10.2 illustrates the desired policy that maps extracted context to a roadmap. Note that it roughly identifies a set of paths that are likely to contain a good solution. Actual identification of the optimal path requires evaluation of the roadmap.

How can we learn such a policy? One approach is to design a policy to select edges from a dense roadmap $\mathcal{G}_{\text{dense}}$. At every iteration i, the policy can take as input the context around an edge and output a $\{0, 1\}$ decision on whether to accept that edge. It can optionally also take as input its past decisions. The policy is invoked till a sufficient budget on roadmap size is reached. Alternatively, one can also learn a generative model from context to roadmap in a similar spirit to Ichter et al. [2017]. The roadmap learning problem might be easier since the policy is not burdened with predicting the optimal path but a set of good paths. We also note that such a policy must operate within some constraints. It can only extract a minimal amount of context from the problem, otherwise, the roadmap generation effort would exceed the roadmap evaluation effort.

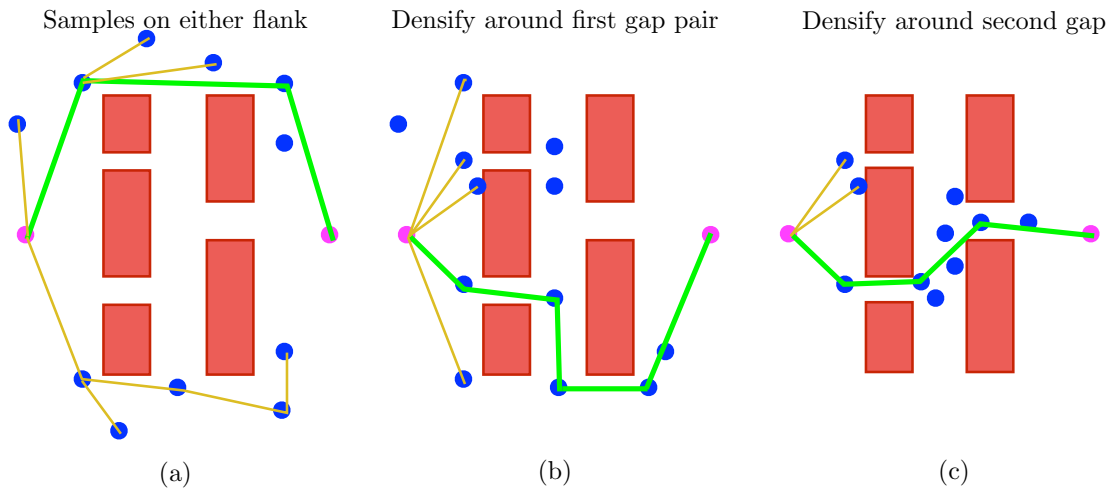


Figure 10.3: Learning anytime planning via incremental densification. (a) Learner tries to find a feasible solution initially. Hence samples are spread out on either flank of the obstacles. (b) Once an initial solution is found, the sampler densifies around the first pair of gaps to improve the solution. (c) In the final stages, the sampler completely focuses on the final gap to get the optimal solution.

10.2.2 Learning anytime policies in planning

Our formulation for learning heuristic or collision checking policies posed the problem as finding a feasible path while incurring minimal search effort. What we observe in practice is that such policies can find paths very quickly, thus having sufficient computational budget left over. We would like such policies to use the remaining budget to continue to refine the solution quality. Hence we consider the anytime planning paradigm.

Key to achieving anytime performance is the property of incremental densification [Choudhury et al., 2017h]. It is best understood in the context of sampling configurations. This property is the ability of a sampler to begin by coarsely spreading out samples and with time concentrate these samples around the optimal solution. Janson et al. [2015a] showed that a class of deterministic sampling schemes are endowed with this property. They examined the case where samples were created by low dispersion deterministic sequence (for details refer to Section 3.3.1) and showed that the length of the shortest path computed on a $r-$disk roadmap can be bounded with respect to the optimal solution. An alternate approach to densification has been pursued by Gammell et al. [2014]. They showed that a solution length at a given time step can be used to restrict the volume of configuration space that can possibly improve solution cost. Sampling within this volume increases the likelihood of finding a better path which in turns shrink the volume.

Both of these approaches fail to reason about the distribution of obstacles in the world. Hence they tend to waste samples by placing them in areas where no solution is likely to exist. This is where learning can play an important role. We can learn a policy to sample in areas where a better solution is likely to exist. Such a policy must learn a trade-off between the likelihood of a path and the solution quality of a path. Fig. 10.3 illustrates the desired behaviour of such a sampler.

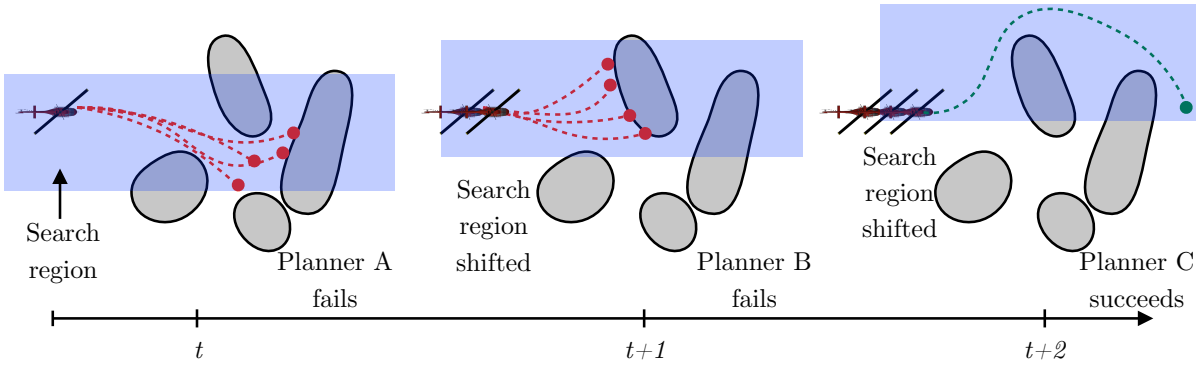


Figure 10.4: Planning problems are correlated temporally. A meta-planner can exploit this correlation. At time-step t, it invokes a planner that fails to find a solution. Hence at time-step $t+1$, it invokes a different planner with a different search region. Failure at both these time-steps is used to select a planner with a different search region at time-step $t+2$. This planner solves the problem.

A promising avenue of future work would be to build upon the formulation of pareto-optimal planning by Choudhury et al. [2017g]. Given a dense roadmap $\mathcal{G}_{\text{dense}}$, and a belief over the configuration space, one can map every path ξ to two criteria - path length $L(\xi)$ and collision measure $M(\xi)$. The pareto-optimal set is the set of paths that dominate in one criteria or the other. Hence we would want to train a policy to map the belief over the configuration space to a subgraph that contains members from this set. As the planner searches with this subgraph, and updates its belief over the space, we would expect the policy to densify around potentially optimal solutions.

An alternate, albeit equally promising avenue would be to pursue anytime prediction from a purely learning perspective. Grubb and Bagnell [2012] formulates the problem of anytime prediction - given a set of predictors, a computation cost for each predictor and a budget, select a set of predictors such that their weighted sum minimizes risk. It would be interesting to import this idea into a path planning setup. Each predictor probes the environment a certain amount and predicts a path. We can define a loss function as the cost of the path. Hence an anytime prediction would correspond to finding better and better paths as more context is extracted from the problem.

10.2.3 Planners as feature extractors

We discussed in Chapter 5 about the accuracy versus complexity trade-off of in context extraction. This motivated the shift from black-box to white-box paradigm. However, there is another paradigm to consider - using the performance of the planners as sources of information.

We now pay attention to the fact that the planning problem distribution is actually a data stream of planning problems that a robot encounters as its moving in the environment as shown in Fig. 10.4. If certain black-box planners were to perform poorly at a given decision step t, it would provide valuable information in deciding what approaches to invoke at subsequent time steps. An architecture for such a meta-planner was discussed in Definition 2.3 in Section 2.4.

Formally speaking, we have a library $\mathcal{L}$ of planners. At time step t, context f_t is extracted

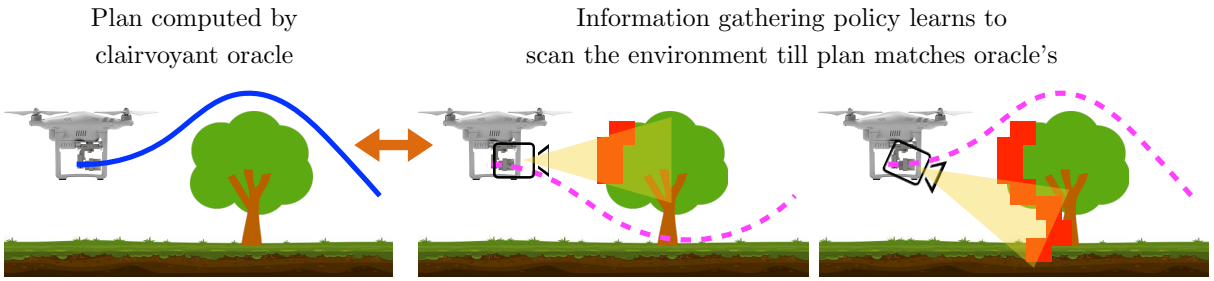


Figure 10.5: Learning to gather enough information to plan correctly. (a) At train time, we have full access to the world map. We can query a clairvoyant oracle to plan a path to the goal. (b) We then simulate the sensor model and observe the planned path with partial information. We see the planned path varies vastly from the one computed by the oracle. (c) The learner plans sensing actions till the plans match (at least in the sensing horizon).

from the planning problem. This is then used by a policy π to predict an ensemble $\mathcal{E}_t$. We receive as feedback a cost vector J_t. Now consider, the fact that we wish to include past performance in the context. We can choose a window K of past data to consider. Then we get an appended feature vector $\tilde{f} = \begin{bmatrix} f_t^T & \mathcal{E}_t & J_t & \dots & \mathcal{E}_{t-K} & J_{t-K} \end{bmatrix}$. This appended context can be used to train the list prediction as presented in Section 5.6. Since this is a richer policy class, we expect better performance.

However, it is no longer the case that greedily minimizing loss is the correct approach. This is because there maybe informative planners - planners that would help subsequent list predictors to make better decisions. If these informative planners do not have low risk, they will not be selected by the greedy method. The dynamic ensemble should balance exploitation - minimizing current loss - with exploration - providing information to aid prediction at future time steps.

The above exploration-exploitation problem is complicated due to the presence of context f and a complex policy class π. However, we have two insights that we can use to attack the problem. Firstly, we have a good reference policy π that we can train to be purely exploitative. We can do this iteratively by training a policy, use it to make predictions on the training dataset and then use that data to update the appended feature vector $\tilde{f}$ and re-train. Our objective is to perform at least as well as this policy. Methods such as 'Learning to Search Better Than Your Teacher' [Chang et al., 2015] address similar problems of policy improvement given a reference policy. Secondly, we have an ensemble where elements can be executed in parallel. Hence, one can assign one element to do pure exploration while the remaining elements exploit. This luxury is not available in the single prediction paradigm and might result in a simplified training process.

10.2.4 Active perception for planning

The trajectory planning architecture that we present in Chapter 9 belongs to a class of receding horizon planning approach. For such approaches, the robot plans a path with the partial information gathered so far, moves a fraction along the path, updates the world model and replans. A common practice is to employ "optimism in the face of uncertainty", i.e. to assume unknown

regions of the world are actually valid.

This can have dangerous consequences. Consider the example shown in Fig. 10.5. By assuming that unknown space is in fact free, the UAV risks planning paths that will inevitably lead to collisions. However, we note that the UAV does not have to have complete knowledge of the world. It simply needs enough knowledge to choose the same motions it would have chosen had it know the world entirely.

Consider learning a information gathering policy π. The objective of this policy is to learn enough information about the world so as to make sure that the receding horizon planning system does not execute bad actions. An approach to training such policies will be to build on ideas introduced in Chapter 7 on the imitation of clairvoyant oracles. During training, we have access to the full world map and can invoke oracles to plan complete paths. We can then train policies to plan minimal sensing actions enough information about the world such that the receding horizon planner outputs a similar path (at least within the sensing range). Such policies might be incentivized to optimally gather information to invalidate the currently planned path.

A related direction of future work would be to plan using multiple hypotheses (as suggested by Dey [2015]) of the world generated from the partial information. The recent success of approaches such as Generative Adversarial Networks [Radford et al., 2015] indicate that such models would be made available from the perception community. If we were to plan using such models, for every plausible world, we would have a different path. Our objective would be to then sense enough locations such that we are left with consensus on one path being feasible. This problem has the same structure as the Bayesian active learning framework presented in Chapter 8. Hence, we can consider adopting those algorithms to compute active sensing policies.

10.2.5 Motion planners as powerful policy classes

Motion planning is not an end in itself. Almost always it is used to generate a policy that enables a robot to acheive a task. The performance on this task is what we actually are concerned about. Hence we set up the motion planning problem such that good planning performance implies good task performance. Can we learn how to plan to achieve good task performance?

This question has been heavily examined in the context of inverse reinforcement learning [Ratliff et al., 2009b, Ziebart et al., 2008] where a cost function is learnt from demonstration. However this assumes that we have access to demonstrated trajectories that we wish to emulate. The setting of reinforcement planning as defined by Zucker and Bagnell [2012] is much closer to what we aim for. In such a setting, a planner uses a cost function to generate a plan which is tracked by a low level control policy. We want to learn a cost function that generates good policies.

Viewing motion planners as a powerful policy class is indeed an attractive idea. Such policies generalize well to new scenarios, have more interpretability and can demonstrate long term intent. Zucker and Bagnell [2012] do make some assumptions such as the existence of a planner that can optimally solve the problem. A promising research direction would be to relax such assumptions. Perhaps a goal to aim for is to learn objectives that not only succeed on the task but are easily amenable to optimization.

10.3 Concluding remarks

We conclude this thesis with a discussion of frequently asked questions and lessons learned.

Q 1 (TLDR). "I have a robot that may or may not be a UAV. How should I apply this thesis?"

We try to briefly summarize the steps one should follow when deciding whether to adopt some of the ideas presented in this thesis:

Step 1: We first need to figure out if the robot needs to invoke a motion planner. Does the robot perform a task that requires it to plan motions? Is the objective function and constraint set known? Does the robot have sufficient information at runtime to solve this planning problem? If not, does it suffice for the robot to plan with partial information and re-plan as it learns more about the world? If the answer to all these questions is yes, we have a problem that requires a plan to be computed. Any module that computes a good plan has a good performance on the original task. Hence from this step onwards, we shall only be reasoning about the planning problem.

Step 2: We now need to figure out if a single planner suffices. Is there a real-time constraint? Is the problem high dimensional (at least > 3)? Is it expensive to evaluate a candidate solution? Are there multiple plausible planning strategies that can solve the problem? Does there exist a single hand-engineered planner that performs reasonably well for all problems in the problem space? If no, we need a design principle to construct an effective planning module.

Step 3: We now need to figure out the problem distribution. Is there a high fidelity simulator that can simulate the kind of problems the robot is expected to encounter? If not, is it possible to manually collect datasets by operating the robot? Is the distribution of planning problems varying, i.e. obstacle configuration, start goal combinations, constraint or objective function? If no, then the planner does not need to be adaptive. We are in the setting of Chapter 4 where we need to design an ensemble of expert planners. Else the planner needs to be adaptive, and we proceed to the next step.

Step 4: We need an adaptive planning system. Can we create a library of planners for our problem? Can we collect datasets and evaluate this library offline? Can we extract features from the problem? If so, we have the capability to design a meta-planner that can select from a library. We can start with a static ensemble and if that does not suffice proceed to a dynamic ensemble. Refer to Chapter 5.

Step 5: When the robot fails at test time, can we ensure the robot executes a contingency? Can we come up with a procedure to solve the failure problem (by possibly invoking a computationally expensive solver or requesting human assistance)? Can we store such solutions and apply it should the robot encounter a similar situation? If so, deciding which solutions to keep and which to discard is addressed in Chapter 6.

Step 6: Finally, we investigate if we need better planning algorithms? Do we have a good implicit graph/tree based method but are in need of search/expansion heuristics? Do we have a roadmap but are unsure which edges to evaluate? If so, we should refer to Chapter 7 and Chapter 8.

Q 2 (Dataset Construction)**.** What are some considerations when creating a planning dataset?

A common practice when evaluating the efficacy of planning algorithms is to either design a good canonical problem (e.g. a narrow passage) or select a compelling problem instance (e.g. hard manipulation problem [Schulman et al., 2013]). However, this work differs in the sense that we deal with a dataset of problems that represent a distribution that the robot encounters.

In the initial phases of our research, creating such datasets were problematic. While we had single instances of problems we wanted to solve (e.g. planning in a Manhattan world), we did not have a database of such problems. The most pragmatic option was to hand design parametric distribution classes that could generate such worlds. A single random seed is sufficient to represent such a world, hence reproducibility becomes easy. When we sampled a database of problems, we encountered two types of problems. Firstly, there were many problems for which no solution exists. Automatically detecting and pruning out such examples is tricky - a simple solution was to run RRT-Connect for a given time-period until a solution was found. Secondly, there were many problems that were trivial, i.e. the straight line solution from start to goal was free. These too had to be pruned out.

When we had a sufficiently flexible simulator, it was possible to collect data from it. We load a map from a database (it also suffices to have a single map), select start and goal locations (such that they are not in collision) and compute a feasible plan offline assuming the map is known. We then have the robot execute the path while incrementally sensing the map. We serialize the planning problems by storing the current grid. When collecting data in this fashion, we found that the robot does not encounter enough "hard" problems. This is because the offline computed plan avoids obstacles from much earlier on and does not guide the robot into difficult situations. A way to circumvent this issue is to come up with a "noisy agent" - the robot follows the plan with ε probability otherwise executes random motions. This seemed to populate the database with harder problems.

Q 3 (Averaging over Worlds)**.** How do we deal with high variance in planner performance?

We observed that often the cost of a planner on a database has a high variance. This variance persists even if we normalize the cost with problem-specific constants. Our solution was to switch to using confidence intervals to report our results. These confidence intervals are based on quantiles and do not make any assumptions about the underlying distribution.

The usual answer to high variance is that the data is insufficient. But based on visual inspection, we conjecture that the planner performance for such databases might be fundamentally multi-modal. An important topic of future research is to lineate methods of comparing such distributions. For example, if we are particularly risk-averse, we may consider comparing the CVaR (Conditional Value at Risk) [Uryasev, 2000] of two planning algorithms.

Q 4 (Planning-as-Inference or Inference-as-Planning)**.** Is there a clear division of roles?

We were driven to the topic of adaptive motion planning based on a concrete need - we wanted a motion planning module that had consistently good performance. We were initially quite clear on the separation of labour between learning and planning - we had a library of good planners and wanted a meta-planner to be able to select from it. As we switched to investigating adaptation within a planning cycle, we realized the two were actually much closer. Once we began viewing the planning operations as sequential decision making, we realized that there were much richer problems than that of supervised learning. A planner is effectively doing inference - based on the history of decisions and outcomes, it is choosing good decisions that will enable it to find feasible/optimal paths quickly. This naturally leads to subproblems in active learning or POMDPs.

Interestingly, this equivalence goes both ways. For certain inference problems in a structure prediction setting, especially in continuous output space, planning algorithms can be quite useful. If we can identify an objective function on the output space, a planner is very effective at being able to systematically search over candidate outputs. Of course such problems are doubly hard than adaptive motion planning - one has to simultaneously learn good objective functions as well as pair them with good search strategies. Nevertheless, we expect some of the insights in this thesis to carry over to these completely new problem settings.

Appendices

A

Planning Problem for UAVs

We present details on the planning problems for different UAV applications. The constraints for the autonomous helicopter and for quadrotors vary slightly. We present a joint framework and highlight these differences in a case by case basis.

A.1 Dynamics constraints

We now define the dynamics constraints $\mathcal{F}_{\text{dyn}}(\sigma) = 0$ and $\mathcal{H}_{\text{dyn}}(\sigma) \leq 0$. Vehicles such as the quadrotor and the hexarotor have a uniform set of dynamic constraints at all speeds. The full scale helicopter has different constraints for low speed and high speed.

A.1.1 Low speed constraints

The system is modelled as a composition of single and double integrator systems. $x(t), y(t), z(t)$ are all independent double integrator models. $\psi(t)$ is a single integrator

$$\begin{aligned}
|\dot{x}(t)| &\leq v_{\max} \\
|\dot{y}(t)| &\leq v_{\max} \\
|\dot{z}(t)| &\leq v_{z,\max} \\
|\ddot{x}(t)| &\leq a_{\max} \\
|\ddot{y}(t)| &\leq a_{\max} \\
|\ddot{z}(t)| &\leq a_{z,\max} \\
|\dddot{x}(t)| &\leq j_{\max} \\
|\dddot{y}(t)| &\leq j_{\max} \\
|\dddot{z}(t)| &\leq j_{z,\max} \\
\left|\dot{\psi}(t)\right| &\leq \dot{\psi}_{\max}
\end{aligned} \tag{A.1}$$

A.1.2 High speed constraints

These set of constraints are applicable only for the full scale helicopter when the speed is above a critical threshold, $\left\|\dot{x}(t) \quad \dot{y}(t) \quad \dot{z}(t)\right\|_2 \geq v_{\text{crit}}$. We model the system as a fixed wing model moving in airframe. Let the airframe be $\mathcal{A}$ and groundframe be $\mathcal{W}$. Let $v_a(t)$ be the airspeed trajectory. Let $\phi_a(t)$ be the airspeed roll trajectory. Let $\psi_a(t)$ be the airspeed roll trajectory.

$$\left.\begin{aligned} \dot{\psi}_a(t) &= \frac{g \tan \phi_a(t)}{v_a(t)} \\ \dot{x}_a(t) &= v_a(t) \cos \psi_a(t) \\ \dot{y}_a(t) &= v_a(t) \sin \psi_a(t) \end{aligned}\right\} \quad \begin{array}{l}\text{Dynamics in} \\ \text{Airframe}\end{array} \tag{A.2}$$

$$\left.\begin{aligned} \dot{x}(t) &= \dot{x}_a(t) + v_w \cos\left(\psi_w - \psi_a\right) \\ \dot{y}(t) &= \dot{y}_a(t) + v_w \sin\left(\psi_w - \psi_a\right) \\ \psi(t) &= \tan^{-1}\left(\frac{\dot{y}(t)}{\dot{x}(t)}\right) \end{aligned}\right\} \quad \begin{array}{l}\text{Airframe to} \\ \text{Groundframe}\end{array} \tag{A.3}$$

$$\left.\begin{aligned} |v_a(t)| &\leq v_{\max} \\ |\dot{v}_a(t)| &\leq a_{\max} \\ |\ddot{v}_a(t)| &\leq j_{\max} \\ |\phi_a(t)| &\leq \phi_{\max} \\ \left|\dot{\phi}_a(t)\right| &\leq \dot{\phi}_{\max} \\ \left|\ddot{\phi}_a(t)\right| &\leq \ddot{\phi}_{\max} \\ |z(t)| &\leq v_{z,\max} \\ |\dot{z}(t)| &\leq a_{z,\max} \\ |\ddot{z}(t)| &\leq j_{z,\max} \end{aligned}\right\} \quad \begin{array}{l}\text{Dynamics} \\ \text{Constraints}\end{array} \tag{A.4}$$

In addition to the dynamics constraints, there are some additional constraints that arise from performance charts for a helicopter.

Height-velocity chart

Fig. A.1(a) illustrates the constraint. The constraint correponds to areas in the state space of the system which are deemed unsafe - areas from where recovery is not possible in the event of engine failure. It corresponds to a set of regions that the height about ground $z(t)$ and airspeed $v_a(t)$ cannot enter.

Torque limits chart

Fig. A.1(b) shows the torque chart. The torque drawn from the motor depends on the airspeed $v_a(t)$ and the climbrate $-\dot{z}(t)$. The total torque cannot exceed 100%. Hence for a given airspeed,

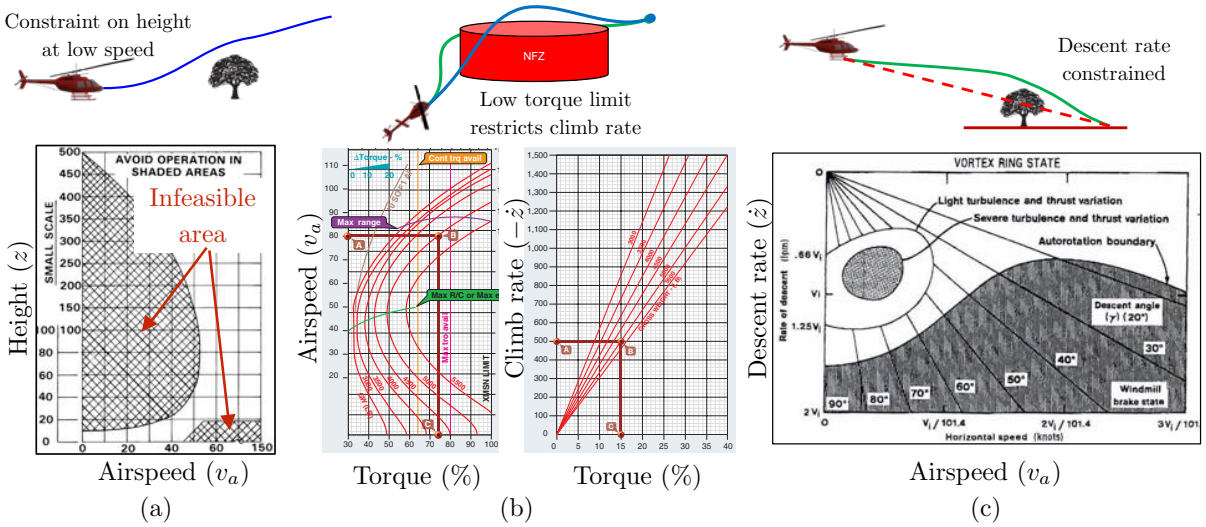


Figure A.1: Dynamics constraints from performance charts of a helicopter. (a) The height-velocity chart (b) The torque limits chart (c) The autorotation limits chart

the climbrate is limited and this dictates if the system can avoid obstacles by flying over them or it has to fly around them.

Autorotation limits chart

Fig. A.1(c) shows the descent rate and airspeed combination that leads to autorotation - a condition where the engine load is 0. This is undesirable. Hence this imposes a constraint on $v_a(t)$ and $\dot{z}(t)$.

A.2 Route constraints

We now define the route constraints $\mathcal{F}_{\text{route}}(\sigma) = 0$ and $\mathcal{H}_{\text{route}}(\sigma) \leq 0$. We will now define constraints imposed by the route specification. Let $\text{wp}_i \in \mathbb{R}^3$ be a 3D waypoint. The route is a set of n waypoints $\{\text{wp}_i\}_{i=1}^{n}$. The waypoints define $n-1$ segments. Each segement is associated with a flight corridor. The function $\texttt{corr}_i(\sigma(t)) \to \{0, 1\}$ is an indicator function to specify whether a point in the trajectory lies inside a flight corridor. Each flight corridor is also associated with a velocity limit $v_{\text{seg},i}$.

The constraints are as follows

$$\begin{aligned} \sum_{i=0}^{n-1} \texttt{corr}_i(\sigma(t)) > 0 \\ \texttt{corr}_i(\sigma(t)) v_a(t) \leq v_{\text{seg},i} \quad \forall i \in \{1, \ldots, n-1\} \end{aligned} \tag{A.5}$$

A.3 Safety constraints

We now describe the safety constraints $\sigma(t) \in \Sigma_{\text{valid}}$. They are due to obstacles and no-fly-zones

Algorithm 20: $\texttt{ttc}\left(\sigma(t)\right)$

$t_{\min} \leftarrow \infty$;
for $\mathbf{p}_{\text{obs}} \in \mathcal{P}_{\text{obs}}$ **do**
  $\mathbf{p} \leftarrow \begin{bmatrix} x_\sigma(t) & y_\sigma(t) & z_\sigma(t) \end{bmatrix}^T$;
  $\mathbf{v} \leftarrow \begin{bmatrix} \dot{x}_\sigma(t) & \dot{y}_\sigma(t) & \dot{z}_\sigma(t) \end{bmatrix}^T$, $\mathbf{v}_{\text{xy}} \leftarrow \begin{bmatrix} \mathbf{v}(1) & \mathbf{v}(2) \end{bmatrix}^T$, $\mathbf{v}_{\text{z}} \leftarrow \begin{bmatrix} ||\mathbf{v}_{\text{xy}}||_2 & \mathbf{v}(3) \end{bmatrix}^T$;
  $\mathbf{d} \leftarrow (\mathbf{p}_{\text{obs}} - \mathbf{p})$, $\mathbf{d}_{\text{xy}} \leftarrow \begin{bmatrix} \mathbf{d}(1) & \mathbf{d}(2) \end{bmatrix}^T$, $\mathbf{d}_{\text{z}} \leftarrow \begin{bmatrix} ||\mathbf{d}_{\text{xy}}||_2 & \mathbf{d}(3) \end{bmatrix}^T$;
  $\Delta_{\text{xy}} \leftarrow \frac{\mathbf{d}_{\text{xy}}^T \mathbf{v}_{\text{xy}}}{||\mathbf{d}_{\text{xy}}||_2 ||\mathbf{v}_{\text{xy}}||_2}$, $\Delta_{\text{z}} \leftarrow \frac{\mathbf{d}_{\text{z}}^T \mathbf{v}_{\text{z}}}{||\mathbf{d}_{\text{z}}||_2 ||\mathbf{v}_{\text{z}}||_2}$;
  $t_{\text{ttc,obs}} \leftarrow$

$$\min\left(t_{\text{ttc,max}}, \underbrace{\left(\frac{||\mathbf{d}||_2}{||\mathbf{v}||_2}\right)}_{\text{time}} \underbrace{\left(1 + \frac{\eta_{\text{xy}}}{2}\max\left(0, \Delta_{\text{xy,max}} - \Delta_{\text{xy}}\right)^2\right)}_{\text{inflation due to head-on collision}} \underbrace{\left(1 + \frac{\eta_{\text{z}}}{2}\max\left(0, \Delta_{\text{z,max}} - \Delta_{\text{z}}\right)^2\right)}_{\text{inflation due to vertical collision}}\right);$$

  if $t_{\text{ttc,obs}} \leq t_{\min}$ **then**
    $t_{\min} \leftarrow t_{\text{ttc,obs}}$;
return $t_{\min}$;

A.3.1 Obstacle constraints

The convention for specifying such constraints is usually that the trajectory must be clear of obstacles by a certain distance. However, we use a different constraint formulation, *time to collision*. This is a constraint that scale with velocity and also depends on the velocity vector direction. It tries to model how far away in temporal space is an obstacle.

Lets talk about the representation offered to us by the perception system. The perception system internally stores $\mathcal{P}_{\text{obs}}$ - the set of all occupied positions under its current belief. It offers a function $\texttt{obs}\left(\mathbf{p}\right) = \mathbb{I}\left(\mathbf{p} \in \mathcal{P}_{\text{obs}}\right)$ to say if a position is in an obstacle or not.

It also offers a distance function $d_{\text{obs}}\left(\mathbf{p}\right) = \min_{\mathbf{p}_{\text{obs}} \in \mathcal{P}_{\text{obs}}} ||\mathbf{p} - \mathbf{p}_{\text{obs}}||_2$ and $p_{\text{obs}}\left(\mathbf{p}\right) = \arg\min_{\mathbf{p}_{\text{obs}} \in \mathcal{P}_{\text{obs}}} ||\mathbf{p} - \mathbf{p}_{\text{obs}}||_2$

We now define a time to collision function $\texttt{ttc}\left(\sigma(t)\right)$ in Algorithm 20. We enforce a constraint where $\texttt{ttc}\left(\sigma(t)\right) \geq t_{\text{coll,min}}$.

A.3.2 No Fly Zones

A No Fly Zone (NFZ) is a volume in airspace defined by the human operator where it is unsafe for the UAV to enter. NFZs can be published at any point of time triggered by reasons such as the presence of other aircrafts, threats to the UAV or local weather disturbances. Avoiding an NFZ is a hard constraint. We do make the assumption that NFZs will be published in a reasonable time frame such that the UAV can avoid it. We define an NFZ as a 2D polygon with a lower and upper height. Each point along a trajectory is checked to see if it is inside this volume.

B

Dynamics Projection Filter

The DPF [Choudhury and Scherer, 2015] accepts an infeasible trajectory (workspace trajetory) as input, uses a controller to track this trajectory and the outputs the configuration trajectory traced out by the system. To provide guarantees about the output, the first component required is a control-Lyapunov function (CLF) that stabilizes around a feasible workspace trajectory. Since the controller can observe all states, has a perfect model and is free from any disturbance, approaches such as feedback linearization and backstepping can be applied [Jung and Tsiotras, 2008, Lapierre et al., 2007, Micaelli et al., 1993].

Let the workspace trajectory be $\xi : [0, 1] \to \mathbb{R}^w$. A control-Lyapunov stabilized trajectory firstly requires the existence of a feedback control $u = K(x, \xi, \tau)$ where $x \in \mathbb{R}^d$ is the configuration space coordinate of the robot, τ is the index of the workspace trajectory. It also requires the definition of index dynamics $\dot{\tau} = \Gamma(x, \xi, \tau)$. Careful selection of these functions can ensure a function $V(x, \xi, \tau)$ to satisfy the Lyapunov criteria when ξ is dynamically feasible. If $x \in X_\xi$ implies perfect tracking, the Lyapunov criteria is $V(x, .) > 0, \dot{V}(x, .) < 0, \forall x \in X \setminus X_\xi$ and $V(x, .) = 0, \forall x \in X_\xi$ (globally asymptotic stable version).

A further local exponential stability is also required, i.e,

$$\left|\frac{\mathrm{d}V(., \tau)}{\mathrm{d}\tau}\right| > \alpha V(., \tau),\ \alpha > 0,\ V(.) < V_{max}.$$

where the rate of exponential stability α determines how fast convergence occurs. This property will be used to guarantee decay properties of the CLF. We used the an extension of the controller presented in Jung and Tsiotras [2008]

The output $x(t) = \mathrm{DPF}(\xi)$ is given by

$$\begin{bmatrix} x(t) \\ \tau(t) \end{bmatrix} = \begin{bmatrix} x(0) \\ 0 \end{bmatrix} + \int_0^t \begin{bmatrix} f(x(t), K(x(t), \xi, \tau(t))) \\ \Gamma(x(t), \xi, \tau(t)) \end{bmatrix} \mathrm{dt} \tag{B.1}$$

In the scope of this work, we consider ξ to be approximated by a set of workspace samples at equal discretization $\Delta\tau$: $\xi \approx (q_1, q_2, \ldots, q_n)^{\mathrm{T}} \in \mathbb{R}^{n \times w}$, with q_0 and q_{n+1} the fixed starting

and ending points of the trajectory. This will facilitate concrete expressions on bounds that are parameterization dependent. However, the method remains valid for other parameterizations, such as splines.

Under the assumption that the linear segments between waypoints are dynamically feasible (there exists $u \in U$ which allows perfect tracking) , the Lyapunov function $V(.)$ converges in the straight line portion of ξ. However, at every waypoint it increases by $\Delta V > 0$ because of the angle change at the waypoint. The more jagged ξ is the more the cumulative effect of ΔV will be. The maximum $V(.)$ at anytime determines how much deviation $x(.)$ has from ξ. We first establish that under certain assumptions about $V(.)$, a bounded Lyapunov value implies a bounded intersegment deviation.

Theorem B.1 (Bounded Intersegment Deviation)**.** Given a desired bound on the Lyapunov function $V(x, \xi, \tau) \leq V_{max}, \forall x, \tau \in \mathrm{DPF}(\xi)$, the intersegment deviation is also bounded, i.e.

$$\left(1 + \frac{(q_i - q_{i-1})^{\mathrm{T}}(q_i - q_{i+1})}{\|q_i - q_{i-1}\| \, \|q_i - q_{i+1}\|}\right) \leq \rho_{max}.$$

C

Library of Expert Planners for UAVs

We now provide details on the hand-designed library of expert planners used across all 3 platforms - full scale helicopter, large hexarotor and small quadrotor. We organize these expert planners into different classes. Each of these classes define a surrogate problem and a class of planning algorithms that can efficiently solve problems in this class.

As discussed in Chapter 4, the parameters of the surrogate state space has to derived from dynamic constraints of the UAV. For a full scale helicopter, limits on the roll angle and velocity are used to choose the curvature constraint that a path planner must satisfy. For a quadrotor, limits on the acceleration is used to choose the curvature limits.

C.1 Sampling-based approaches on a curvature constrained state space

Table C.1: Surrogate problem on curvature constrained state space

Space	Start / Goal	Objective	Constraints
$\mathcal{S} = \mathcal{P} \circ SO(2)$	$\mathbf{x_s} = (s_x, s_y, s_z, s_\psi)$	$\tilde{J}(\xi) = \int_0^1 \left\|\dot{\xi}(\tau)\right\| \mathrm{d}\tau$	$\tilde{\mathcal{F}}$: $\psi = \tan^{-1}\left(\frac{\dot{y}}{\dot{x}}\right)$
$\xi[0,1] \mapsto (x, y, z, \psi)$	$\mathbf{x_g} = (g_x, g_y, g_z, g_\psi)$		$\tilde{\mathcal{H}}$: $\frac{\dot{x}\ddot{y} - \dot{y}\ddot{x}}{(\dot{x}^2+\dot{y}^2)^{3/2}} \leq \kappa_{\max}$
			$\frac{\dot{z}}{\sqrt{\dot{x}^2+\dot{y}^2}} \leq \gamma_{\max}$
			$\tilde{\Xi}_{\text{valid}}$: $\texttt{nfz}(x, y, z) = 0$
			$\texttt{ttc}(x, y, z) \geq t_{\text{coll,min}}$

Table C.1 shows the surrogate problem. We now specify different sampling strategies which can be used to create different expert planners as mentioned in Table C.2.

1. `Uniform` Let $U(\texttt{Vol})$ denote an uninform probability distribution over volume `Vol`. Let $\texttt{Cuboid}(\mathbf{p}_{\min}, \mathbf{p}_{\max})$ be the volume of the cuboid denoting the valid workspace where $\mathbf{p}_{\min}$

and $\mathbf{p}_{\max}$ are the corners.

2. `Workspace` This sampling scheme is specific to the RRT* algorithm. The overview of this sampling strategy is that a 3D workspace point (x, y, z) is sampled leaving the dimension ψ as a free variable. Among the set of near neighbours, the best parent to this workspace point is computed. The ψ value is then the resulting value by steering the parent towards this configuration.

3. `Tunnel`

   This sampling scheme is specific to the RRT* algorithm. This extends the `Workspace` sampling strategy by sampling workspace points in a tunnel around a nominal 3D path (usually the analytic Dubins path joining $\mathbf{x_s}$ and $\mathbf{x_g}$). The tunnel radius is specified by δ_{tunn}.

Table C.2: Library of expert sampling based planner for curvature constrained state space

Name	Algorithm	Sampler	Parameters
`RRT*Uniform`[1]	RRT*	`Uniform`	Radius (γ) : 3.0
`RRT*Workspace`[1]	RRT*	`Workspace`	Radius (γ) : 3.0
`RRT*Workspace`[2]	RRT*	`Workspace`	Radius (γ) : 1.0
`RRT*Tunnel`[1]	RRT*	`Tunnel`	Radius (γ) : 3.0, Tunnel (δ_{tunn}) : $\frac{1}{\kappa_{\max}}$
`RRT*Tunnel`[2]	RRT*	`Tunnel`	Radius (γ) = 3.0, Tunnel (δ_{tunn}) : $\frac{1}{3\kappa_{\max}}$
`BIT*`[1]	BIT*	`Uniform`	Radius (γ) = 3.0, Batch : 300
`BIT*`[2]	BIT*	`Uniform`	Radius (γ) = 3.0, Batch : 500
`RRTConnect`[1]	RRTConnect	`Uniform`	
`RRTConnect`[2]	RRTConnect	`Tunnel`	Tunnel (δ_{tunn}) : $\frac{1}{\kappa_{\max}}$
`T-RRT`[1]	T-RRT	`Uniform`	Temperature (T) : 2.0, Frontier (ρ) : 0.1
`T-RRT`[2]	T-RRT	`Uniform`	Temperature (T) : 10.0, Frontier (ρ) : 0.1
`LBT-RRT`[1]	LBT-RRT	`Uniform`	Approximation (ε) : 0.40
`LBT-RRT`[2]	LBT-RRT	`Uniform`	Approximation (ε) : 0.04
`InformedRRT*`[1]	Informed RRT*	`Uniform`	Radius (γ) : 3.0
`STRIDE`[1]	STRIDE	`Uniform`	
`EST`[1]	EST	`Uniform`	
`EST`[2]	EST	`Tunnel`	Tunnel (δ_{tunn}) : $\frac{1}{\kappa_{\max}}$

C.2 Sampling-based approaches on a holonomic 3D state space

Table C.3 shows the surrogate problem. The heading ψ can be computed as $\psi = \tan^{-1}\left(\frac{\dot{y}}{\dot{x}}\right)$ once the path is computed. We now specify different expert planners in Table. C.4.

Table C.3: Surrogate problem on 3D state space

Space	Start / Goal	Objective	Constraints
$\mathcal{S} = \mathcal{P}$ $\xi[0,1] \mapsto (x,y,z)$	$\mathbf{x_s} = (s_x, s_y, s_z)$ $\mathbf{x_g} = (g_x, g_y, g_z,)$	$\tilde{J}(\xi) = \int_0^1 \left\|\left\|\dot{\xi}(\tau)\right\|\right\| \mathrm{d}\tau$	$\tilde{\Xi}_{\text{valid}}$: $\texttt{nfz}(x,y,z) = 0$ $\texttt{ttc}(x,y,z) \geq t_{\text{coll,min}}$

Table C.4: Library of expert sampling based planner for 3D holonomic state space

Name	Algorithm	Sampler	Parameters
BIT*[3]	BIT*	Uniform	Radius (γ) = 3.0, Batch : 300
RRTConnect[3]	RRTConnect	Uniform	

Table C.5: Surrogate problem on space of smooth trajectories

Space	Start / Goal	Objective	Constraints
$\mathcal{S} = \mathcal{P}$ $\xi[0,1] \mapsto (x,y,z)$	$\mathbf{x_s} = (s_x, s_y, s_z)$ $\mathbf{x_g} = (g_x, g_y, g_z)$	$\tilde{J}(\xi) = \int_0^1 \left(\lambda \left\|\left\|\dot{\xi}(\tau)\right\|\right\|_2^2 + \frac{1}{2}\left(\texttt{ttc}_{\max} - \texttt{ttc}(\xi(\tau))\right)^2 \left\|\left\|\dot{\xi}(\tau)\right\|\right\|\right) \mathrm{d}\tau$	$\tilde{\mathcal{H}}$: $\frac{\dot{z}}{\sqrt{\dot{x}^2+\dot{y}^2}} \leq \gamma_{\max}$

C.3 Trajectory Optimization Approaches on the Space of Smooth Trajectories

Table C.5 shows how a surrogate trajectory optimization problem can be created. We created two such expert planners CHOMP[1], which ignores the glidelsope constraint $\tilde{\mathcal{H}}$ and CHOMP[2] which solves a constrained optimization problem. Both these planners use covariant gradient descent [Ratliff et al., 2009a] to solve the optimization problem.

C.4 Discrete search on a graph of dynamically feasible motion primitives

Table C.6: Surrogate problem on a graph of dynamically feasible motion primitives

Space	Start / Goal	Objective	Constraints
$\mathcal{S} : \mathcal{G} = (V, E)$ $\xi = (e_1, e_2, \ldots, e_l)$	$\mathbf{x_s} = \texttt{ProjOnGraph}(s, \mathcal{G})$ $\mathbf{x_g} = \texttt{ProjOnGraph}(g, \mathcal{G})$	$\tilde{J}(\xi) = \sum_{i=1}^{l} \lvert e_i \rvert$	$\tilde{\mathcal{F}}$: $\texttt{corr}(e_i) = 1$ $\texttt{nfz}(e_i) = 0$ $\tilde{\mathcal{H}}$: $\texttt{ttc}(e_i) \geq t_{\text{coll,min}}$

Table C.6 shows the surrogate problem. Since the planners are discrete search based, the algorithms only deal with the graph. The resolution and primitives of the graph however encode the dynamic constraints. We use two kinds of graphs, ones with primitives constant curvature and others with primitives as straight lines. The former graph is also 4D while the latter is 3D. For the latter, the heading ψ can be computed as $\psi = \tan^{-1}\left(\frac{\dot{y}}{\dot{x}}\right)$ once the path is computed. We now specify the different expert planners n Table. C.7. All of these planners employ weighted

A* heuristic search.

Table C.7: Library of expert discrete search based planners

Name	Space	Primitives	Heuristic
`A*`^{1}	$\mathcal{S} = \mathcal{P} \circ SO(2)$	$\pm\frac{\pi}{2}$ constant curvature arcs	Dubins, Inflation: 1.0
`A*`^{2}	$\mathcal{S} = \mathcal{P} \circ SO(2)$	$\pm\frac{\pi}{4}$ constant curvature arcs	Dubins, Inflation: 10.0
`A*`^{3}	$\mathcal{S} = \mathcal{P}$	straight line, $5m$ resolution	Euclidean, Inflation: 1.0
`A*`^{4}	$\mathcal{S} = \mathcal{P}$	straight line, $1m$ resolution	Euclidean, Inflation: 10.0

C.5 Hand-designed precision planners

Finally, this last group of planners are designed to overfit to certain planning problems. They are created based on inspecting commonly occurring problems and designing custom sampling strategies / brute force path enumeration strategies.

C.5.1 Sampling detour points

This class of planners use the curvature constrained state space defined in Table C.1. They also use the RRT* algorithm to plan. However, they use a highly customized sampling strategy which belongs to one of two categories:

1. `SingleDetour`: In this sampling strategy, we first find a violation point p. To find this, we step through points along the line joining start to the goal till a point p is found to be in collision with obstacles. We then create a surface perpendicular to the direction of the line at this point p. The samples are constrained to lie on this surface. This is equivalent to finding a "detour point". Depending on the width of the surface (expressed as fraction of $\frac{1}{\kappa_{\max}}$) we have two planners - `SingleDetour`^{1} (width: $\frac{1}{\kappa_{\max}}$) and `SingleDetour`^{2} (width: $\frac{1}{3\kappa_{\max}}$).
2. `DoubleDetour`: This is similar to `SingleDetour` except we find two violation points p_1 and p_2 - one from start to goal and one in the reverse direction. Hence two surfaces are created to constrain sampling. We also create two such planners. `DoubleDetour`^{1} uses a width: $\frac{1}{\kappa_{\max}}$ and samples uniformly in heading space. `DoubleDetour`^{2} uses a width: $\frac{1}{\kappa_{\max}}$ but uses `Workspace` sampling.

C.5.2 Fixed descent direction

In this method, we collect a set of problems which none of the planners in the library were able to solve. We then solve such problems by running an RRT* method for a long time (100s). The solutions of all these problems are collected. All these solutions are then discretized to 50 waypoints. They are then concatenated to create a matrix M with dimensions $n \times 50$ where n is the number of trajectories (one matrix for each dimension). We apply principle component

analysis to extract a set of principle axis for this matrix. These directions represent motions along with a trajectory has to descend to recover the paths. The five principle directions then give rise to planners `FixedDescent`^{1}, `FixedDescent`^{2}, `FixedDescent`^{3}, `FixedDescent`^{4}, `FixedDescent`^{5}.

C.5.3 Brute force path search

The final precision planner that we create is doing a brute force path search by perturbing the path laterally to create a planner `LatPrim`^{1}.

D

Feature Extraction for Black-box Planners

D.1 Features and learners used for experiments in Section 5.7

The feature vector f is representative of context extracted from a planning problem. Policies to select an ensemble are defined on the space of contexts. Hence the context f must be able to capture sufficient statistics to characterize the performance of planners. In this section, we will define different context extractors used and the shortcomings of each such approach.

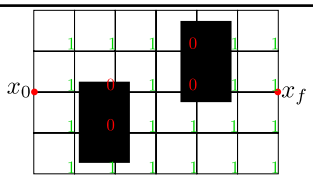
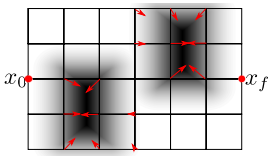
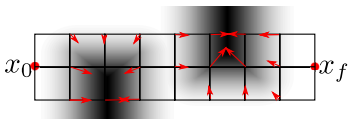
Coarse global bitmap	Coarse global gradients	Dense local gradients
		
Workspace sampled at coarse resolution	Distance map sampled at coarse resolution	Distance map sampled densely around start goal line

Figure D.1: Different feature extractors (a) Coarse bitmap created by sampling workspace at lattice points (b) Coarse gradients obtained by sampling a distance field function at lattice points. (c) Dense gradients by sampling points from a dense lattice focused about the straight line joining start and goal.

1. *Bitmap*: A workspace lattice is created at a fixed resolution, rotated and translated between start and goal locations. At each location of the lattice, a check is performed to see if the point is in collision or not. There exist two variants - coarse and dense. Coarse implies the lattice has a lower resolution but covers a larger area. Dense implies a higher resolution but the lattice is focuses in a volume around the straight line joining start and goal.

2. *Gradients*: While the bitmap only provided a binary signal, a distance field gradient provides more information. At every query point, the direction and distance to the nearest

obstacle is obtained. This also has coarse and dense fidelities.

3. *Graph*: A connectivity graph can provide information about connectedness of space. This is similar to the bitmap lattice except each edge is checked for collision. Hence this is more expensive to compute.

4. *Pyramid*: The bitmap can be used as input to a spatial pyramid [Lazebnik et al., 2006]. This can serve as a multi-resolution source of bitmap information.

5. *Histogram of Gradients (HOG)*: The gradients can be used as input to a histogram binning technique [Dalal and Triggs, 2005].

Once projected to this feature space, a variety of options exist to train learners to classify / regress to required metrics. Below are the models that we experimented with

1. *Linear*: This is the simplest model to train with convex optimization objective however has a high bias in practice.

2. *Random forest*: Proposed by Breiman [2001], these are popular for their robustness despite limited guarantees.

3. *Intersection kernel*: One can also resort to kernel methods. This kernel sum the element wise minimum of two feature vectors.

4. *Chisquare kernel*: It normalizes the squared difference between two feature vectors by their sum.

5. *Graph kernel*: A graph kernel [Vishwanathan et al., 2010] takes as input two adjacency matrix and computes shortest paths on them and compares statistic. While this is the slowest kernel function to compute, it is able to capture connectivity information.

Fig. D.2 shows a comparison of different feature representations with learning methods. The task was to predict whether expert planners could solve a planning problem. The planners and the dataset werer referred to in Section 5.7. We wish to highlight 3 combination of feature extractions and learning methods that had sufficient performance while being diverse

1. *Gradient features with kernel methods are powerful* The combination of (HOG Coarse + ChiSquare kernel) had the best accuracy among all combinations. Examination of this combination revealed that HOG was able to effectively encode artifacts such as passages, cul-de-sacs or walls - all of which dictate the effectiveness of a planning algorithm. In particular, the gradient information was particularly useful in predicting presence of bad local minima which affect the performance of trajectory optimization methods.

2. *Bitmap features with random forests are easy to compute* The attractiveness of the bitmap approach is that it requires the existence of a `IsValid`() function. This is far more general than requiring a gradient to the nearest invalid region. However, since this is an admittedly less informative feature space, a more powerful model is required. Random forest are one such example. Effectively - they are an ensemble of decision trees which use the bitmap information to remember failure cases.

Features	Learner	CHOMP	RRT*-Tunnel	RRT*	RRT*-Obs
HOG Dense	Linear	0.055	0.227	0.049	0.131
HOG Coarse	Linear	0.100	0.171	0.057	0.178
Pyramid Dense	Linear	0.203	0.411	0.149	0.278
Pyramid Coarse	Linear	0.503	0.777	0.681	0.688
HOG Dense	Random Forest	0.086	0.214	0.065	0.156
HOG Coarse	Random Forest	0.072	0.171	0.060	0.151
Pyramid Dense	Random Forest	0.089	0.232	0.066	0.153
Pyramid Coarse	Random Forest	0.065	0.172	0.058	0.132
HOG Dense	ChiSquare Kernel	0.063	0.228	0.053	0.130
HOG Coarse	ChiSquare Kernel	0.058	0.158	0.053	0.130
Pyramid Dense	Intersection Kernel	0.108	0.230	0.053	0.130
Pyramid Coarse	ChiSquare Kernel	0.088	0.243	0.100	0.245
Coarse Graph	Graph Kernel	0.170	0.214	0.053	0.130

Figure D.2: Comparison of learners and feature extractors

3. *Graph kernels effectively perform planning* A surprising result is the effectiveness of graph kernels methods. These methods first require computing a coarse connectivity graph on the environment. These graphs are then fed to a kernel function. An example function is the shortest path kernel - it computes the shortest path between all pairs of vertices and then computes a similarity between the two graphs using the statistics of these paths (such as length).

   We found that even with a coarse adjacency graph, the accuracy was sufficiently high. However, computing this kernel is expensive. Not only that, the kernel method is invoking a planning algorithm itself. This observation motivates us to think along the lines of a white-box planner where planning algorithms themselves contribute to creating a feature vector.

D.2 Features and learners used for closed-loop evaluation in Section 5.8

For the closed loop evaluation, since problems were simpler than the experiments, we decided to use a lower dimension context vector. We used a variant of the Bitmap feature extractor. We first divided the workspace into stripes. A stripe is a volume that runs either longitudinally or laterally. We divide the workspace into 7 longitudinal stripes and 5 lateral stripes. For each of these stripes, we uniformly randomly sample points and evaluate if these points are in collision or not. Hence we estimate the density of invalid space in each of these volumes. We then create a volume around start and goal and estimate the density there. This adds 2 more dimensions. We also add the normalized distance between start and goal as well as a unit vector. This gives us a total of 16 dimensions. We use a random forest learner.

References

Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1. ACM, 2004.

Sandip Aine and Maxim Likhachev. Search portfolio with sharing. In *Twenty-Sixth International Conference on Automated Planning and Scheduling*, 2016.

Sandip Aine, Charupriya Sharma, and Maxim Likhachev. Learning to search more efficiently from experience: A multi-heuristic approach. In *Eighth Annual Symposium on Combinatorial Search*, 2015.

Sandip Aine, Siddharth Swaminathan, Venkatraman Narayanan, Victor Hwang, and Maxim Likhachev. Multi-heuristic a. *The International Journal of Robotics Research*, 35(1-3):224–243, 2016.

Baris Akgun and Mike Stilman. Sampling heuristics for optimal motion planning in high dimensions. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2640–2645. IEEE, 2011.

Ron Alterovitz, Sachin Patil, and Anna Derbakova. Rapidly-exploring roadmaps: Weighing exploration vs. refinement in optimal motion planning. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3706–3712. IEEE, 2011.

Shahab Jabbari Arfaee, Sandra Zilles, and Robert C Holte. Learning heuristic functions for large state spaces. *Artificial Intelligence*, 175(16):2075–2098, 2011.

S. Arora, S. Choudhury, D. Althoff, and S. Scherer. Emergency maneuver library - ensuring safe navigation in partially known environments. In *IEEE International Conference on Robotics and Automation, ICRA*, 2015.

Sankalp Arora and Sebastian Scherer. Randomized algorithm for informative path planning with budget constraints. In *ICRA*, 2017.

Oktay Arslan and Panagiotis Tsiotras. Use of relaxation methods in sampling-based algorithms for optimal motion planning. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 2421–2428. IEEE, 2013.

Oktay Arslan and Panagiotis Tsiotras. Machine learning guided exploration for sampling-based motion planning algorithms. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 2646–2652. IEEE, 2015.

Oktay Arslan and Panagiotis Tsiotras. Incremental sampling-based motion planners using policy iteration methods. *arXiv preprint arXiv:1609.05960*, 2016.

Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*, 2017.

John Asmuth and Michael L Littman. Approaching bayes-optimalilty using monte-carlo tree search. In *Proc. 21st Int. Conf. Automat. Plan. Sched., Freiburg, Germany*, 2011.

Bernardo Ávila Pires, Csaba Szepesvari, and Mohammad Ghavamzadeh. Cost-sensitive multiclass classification risk bounds. In *ICML*, 2013.

Ian Baldwin and Paul Newman. Teaching a randomized planner to plan with semantic fields. *Towards Autonomous Robotic Systems*, 2010.

J. Barraquand and J.-C. Latombe. Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 2328–2335, 1991.

Laszlo A. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Systems journal*, 5(2):78–101, 1966.

Richard Bellman. The theory of dynamic programming. Technical report, DTIC Document, 1954.

Stefan Berchtold and Bernhard Glavina. A scalable optimizer for automatically generated manipulator motions. In *Intelligent Robots and Systems' 94.'Advanced Robotic Systems and the Real World', IROS'94. Proceedings of the IEEE/RSJ/GI International Conference on*, volume 3, pages 1796–1802. IEEE, 1994.

Dmitry Berenson, Rosen Diankov, Koichi Nishiwaki, Satoshi Kagami, and James Kuffner. Grasp planning in complex scenes. In *Humanoid Robots, 2007 7th IEEE-RAS International Conference on*, pages 42–48. IEEE, 2007.

Dmitry Berenson, Pieter Abbeel, and Ken Goldberg. A robot path planning framework that learns from experience. In *IEEE International Conference on Robotics and Automation, ICRA*, 2012.

Mohak Bhardwaj, Sanjiban Choudhury, and Sebastian Scherer. Learning heuristic search via imitation. In *CoRL*, 2017.

Avrim Blum. On-line algorithms in machine learning. In *Online algorithms*, pages 306–325. Springer, 1998.

James E Bobrow, Steven Dubowsky, and JS Gibson. Time-optimal control of robotic manipulators along specified paths. *The international journal of robotics research*, 4(3):3–17, 1985.

Robert Bohlin and Lydia E Kavraki. Path planning using lazy prm. In *ICRA*, 2000.

J.-D. Boissonnat and S. Lazard. A polynomial-time algorithm for computing a shortest path of bounded curvature amidst moderate obstacles. In *Proceedings ACM Symposium on Computational Geometry*, pages 242–251, 1996.

Bela Bollobas and Oliver Riordan. *Percolation*. Cambridge University Press, 2006.

Byron Boots, Sajid M Siddiqi, and Geoffrey J Gordon. Closing the learning-planning loop with predictive state representations. *The International Journal of Robotics Research*, 30(7):954–966, 2011.

Allan Borodin, Sandy Irani, Prabhakar Raghavan, and Baruch Schieber. Competitive paging with locality of reference. *Journal of Computer and System Sciences*, 50(2):244–258, 1995.

Ronen I Brafman and Moshe Tennenholtz. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3(Oct):213–231, 2002.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

A. E. Bryson and Y.-C. Ho. *Applied Optimal Control*. Hemisphere Publishing Corp., New York, 1975.

Niv Buchbinder, Joseph Seffi Naor, et al. The design of competitive online algorithms via a primal–dual approach. *Foundations and Trends® in Theoretical Computer Science*, 3(2–3):93–263, 2009.

Brendan Burns. *Exploiting structure: A guided approach to sampling-based robot motion planning*. ProQuest, 2007.

Brendan Burns and Oliver Brock. Sampling-based motion planning using predictive models. In *ICRA*, 2005a.

Brendan Burns and Oliver Brock. Toward optimal configuration space sampling. In *Robotics: Science and Systems*, pages 105–112. Citeseer, 2005b.

John Canny. *The complexity of robot motion planning*. MIT press, 1988a.

John Canny. Some algebraic and geometric computations in pspace. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 460–467. ACM, 1988b.

Kathryn Chaloner and Isabella Verdinelli. Bayesian experimental design: A review. *Statistical Science*, pages 273–304, 1995.

Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daume, and John Langford. Learning to search better than your teacher. In *ICML*, 2015.

Benjamin Charrow, Gregory Kahn, Sachin Patil, Sikang Liu, Ken Goldberg, Pieter Abbeel, Nathan Michael, and Vijay Kumar. Information-theoretic planning with trajectory optimization for dense 3d mapping. In *RSS*, 2015.

Chandra Chekuri and Martin Pal. A recursive greedy algorithm for walks in directed graphs. In *FOCS*, 2005.

Min Chen, Emilio Frazzoli, David Hsu, and Wee Sun Lee. Pomdp-lite for robust robot planning under uncertainty. *arXiv preprint arXiv:1602.04875*, 2016a.

Yuxin Chen, Shervin Javdani, Amin Karbasi, Drew Bagnell, Siddhartha Srinivasa, and Andreas Krause. Submodular surrogates for value of information. In *AAAI*, 2015a.

Yuxin Chen, Shervin Javdani, Amin Karbasi, J. Bagnell, Siddhartha Srinivasa, and Andreas Krause. Submodular surrogates for value of information, 2015b.

Yuxin Chen, S. Hamed Hassani, and Andreas Krause. Near-optimal bayesian active learning with correlated and noisy tests. *CoRR*, abs/1605.07334, 2016b.

Howie M Choset. *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.

Sanjiban Choudhury. Lower and upper bounds for the survival of infinite absorbing markov chains. Technical Report CMU-RI-TR-05-04, Robotics Institute, Pittsburgh, PA, January 2015.

Sanjiban Choudhury and Sebastian Scherer. The dynamics projection filter (DPF) - Real-time nonlinear trajectory optimization using projection operators. In *IEEE International Conference on Robotics and Automation, ICRA*, 2015.

Sanjiban Choudhury, Sebastian Scherer, and Sanjiv Singh. RRT*-AR: Sampling-based alternate routes planning with applications to autonomous emergency landing of a helicopter. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 3947–3952. IEEE, 2013.

Sanjiban Choudhury, Sankalp Arora, and Sebastian Scherer. The planner ensemble and trajectory executive: A high performance motion planning system with guaranteed safety. In *AHS 70th Annual Forum, Montreal, Quebec, Canada*, May 2014.

Sanjiban Choudhury, Sankalp Arora, and Sebastian Scherer. The Planner Ensemble: Motion planning by executing diverse algorithms. In *IEEE International Conference on Robotics and Automation, ICRA*, 2015a.

Sanjiban Choudhury, Sebastian Scherer, and Andrew Bagnell. Theoretical limits of speed and resolution for kinodynamic planning in a poisson forest. In *Proceedings of Robotics: Science and Systems*, Rome, Italy, July 2015b. .

Sanjiban Choudhury, Jonathan D. Gammell, Timothy D. Barfoot, Siddhartha Srinivasa, and Sebastian Scherer. Regionally accelerated batch informed trees (RABIT*): A framework to integrate local information into optimal path planning. In IEEE, editor, *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016a.

Sanjiban Choudhury, Mohak Bhardwaj, Sankalp Arora, Ashish Kapoor, Gireeja Ranade, Sebastian Scherer, and Debadeepta Dey. Data-driven planning via imitation learning. *arXiv preprint arXiv:1711.06391*, 2017a.

Sanjiban Choudhury, Shervin Javdani, Siddhartha Srinivasa, and Sebastian Scherer. Near-optimal edge evaluation in explicit generalized binomial graphs. In *NIPS*, 2017b.

Sanjiban Choudhury, Shervin Javdani, Siddhartha Srinivasa, and Sebastian Scherer. Near-optimal edge evaluation in explicit generalized binomial graphs. *arXiv preprint arXiv:1706.09351*, 2017c.

Sanjiban Choudhury, Ashish Kapoor, Gireeja Ranade, and Debadeepta Dey. Learning to gather information via imitation. In *ICRA*, 2017d.

Sanjiban Choudhury, Ashish Kapoor, Gireeja Ranade, Debadeepta Dey, and Sebastian Scherer. Adaptive information gathering via imitation learning. In *RSS*, 2017e.

Sanjiban Choudhury, Siddhartha Srinivasa, and Sebastian Scherer. Bayesian active edge evaluation on expensive graphs. *arXiv preprint arXiv:1711.07329*, 2017f.

Shushman Choudhury, Christopher M Dellin, and Siddhartha S Srinivasa. Pareto-optimal search over configuration space beliefs for anytime motion planning. In *IROS*, 2016b.

Shushman Choudhury, Oren Salzman, Sanjiban Choudhury, Christopher M Dellin, and Siddhartha S Srinivasa. Anytime motion planning on large dense roadmaps with expensive edge evaluations. *arXiv preprint arXiv:1711.04040*, 2017g.

Shushman Choudhury, Oren Salzman, Sanjiban Choudhury, and Siddhartha S Srinivasa. Densification strategies for anytime motion planning over large dense roadmaps. In *ICRA*, 2017h.

Benjamin Cohen, Mike Phillips, and Maxim Likhachev. Planning single-arm manipulations with n-arm robots. In *Eigth Annual Symposium on Combinatorial Search*, 2015.

William W Cohen and Vitor R Carvalho. Stacked sequential learning. In *International Joint Conference on Artificial Intelligence*. Citeseer, 2005.

Hugh Cover, Sanjiban Choudhury, Sebastian Scherer, and Sanjiv Singh. Sparse tangential network (spartan): Motion planning for micro aerial vehicles. In *International Conference on Robotics and Automation*, May 2013.

Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893. IEEE, 2005.

Sébastien Dalibard and Jean-Paul Laumond. Linear dimensionality reduction in random motion planning. *The International Journal of Robotics Research*, page 0278364911403335, 2011.

Sanjoy Dasgupta. Analysis of a greedy active learning strategy. In *NIPS*, 2004.

Hal Daumé, John Langford, and Daniel Marcu. Search-based structured prediction. *Machine learning*, 75(3):297–325, 2009.

Christopher Dellin. *Completing Manipulation Tasks Efficiently in Complex Environments*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, September 2016.

Christopher M Dellin and Siddhartha S Srinivasa. A unifying formalism for shortest path problems with expensive edge evaluations via lazy best-first search over paths with edge selectors. In *Proceedings of the Twenty-Sixth International Conference on International Conference on Automated Planning and Scheduling*, pages 459–467. AAAI Press, 2016.

Christopher M Dellin, Kyle Strabala, G Clark Haynes, David Stager, and Siddhartha S Srinivasa. Guided manipulation planning at the darpa robotics challenge trials. In *Experimental Robotics*, 2016.

Debadeepta Dey. Predicting sets and lists: Theory and practice. Technical report, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 2015.

Debadeepta Dey, Tian Yu Liu, Boris Sofman, and James Andrew Bagnell. Efficient optimization of control libraries. In *AAAI*, volume 1, page 4, 2012.

Debadeepta Dey, Tian Yu Liu, Martial Hebert, and J Andrew Bagnell. Contextual sequence prediction with application to control library optimization. In *Robotics Science and Systems, RSS*, 2013.

Rosen Diankov and James Kuffner. Randomized statistical path planning. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1–6. IEEE, 2007.

Andrew Dobson and Kostas E Bekris. A study on the finite-time near-optimality properties of sampling-based motion planners. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1236–1241. IEEE, 2013.

Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. Practical search techniques in path planning for autonomous driving. *AAAI*, 2008.

Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. Path planning for autonomous vehicles in unknown semi-structured environments. *The International Journal of Robotics Research*, 29(5):485–501, 2010.

Anca Dragan, Geoffrey Gordon, and Siddhartha Srinivasa. Learning from experience in manipulation planning: Setting the right goals. In *Proceedings of the International Symposium on Robotics Research (ISRR) 2011*, July 2011.

Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338, 2016.

Geetesh Dubey, Sankalp Arora, and Sebastian Scherer. Droan-disparity-space representation for obstacle avoidance. IROS, 2017.

Lester E Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of mathematics*, 79(3):497–516, 1957.

Vishal Dugar, Sanjiban Choudhury, and Sebastian Scherer. A kite in the wind: Smooth trajectory optimization in a moving reference frame. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017a.

Vishal Dugar, Sanjiban Choudhury, and Sebastian Scherer. Smooth trajectory optimization in wind: First results on a full-scale helicopter. In *AHS Forum, Vol 73*. IEEE, 2017b.

Leah Epstein, Rob Van Stee, and Tami Tamir. Paging with request sets. *Theory of Computing Systems*, 44(1):67–81, 2009.

Lawrence H Erickson and Steven M LaValle. Survivability: Measuring and ensuring path diversity. In *IEEE International Conference on Robotics and Automation, ICRA*, 2009.

Dave Ferguson and Anthony Stentz. Anytime rrts. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5369–5375. IEEE, 2006.

Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International Conference on Machine Learning*, pages 49–58, 2016.

T. Fraichard and J.-M. Ahuactzin. Smooth path planning for cars. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 3722–3727, 2001.

T. Fraichard and A. Scheuer. From Reeds and Shepp's to continuous-curvature paths. *IEEE Transactions on Robotics*, 20(6):1025–1035, December 2004.

Alan Frieze and Michał Karoński. *Introduction to random graphs*. Cambridge Press, 2015.

Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. Informed rrt*: Optimal incremental path planning focused through an admissible ellipsoidal heuristic. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2014.

Jonathan D. Gammell, Siddhartha S. Srinivasa, and Timothy D. Barfoot. Batch Informed Trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In *IEEE International Conference on Robotics and Automation, ICRA*, 2015.

Caelan Reed Garrett, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Learning to rank for synthesizing planning heuristics. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pages 3089–3095. AAAI Press, 2016.

Roland Geraerts and Mark H Overmars. Creating high-quality paths for motion planning. *The International Journal of Robotics Research*, 26(8):845–863, 2007.

Edward N Gilbert. Random plane networks. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):533–543, 1961.

C. Goerzen, Z. Kong, and B. Mettler. A survey of motion planning algorithms from the perspective of autonomous uav guidance. *Journal of Intelligent & Robotic Systems*, 57(1):65–100, 2010.

Corey Goldfeder, Matei Ciocarlie, Jaime Peretzman, Hao Dang, and Peter K Allen. Data-driven grasping with partial sensor data. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 1278–1283. IEEE, 2009.

Daniel Golovin and Andreas Krause. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *Journal of Artificial Intelligence Research*, 2011.

Daniel Golovin, Andreas Krause, and Debajyoti Ray. Near-optimal bayesian active learning with noisy observations. In *NIPS*, 2010.

Colin Green and Alonzo Kelly . Toward optimal sampling in the space of paths. In *13th International Symposium of Robotics Research*, November 2007.

Geoffrey Grimmett. *What is Percolation?* Springer, 1999.

Alex Grubb and Drew Bagnell. Speedboost: Anytime prediction with uniform near-optimality. In *Artificial Intelligence and Statistics*, pages 458–466, 2012.

Anupam Gupta, Viswanath Nagarajan, and R Ravi. Approximation algorithms for optimal decision trees and adaptive tsp problems. In *International Colloquium on Automata, Languages, and Programming*, 2010.

Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. In *CVPR*, 2017.

J. H. Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik*, 2:84–90, 1960.

Nikolaus Hansen. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. on Systems Science and Cybernetics*, 1968.

Kris Hauser. Lazy collision checking in asymptotically-optimal motion planning. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 2951–2957. IEEE, 2015.

Kris Hauser and Victor Ng-Thow-Hing. Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2493–2498. IEEE, 2010.

Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. In *2015 AAAI Fall Symposium Series*, 2015.

Lionel Heng, Lorenz Meier, Petri Tanskanen, Friedrich Fraundorfer, and Marc Pollefeys. Autonomous obstacle avoidance and maneuvering on a vision-guided mav using on-board processing. In *Robotics and automation (ICRA), 2011 IEEE international conference on*, pages 2472–2477. IEEE, 2011.

Lionel Heng, Alkis Gotovos, Andreas Krause, and Marc Pollefeys. Efficient visual exploration and coverage with a micro aerial vehicle in unknown environments. In *ICRA*, 2015.

Jörg Hoffmann and Bernhard Nebel. The ff planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

Geoffrey A Hollinger and Gaurav S Sukhatme. Sampling-based motion planning for robotic information gathering. In *RSS*, 2013.

Geoffrey A Hollinger, Urbashi Mitra, and Gaurav S Sukhatme. Active classification: Theory and application to underwater inspection. *arXiv preprint arXiv:1106.5829*, 2011.

Geoffrey A Hollinger, Brendan Englot, Franz S Hover, Urbashi Mitra, and Gaurav S Sukhatme. Active planning for underwater inspection and the benefit of adaptivity. *IJRR*, 2012.

Ronald A Howard. Information value theory. *IEEE Tran. Systems Science Cybernetics*, 1966.

Thomas M. Howard. *Adaptive Model-predictive Motion Planning for Navigation in Complex Environments*. PhD thesis, Pittsburgh, PA, USA, 2009a. AAI3405180.

Thomas M Howard. *Adaptive model-predictive motion planning for navigation in complex environments*. Carnegie Mellon University, 2009b.

D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. *International Journal Computational Geometry & Applications*, 4:495–512, 1999a.

David Hsu, J-C Latcombe, and Stephen Sorkin. Placing a robot manipulator amid obstacles for optimized execution. In *Assembly and Task Planning, 1999.(ISATP'99) Proceedings of the 1999 IEEE International Symposium on*, pages 280–285. IEEE, 1999b.

David Hsu, Robert Kindel, Jean-Claude Latombe, and Stephen Rock. Randomized kinodynamic motion planning with moving obstacles. *The International Journal of Robotics Research*, 21(3):233–255, 2002.

David Hsu, Gildardo Sánchez-Ante, and Zheng Sun. Hybrid prm sampling with a cost-sensitive adaptive strategy. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 3874–3880. IEEE, 2005.

Jinwook Huh and Daniel D Lee. Learning high-dimensional mixture models for fast collision detection in rapidly-exploring random trees. In *ICRA*, 2016.

Jeong hwan Jeon, Raghvendra V Cowlagi, Steven C Peters, Sertac Karaman, Emilio Frazzoli, Panagiotis Tsiotras, and Karl Iagnemma. Optimal motion planning with the half-car dynamical model for autonomous high-speed driving. In *2013 American Control Conference*, pages 188–193. IEEE, 2013.

Myung Hwangbo, James Kuffner, and Takeo Kanade. Efficient two-phase 3d motion planning for small fixed-wing uavs. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 1035–1041. IEEE, 2007.

Jeffrey Ichnowski and Ron Alterovitz. Parallel sampling-based motion planning with superlinear speedup. In *IROS*, pages 1206–1212, 2012.

Brian Ichter, James Harrison, and Marco Pavone. Learning sampling distributions for robot motion planning. *arXiv preprint arXiv:1709.05448*, 2017.

Fahad Islam, Venkatraman Narayanan, and Maxim Likhachev. Dynamic multi-heuristic a. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2376–2382. IEEE, 2015.

Stefan Isler, Reza Sabzevari, Jeffrey Delmerico, and Davide Scaramuzza. An information gain formulation for active volumetric 3d reconstruction. In *ICRA*, 2016.

Rishabh K Iyer and Jeff A Bilmes. Submodular optimization with submodular cover and submodular knapsack constraints. In *NIPS*, 2013.

P. Jacobs and J. Canny. Planning smooth paths for mobile robots. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 2–7, 1989.

Léonard Jaillet, Juan Cortés, and Thierry Siméon. Sampling-based path planning on configuration-space costmaps. *IEEE Transactions on Robotics*, 26(4):635–646, 2010.

Lucas Janson, Brian Ichter, and Marco Pavone. Deterministic sampling-based motion planning: Optimality, complexity, and performance. *arXiv preprint arXiv:1505.00023*, 2015a.

Lucas Janson, Edward Schmerling, Ashley Clark, and Marco Pavone. Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *The International journal of robotics research*, page 0278364915577958, 2015b.

Shervin Javdani, Matthew Klingensmith, J. Andrew (Drew) Bagnell, Nancy Pollard , and Siddhartha Srinivasa. Efficient touch based localization through submodularity. In *ICRA*, 2013.

Shervin Javdani, Yuxin Chen, Amin Karbasi, Andreas Krause, Drew Bagnell, and Siddhartha Srinivasa. Near optimal bayesian active learning for decision making. In *AISTATS*, 2014a.

Shervin Javdani, Yuxin Chen, Amin Karbasi, Andreas Krause, J. Andrew (Drew) Bagnell, and Siddhartha Srinivasa. Near optimal bayesian active learning for decision making. In *AISTATS*, 2014b.

Shervin Javdani, Siddhartha S Srinivasa, and J Andrew Bagnell. Shared autonomy via hindsight optimization. In *RSS*, 2015.

Nikolay Jetchev and Marc Toussaint. Fast motion planning from experience: trajectory prediction for speeding up movement generation. *Autonomous Robots*, 34(1-2):111–127, 2013.

Sergio Jiménez, Tomás De La Rosa, Susana Fernández, Fernando Fernández, and Daniel Borrajo. A review of machine learning for automated planning. *The Knowledge Engineering Review*, 27(04):433–467, 2012.

Dongwon Jung and Panagiotis Tsiotras. Bank-to-turn control for a small uav using backstepping and parameter adaptation. *Jung*, 2008.

Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 1998.

Gregory Kahn, Tianhao Zhang, Sergey Levine, and Pieter Abbeel. Plato: Policy learning using adaptive trajectory optimization. In *ICRA*, 2017.

Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal. Stomp: Stochastic trajectory optimization for motion planning. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 4569–4574. IEEE, 2011.

S. Karaman and E. Frazzoli. Incremental sampling-based algorithms for optimal motion planning. In *Proc. Robotics: Science and Systems*, 2010.

S. Karaman and E. Frazzoli. High-speed flight in an ergodic forest. *Arxiv preprint arXiv:1202.0253*, 2012a.

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

Sertac Karaman and Emilio Frazzoli. High-speed motion with limited sensing range in a poisson forest. In *CDC*, pages 3735–3740, 2012b.

Sertac Karaman and Emilio Frazzoli. Sampling-based optimal motion planning for non-holonomic dynamical systems. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 5041–5047. IEEE, 2013.

Peter Karkus, David Hsu, and Wee Sun Lee. Qmdp-net: Deep learning for planning under partial observability. *arXiv preprint arXiv:1703.06692*, 2017.

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

Michael J Kearns, Yishay Mansour, and Andrew Y Ng. Approximate planning in large pomdps via reusable trajectories. In *Advances in Neural Information Processing Systems*, pages 1001–1007, 2000.

A. Kelly and B. Nagy. Reactive nonholonomic trajectory generation via parametric optimal control. *International Journal of Robotics Research*, 22(7-8):583–601, 2003.

F. Kendoul. Survey of advances in guidance, navigation, and control of unmanned rotorcraft systems. *Journal of Field Robotics*, 2012.

Scott Kiesel, Ethan Burns, and Wheeler Ruml. Abstraction-guided sampling for motion planning. In *SOCS*, 2012.

Ross A Knepper and Matthew T Mason. Empirical sampling of path sets for local area motion planning. In *Experimental Robotics*, pages 451–462. Springer, 2009.

Ross A Knepper and Matthew T Mason. Real-time informed path sampling for motion planning search. *The International Journal of Robotics Research*, 31(11):1231–1250, 2012.

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

Sven Koenig, Maxim Likhachev, and David Furcy. Lifelong planning aâĹŮ. *Artificial Intelligence*, 155 (1):93–146, 2004.

Igor Kononenko. Machine learning for medical diagnosis: History, state of the art and perspective. *Artificial Intelligence in Medicine*, 2001.

T John Koo, Yi Ma, and S Shankar Sastry. Nonlinear control of a helicopter based unmanned aerial vehicle model. 2001.

Michael Koval, Nancy Pollard, and Siddhartha Srinivasa. Pre- and post-contact policy decomposition for planar contact manipulation under uncertainty. In *RSS*, 2014.

Andreas Krause and Daniel Golovin. Submodular function maximization. *Tractability: Practical Approaches to Hard Problems*, 2012.

Andreas Krause and Carlos Guestrin. Near-optimal observation selection using submodular functions. In *AAAI*, 2007.

Andreas Krause and Carlos Guestrin. Optimal value of information in graphical models. *Journal of Artificial Intelligence Research*, 35:557–591, 2009.

Andreas Krause, Jure Leskovec, Carlos Guestrin, Jeanne VanBriesen, and Christos Faloutsos. Efficient sensor placement optimization for securing large water distribution networks. *Journal of Water Resources Planning and Management*, 134(6):516–526, 2008.

James J Kuffner and Steven M LaValle. RRT-Connect: An efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation, ICRA*, 2000.

Hanna Kurniawati and David Hsu. Workspace-based connectivity oracle: An adaptive sampling strategy for prm planning. In *Algorithmic Foundation of Robotics VII*, pages 35–51. Springer, 2008.

Hanna Kurniawati, David Hsu, and Wee Sun Lee. Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In *RSS*, 2008.

Lionel Lapierre, Rene Zapata, and Pascal Lepinay. Combined path-following and obstacle avoidance control of a wheeled robot. *The International Journal of Robotics Research*, 26(4):361–375, 2007.

Jean-Claude Latombe. *Robot motion planning*. Springer, 1991.

B. Lau, C. Sprunk, and W. Burgard. Improved updating of euclidean distance maps and voronoi diagrams. In *IEEE Intl. Conf. on Intelligent Robots and Systems (IROS), Taipei, Taiwan*, 2010.

J.-P. Laumond. Trajectories for mobile robots with kinematic and environment constraints. In *Proceedings International Conference on Intelligent Autonomous Systems*, pages 346–354, 1986.

J.-P. Laumond. *Robot Motion Planning and Control*. Springer-Verlag, Berlin, 1998. Available online at http://www.laas.fr/∼jpl/book.html.

Jean-Paul Laumond, S Sekhavat, and F Lamiraux. Guidelines in nonholonomic motion planning for mobile robots. In *Robot motion planning and control*, pages 1–53. Springer, 1998.

S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

S.M. LaValle. Rapidly-exploring random trees a new tool for path planning. 1998.

Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 2169–2178. IEEE, 2006.

Sergey Levine and Vladlen Koltun. Guided policy search. In *ICML*, 2013.

Hui Li, Xuejun Liao, and Lawrence Carin. Multi-task reinforcement learning in partially observable stochastic environments. *Journal of Machine Learning Research*, 2009.

Jiwei Li, Will Monroe, Alan Ritter, Michel Galley, Jianfeng Gao, and Dan Jurafsky. Deep reinforcement learning for dialogue generation. *arXiv preprint arXiv:1606.01541*, 2016.

Yanbo Li, Zakary Littlefield, and Kostas E Bekris. Sparse methods for efficient asymptotically optimal kinodynamic planning. In *Algorithmic Foundations of Robotics XI*, pages 263–282. Springer, 2015.

Z. Li and J. F. Canny. *Nonholonomic Motion Planning*. Kluwer, Boston, MA, 1993.

Thomas M Liggett. Survival of discrete time growth models, with applications to oriented percolation. *The Annals of Applied Probability*, pages 613–636, 1995.

Maxim Likhachev and Dave Ferguson. Planning long dynamically feasible maneuvers for autonomous vehicles. *The International Journal of Robotics Research*, 28(8):933–945, 2009.

Zhan Wei Lim, David Hsu, and Wee Sun Lee. Adaptive stochastic optimization: From sets to paths. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *NIPS*. 2015.

Zhan Wei Lim, David Hsu, and Wee Sun Lee. Adaptive informative path planning in metric spaces. *IJRR*, 2016.

Stephen R Lindemann and Steven M LaValle. Multiresolution approach for motion planning under differential constraints. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 139–144. IEEE, 2006.

Michael L Littman and Richard S Sutton. Predictive representations of state. In *Advances in neural information processing systems*, pages 1555–1561, 2002.

Michael L. Littman, Anthony R. Cassandra, and Leslie Pack Kaelbling. Learning policies for partially observable environments: Scaling up. In *ICML*, 1995.

Miao Liu, Xuejun Liao, and Lawrence Carin. Online expectation maximization for reinforcement learning in pomdps. In *IJCAI*, 2013.

Ryan Luna, Ioan A Şucan, Mark Moll, and Lydia E Kavraki. Anytime solution optimization for sampling-based motion planning. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 5068–5074. IEEE, 2013.

Brian MacAllister, Jonathan Butzke, Alex Kushleyev, Harsh Pandey, and Maxim Likhachev. Path planning for non-circular micro aerial vehicles in constrained environments. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 3933–3940. IEEE, 2013.

Omid Madani, Steve Hanks, and Anne Condon. On the undecidability of probabilistic planning and related stochastic optimization problems. *Artificial Intelligence*, 147(1-2):5–34, 2003.

David A McAllester and Satinder Singh. Approximate planning for factored pomdps using belief state simplification. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 409–416. Morgan Kaufmann Publishers Inc., 1999.

Timothy G McGee, Stephen Spry, and J Karl Hedrick. Optimal path planning in a constant wind with a bounded turning rate. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, pages 1–11. Reston, VA, 2005.

Nicolas Meuleau, Christian Plaunt, David E Smith, and Tristan B Smith. An emergency landing planner for damaged aircraft. In *IAAI*, 2009.

Alain Micaelli, Claude Samson, et al. Trajectory tracking for unicycle-type and two-steering-wheels mobile robots. 1993.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 2015.

Kartik Mohta, Michael Watterson, Yash Mulgaonkar, Sikang Liu, Chao Qu, Anurag Makineni, Kelsey Saulnier, Ke Sun, Alex Zhu, Jeffrey Delmerico, et al. Fast, autonomous flight in gps-denied and cluttered environments. *Journal of Field Robotics*, 35(1):101–120, 2018.

Marco Morales, Lydia Tapia, Roger Pearce, Samuel Rodriguez, and Nancy M Amato. A machine learning approach for feature-sensitive motion planning. In *Algorithmic Foundations of Robotics VI*, pages 361–376. Springer, 2005.

R. M. Murray, M. Rathinam, and W. M. Sluis. Differential flatness of mechanical control systems. In *Proceedings ASME International Congress and Exposition*, 1995.

S Muthukrishnan and Gopal Pandurangan. The bin-covering technique for thresholding random geometric graph properties. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 989–998. Society for Industrial and Applied Mathematics, 2005.

Michael E Napoli, Harel Biggie, and Thomas M Howard. Learning models for predictive adaptation in state lattices. In *Field and Service Robotics*, pages 285–300. Springer, 2018.

Venkatraman Narayanan and Maxim Likhachev. Heuristic search on graphs with existence priors for expensive-to-evaluate edges. In *ICAPS*, 2017.

Venkatraman Narayanan, Sandip Aine, and Maxim Likhachev. Improved multi-heuristic a* for searching with uncalibrated heuristics. In *Eighth Annual Symposium on Combinatorial Search*, 2015.

Jauwairia Nasir, Fahad Islam, Usman Malik, Yasar Ayaz, Osman Hasan, Mushtaq Khan, and Mannan Saeed Muhammad. Rrt*-smart: A rapid convergence implementation of rrt. *International Journal of Advanced Robotic Systems*, 10, 2013.

Erik Nelson and Nathan Michael. Information-theoretic occupancy grid compression for high-speed information-based exploration. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 4976–4982. IEEE, 2015.

Andrew Y Ng and Michael Jordan. Pegasus: A policy search method for large mdps and pomdps. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 406–415. Morgan Kaufmann Publishers Inc., 2000.

Harald Niederreiter. Quasi-monte carlo methods and pseudo-random numbers. *Bulletin of the American Mathematical Society*, 84(6):957–1041, 1978.

Christian L Nielsen and Lydia E Kavraki. A 2 level fuzzy prm for manipulation planning. In *IROS*, 2000.

Stephen Nuske, Sanjiban Choudhury, Sezal Jain, Andrew Chambers, Luke Yoder, Sebastian Scherer, Lyle Chamberlain, Hugh Cover, and Sanjiv Singh. Autonomous exploration and motion planning for an unmanned aerial vehicle navigating rivers. *Journal of Field Robotics*, 32(8):1141–1162, 2015.

Peyman Oreizy, Michael M Gorlick, Richard N Taylor, Dennis Heimbigner, Gregory Johnson, Nenad Medvidovic, Alex Quilici, David S Rosenblum, and Alexander L Wolf. An architecture-based approach to self-adaptive software. *IEEE Intelligent systems*, 14(3):54–62, 1999.

Michael Otte and Nikolaus Correll. C-forest: parallel shortest path planning with superlinear speedup. *Robotics, IEEE Transactions on*, 29(3):798–806, 2013.

Michael Otte, William Silva, and Eric Frew. Any-time path-planning: Time-varying wind field+ moving obstacles. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 2575–2582. IEEE, 2016.

Michael W Otte. A survey of machine learning approaches to robotic path-planning. 2009.

Ugur Ozdemir, Yucel Orkut Aktas, Aslihan Vuruskan, Yasin Dereli, Ahmed Farabi Tarhan, Karaca Demirbag, Ahmet Erdem, Ganime Duygu Kalaycioglu, Ibrahim Ozkol, and Gokhan Inalhan. Design of a commercial hybrid vtol uav system. *Journal of Intelligent & Robotic Systems*, 74(1-2):371–393, 2014.

Brian Paden and Emilio Frazzoli. A generalized label correcting method for optimal kinodynamic motion planning. *arXiv preprint arXiv:1607.06966*, 2016.

Brian Paden, Valerio Varricchio, and Emilio Frazzoli. Verification and synthesis of admissible heuristics for kinodynamic motion planning. *IEEE Robotics and Automation Letters*, 2(2):648–655, 2017.

Luigi Palmieri and Kai O Arras. Distance metric learning for rrt-based motion planning with constant-time inference. In *IEEE International Conference on Robotics and Automation, ICRA*, 2015.

Luigi Palmieri, Sven Koenig, and Kai O Arras. Rrt-based nonholonomic motion planning using any-angle path biasing. 2016.

Jia Pan, Sachin Chitta, and Dinesh Manocha. Faster sample-based motion planning using instance-based learning. In *WAFR*. Springer Verlag, 2012.

Jia Pan, Zhuo Chen, and Pieter Abbeel. Predicting initialization effectiveness for trajectory optimization. In *IEEE International Conference on Robotics and Automation, ICRA*, 2014.

Christos H Papadimitriou and John N Tsitsiklis. The complexity of markov decision processes. *Mathematics of operations research*, 12(3):441–450, 1987.

Judea Pearl. Heuristics: intelligent search strategies for computer problem solving. 1984.

Mathew Penrose. *Random geometric graphs*. Oxford University Press, 2003.

Guilherme AS Pereira, Sanjiban Choudhury, and Sebastian Scherer. A framework for optimal repairing of vector field-based motion plans. In *Unmanned Aircraft Systems (ICUAS), 2016 International Conference on*, pages 261–266. IEEE, 2016.

Sven Mikael Persson and Inna Sharf. Sampling-based a* algorithm for robot path-planning. *The International Journal of Robotics Research*, 33(13):1683–1708, 2014.

Jan Peters and Stefan Schaal. Policy gradient methods for robotics. In *IROS*, 2006.

Mike Phillips, Benjamin J Cohen, Sachin Chitta, and Maxim Likhachev. E-graphs: Bootstrapping planning with experience graphs. In *Robotics: Science and Systems*, volume 5, 2012.

Mike Phillips, Venkatraman Narayanan, Sandip Aine, and Maxim Likhachev. Efficient search with an ensemble of heuristics. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence. AAAI Press (to appear)*, 2015.

Mihail Pivtoraiko, Ross A Knepper, and Alonzo Kelly. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, 26(3):308–333, 2009.

M Poffald, Y Zhang, and K Hauser. Learning problem space metrics for motion primitive selection. In *IROS Workshop on Machine Learning in Planning and Control of Robot Motion*, 2014.

Raymond W Prouty. *Helicopter performance, stability, and control*. 1995.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Filip Radlinski, Robert Kleinberg, and Thorsten Joachims. Learning diverse rankings with multi-armed bandits. In *Proceedings of the 25th international conference on Machine learning*, pages 784–791. ACM, 2008.

Karthik Raman, Pannaga Shivaswamy, and Thorsten Joachims. Online learning to diversify from implicit feedback. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 705–713. ACM, 2012.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*, 2015.

Nathan Ratliff, Matthew Zucker, J Andrew Bagnell, and Siddhartha Srinivasa. CHOMP: Gradient optimization techniques for efficient motion planning. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 489–494. IEEE, 2009a.

Nathan D Ratliff, David Silver, and J Andrew Bagnell. Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots*, 27(1):25–53, 2009b.

Barak Raveh, Angela Enosh, and Dan Halperin. A little more, a lot better: Improving path quality by a path-merging algorithm. *IEEE Transactions on Robotics*, 27(2):365–371, 2011.

J. A. Reeds and L. A. Shepp. Optimal paths for a car that goes both forwards and backwards. *Pacific Journal of Mathematics*, 145(2):367–393, 1990.

J. Reif and H. Wang. Non-uniform discretization approximations for kinodynamic motion planning. In J.-P. Laumond and M. H. Overmars, editors, *Algorithms for Robotic Motion and Manipulation*, pages 97–112. A.K. Peters, Wellesley, MA, 1997.

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

Markus Rickert, Oliver Brock, and Alois Knoll. Balancing exploration and exploitation in motion planning. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 2812–2817. IEEE, 2008.

Samuel Rodriguez, Shawna Thomas, Roger Pearce, and Nancy M Amato. Resampl: A region-sensitive adaptive motion planner. In *Algorithmic Foundation of Robotics VII*, pages 285–300. Springer, 2008.

Stéphane Ross and Drew Bagnell. Efficient reductions for imitation learning. In *AISTATS*, 2010.

Stephane Ross and J Andrew Bagnell. Reinforcement and imitation learning via interactive no-regret learning. *arXiv preprint arXiv:1406.5979*, 2014.

Stéphane Ross, Joelle Pineau, Sébastien Paquet, and Brahim Chaib-Draa. Online planning algorithms for pomdps. *JAIR*, 2008.

Stéphane Ross, Geoffrey J Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, volume 1, page 6, 2011.

Stephane Ross, Jiaji Zhou, Yisong Yue, Debadeepta Dey, and J Andrew Bagnell. Learning policies for contextual submodular prediction. *arXiv preprint arXiv:1305.2532*, 2013.

M Sahini and M Sahimi. *Applications of percolation theory*. CRC Press, 1994.

Oren Salzman and Dan Halperin. Asymptotically-optimal motion planning using lower bounds on cost. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4167–4172. IEEE, 2015.

Davide Scaramuzza, Michael C Achtelik, Lefteris Doitsidis, Fraundorfer Friedrich, Elias Kosmatopoulos, Agostino Martinelli, Markus W Achtelik, Margarita Chli, Savvas Chatzichristofis, Laurent Kneip, et al. Vision-controlled micro flying robots: from system design to autonomous navigation and mapping in gps-denied environments. *IEEE Robotics & Automation Magazine*, 21(3):26–40, 2014.

A. Scheuer and C. Laugier. Planning sub-optimal and continuous-curvature paths for car-like robots. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 25–31, 1998.

John Schulman, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, and Pieter Abbeel. Finding locally optimal, collision-free trajectories with sequential convex optimization. In *Robotics: Science and Systems*, volume 9, pages 1–10. Citeseer, 2013.

Sepanta Sekhavat, Petr Svestka, Jean-Paul Laumond, and Mark H Overmars. Multilevel path planning for nonholonomic robots using semiholonomic subsystems. *The international journal of robotics research*, 17(8):840–857, 1998.

Martin Seleck, Petr Vana, Milan Rollo, and Tomáš Meiser. Wind corrections in flight path planning. *Int J Adv Robotic Sy*, 10(248), 2013.

Shaojie Shen, Nathan Michael, and Vijay Kumar. Autonomous indoor 3d exploration with a micro-aerial vehicle. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 9–15. IEEE, 2012.

David Silver and Joel Veness. Monte-carlo planning in large pomdps. In *NIPS*, 2010.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Amarjeet Singh, Andreas Krause, Carlos Guestrin, William Kaiser, and Maxim Batalin. Efficient planning of informative paths for multiple robots. In *IJCAI*, 2007.

Amarjeet Singh, Andreas Krause, and William J. Kaiser. Nonmyopic adaptive informative path planning for multiple robots. In *IJCAI*, 2009.

Daniel D Sleator and Robert E Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

Trey Smith and Reid Simmons. Point-based pomdp algorithms: Improved analysis and implementation. *arXiv preprint arXiv:1207.1412*, 2012.

Adhiraj Somani, Nan Ye, David Hsu, and Wee Sun Lee. Despot: Online pomdp planning with regularization. In *NIPS*, 2013.

Matthijs TJ Spaan and Nikos Vlassis. Perseus: Randomized point-based value iteration for pomdps. *Journal of artificial intelligence research*, 24:195–220, 2005.

Matthew Streeter and Daniel Golovin. An online algorithm for maximizing submodular functions. In *NIPS*, 2009.

J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *IROS*, 2012.

Aleksandr G Sukharev. Optimal strategies of the search for an extremum. *USSR Computational Mathematics and Mathematical Physics*, 11(4):119–137, 1971.

Wen Sun, Arun Venkatraman, Geoffrey J Gordon, Byron Boots, and J Andrew Bagnell. Deeply aggrevated: Differentiable imitation learning for sequential prediction. In *ICML*, 2017.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

Abhijeet Tallavajhula and Sanjiban Choudhury. List prediction for motion planning: Case Studies. Technical Report CMU-RI-TR-15-25, Robotics Institute, Pittsburgh, PA, September 2015.

Abhijeet Tallavajhula, Sanjiban Choudhury , Sebastian Scherer, and Alonzo Kelly . List prediction applied to motion planning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016.

Aviv Tamar, Garrett Thomas, Tianhao Zhang, Sergey Levine, and Pieter Abbeel. Learning from the hindsight plan–episodic mpc improvement. *arXiv preprint arXiv:1609.09001*, 2016.

Xinyu Tang, Jyh-Ming Lien, NM Amato, et al. An obstacle-based rapidly-exploring random tree. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 895–900. IEEE, 2006.

Laszlo Techy. Optimal navigation in planar time-varying flow: ZermeloâĂŹs problem revisited. *Intelligent Service Robotics*, 4(4):271–283, 2011.

Jordan Tyler Thayer, Austin J Dionne, and Wheeler Ruml. Learning inadmissible heuristics during search. In *ICAPS*, 2011.

C. Urmson and R. Simmons. Approaches for heuristically biasing RRT growth. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2003.

Stanislav Uryasev. Conditional value-at-risk: Optimization algorithms and applications. In *Computational Intelligence for Financial Engineering, 2000.(CIFEr) Proceedings of the IEEE/IAFE/INFORMS 2000 Conference on*, pages 49–57. IEEE, 2000.

Jes ús Virseda, Daniel Borrajo, and Vidal Alcázar. Learning heuristic functions for cost-based planning. *Planning and Learning*, page 6, 2013.

Arun Venkatraman, Byron Boots, Martial Hebert, and J Andrew Bagnell. Data as demonstrator with applications to system identification. In *ALR Workshop, NIPS*, 2014.

Paul Vernaza and Daniel D Lee. Learning dimensional descent for optimal motion planning in high-dimensional spaces. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

S Vichy N Vishwanathan, Nicol N Schraudolph, Risi Kondor, and Karsten M Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11(Apr):1201–1242, 2010.

Jacob Walker, Carl Doersch, Abhinav Gupta, and Martial Hebert. An uncertain future: Forecasting from static images using variational autoencoders. In *European Conference on Computer Vision*, pages 835–851. Springer, 2016.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *ICML*, 2016.

Michael Warren, Luis Mejias, Jonathan Kok, Xilin Yang, Felipe Gonzalez, and Ben Upcroft. An automated emergency landing system for fixed-wing aircraft: Planning and control. *Journal of Field Robotics*, 32(8):1114–1140, 2015.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

Dustin J Webb and Jur van den Berg. Kinodynamic rrt*: Asymptotically optimal motion planning for robots with linear dynamics. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 5054–5061. IEEE, 2013.

Matthew S Whalley, Marc D Takahashi, Jay W Fletcher, Ernesto Moralez, LTC Ott, LTC Olmstead, James C Savage, Chad L Goerzen, Gregory J Schulein, Hoyt N Burns, et al. Autonomous black hawk in flight: Obstacle field navigation and landing-site selection on the rascal juh-60a. *Journal of Field Robotics*, 31(4):591–616, 2014.

Christopher Makoto Wilt and Wheeler Ruml. Building a heuristic for greedy search. In *Eighth Annual Symposium on Combinatorial Search*, 2015.

Mariusz Wzorek, Jonas Kvarnström, and Patrick Doherty. Choosing replanning strategies for unmanned aircraft systems. In *International Conference on Automated Planning and Scheduling, ICAPS*, 2010.

Yuehua Xu, Alan Fern, and Sung Wook Yoon. Discriminative learning of beam-search heuristics for planning. In *IJCAI*, pages 2041–2046, 2007.

Yuehua Xu, Alan Fern, and Sungwook Yoon. Learning linear ranking functions for beam search with application to planning. *Journal of Machine Learning Research*, 10(Jul):1571–1610, 2009.

Yuehua Xu, Alan Fern, and Sung Wook Yoon. Iterative learning of weighted rule sets for greedy search. In *ICAPS*, pages 201–208, 2010.

Feng Xue and Panganamala R Kumar. The number of neighbors needed for connectivity of wireless networks. *Wireless networks*, 10(2):169–181, 2004.

Luke Yoder and Sebastian Scherer. *Autonomous Exploration for Infrastructure Modeling with a Micro Aerial Vehicle*, pages 427–440. Springer International Publishing, Cham, 2016. ISBN 978-3-319-27702-8. . URL `http://dx.doi.org/10.1007/978-3-319-27702-8_28`.

Sung Wook Yoon, Alan Fern, and Robert Givan. Learning heuristic functions from relaxed plans. In *ICAPS*, 2006.

Sungwook Yoon, Alan Fern, and Robert Givan. Learning control knowledge for forward search planning. *Journal of Machine Learning Research*, 9(Apr):683–718, 2008.

Takayuki Yoshizumi, Teruhisa Miura, and Toru Ishida. A* with partial expansion for large branching factor problems. In *AAAI/IAAI*, pages 923–929, 2000.

Jingjin Yu, Mac Schwager, and Daniela Rus. Correlated orienteering problem and its application to informative path planning for persistent monitoring tasks. In *IROS*, 2014.

Yisong Yue and Carlos Guestrin. Linear submodular bandits and their application to diversified retrieval. In *Advances in Neural Information Processing Systems*, pages 2483–2491, 2011.

Haifeng Zhang and Yevgeniy Vorobeychik. Submodular optimization with routing constraints, 2016.

T. Zhang, G. Kahn, S. Levine, and P. Abbeel. Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. In *ICRA*, 2016.

Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

Matt Zucker and J Andrew Bagnell. Reinforcement planning: Rl for optimal planners. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1850–1855. IEEE, 2012.

Matt Zucker, James Kuffner, and J Andrew Bagnell. Adaptive workspace biasing for sampling-based planners. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 3757–3762. IEEE, 2008.

Matthew Zucker. A data-driven approach to high level planning. Technical Report CMU-RI-TR-09-42, Robotics Institute, Pittsburgh, PA, January 2009.