ABSTRACT

Title of dissertation: TRAJECTORY PLANNING FOR AUTONOMOUS VEHICLES PERFORMING INFORMATION GATHERING TASKS

Michael J. Kuhlman, Doctor of Philosophy, 2018

Dissertation directed by: Professor Satyandra K. Gupta
Department of Mechanical Engineering

This dissertation investigates mission scenarios for autonomous vehicles in which the objective is to gather information. This includes minimizing uncertainty of a target's estimated location, generating coverage plans to cover an area, or persistent monitoring tasks such as generating informative patrol routes. Information gathering tasks cannot be solved with shortest path planning algorithms since the rewards are path-dependent. Further, in order to deploy such algorithms effectively in the real world, generated plans must safely avoid obstacles, account for the motion uncertainty (e.g. due to swift currents), and constraints on the vehicle's dynamics such as maximum speed/acceleration. This work extends state-of-the-art information gathering algorithms by generating dynamically feasible trajectories for autonomous vehicles that are able to exploit the environment to find higher quality solutions, reducing mission costs. We also reduce mission risk without

sacrificing the amount of information gathered. The focus of this dissertation will be to solve three related information gathering tasks that require generating dynamically feasible trajectories for reliable plan execution.

When searching for targets, minimizing target location uncertainty with autonomous vehicles improves the effectiveness of ground relief crews. We investigate the use of mutual information for efficiently generating long duration multi-pass trajectories to minimize target location uncertainty in natural environments. We develop $\epsilon - admissible$ heuristics to create the $\epsilon - admissible$ Branch and Bound algorithm to gather the most information. Next, we investigate coordination techniques for underwater vehicle teams conducting large-scale geospatial tasks such as adaptive sampling or coverage planning. It is advantageous to exploit the currents of the ocean to increase endurance, which requires accounting for forecast uncertainty. We adapt Monte Carlo Tree Search and Cross Entropy Method to maximize path-dependent reward, and introduce an iterative greedy solver that outperforms state-of-the-art planners. Finally, we investigate persistent monitoring tasks such as sentry patrol routes and monitoring of harmful algae blooms in a littoral environment. Such an automated planner needs to generate collision-free coverage paths by moving waypoints to locations that both minimize path traversal costs and maximize the amount of information gathered along the path. We extend previous Lloyd's based algorithms by factoring in the ocean currents and introduce greedy methods that minimize mission risk while maximizing information gathered.

TRAJECTORY PLANNING FOR AUTONOMOUS VEHICLES PERFORMING INFORMATION GATHERING TASKS

by

Michael Joseph Kuhlman

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2018

Advisory Committee:
Professor Satyandra K. Gupta, Chair/Advisor
Professor Hugh Bruck
Associate Professor Nikhil Chopra
Assistant Professor Mark D. Fuge
Professor Dana S. Nau (Dean's representative)

All that is gold does not glitter,
Not all those who wander are lost;

J. R. R. Tolkien

To my parents and Elizabeth, who were with me every step of the journey.

Acknowledgments

It takes a village to raise a robot. First, I would like to thank committee members Prof. Hugh Bruck, Associate Prof. Nikhil Chopra, Assistant Prof. Mark D. Fuge, and Prof. Dana S. Nau for taking the time to review my dissertation. I could not have asked for a better PhD advisor than Prof. Satyandra K. Gupta. This work was performed in part at the Naval Research Laboratory, funded by the US Department of Defense, Office of Naval Research grant numbers N0001416WX01272, N0001413WX21045 under "Mobile Autonomous Navy Teams for Information Search and Surveillance (MANTISS)." Other portions of this work was funded by NRL Base Funding project Adaptive Real-Time Algorithms for Multiagent Cooperation in Adversarial Environments (ARTAMAC), JON 55-6A68-08. Special thanks to Donald Sofge at Naval Research Laboratory who gave me many opportunities to contribute to various disciplines. I'd like to extend thanks to all my coauthors who helped in all my research endeavors: Dylan Jones, Prof. Geoffrey Hollinger, Krishnanand Kaipa, Petr Švec, and especially Prof. Michael Otte who helped me in my research. I'd also like to thank fellow NRL colleagues Christopher Taylor, William Curran and Victoria Edwards for their help reviewing this dissertation and defense presentation, Jeff Byers for discussions on information theory, James McMahon for discussion on MCTS, Keith Sullivan for help on algorithms development and Corbin Wilhelmi for reviewing various manuscripts. Also, thank you to Alan Schultz, Allen O'Hara, Darrel King, and Danielle Thorp, who kept the Laboratory

for Autonomous Systems Research running smoothly, and thanks to Keith Mattingly and Courtnie Grierson, who helped me negotiate some of the daily challenges when working at NRL.

Many thanks to my colleagues in S.K.'s former lab at UMD: Bruah Shah, Josh Langsfeld, Shaurya Shriyam, Ariyan Kabir, and Galen Mullins for all their help over the years. I'd also like to thank Prof. David Mount for our discussions on computational geometry techniques. I would also like to thank my friends David Meichle, Vidya Raju and Celeste Poley for all the great discussions we've had, and Gustavo Saraiva, Carson Dunbar, and Maice Costa for being there for me through good times and bad.

Finally, I would like to thank those closest to me. Thanks to my parents Diane Duray and Richard Kuhlman who raised me right and gave me their continuing support through all the years. My cat Kooshka helped me maintain perspective throughout the struggle. Also, thanks to Elizabeth Soergel for all her support during my endeavors while helping me with numerous reference and citation questions.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ASV	Autonomous Surface Vessel
B&B	Branch and Bound
BVP	Boundary Value Problem
CEM	Cross Entropy Method
CEMAP	Cross Entropy Method applied to Policies
CEMAS	Cross Entropy Method applied to Action Sequences
DFS	Depth First Search
DP	Dynamic Programming
FOV	Field of View
GIS	Geospatial Information System
GoM	Gulf of Mexico
GP	Gaussian Process
L'OGRE	Locally optimal greedy
LUT	Lookup Table
MAB	Multi-armed Bandit
MAP	Maximum *a posteriori*
MAV	Micro Aerial Vehicle
MC	Monte Carlo
MCTS	Monte Carlo Tree Search
MDP	Markov Decision Process
MLE	Maximum Likelihood estimate
M-TSP	multiagent Traveling Salesman Problem
NCOM	Navy Coastal Ocean Model
NRL	Naval Research Laboratory
ONR	Office of Naval Research
RRT	Rapidly Exploring Random Tree
RV	Random Variable
SGA	Sequential Greedy Allocation
Sto.GA	Stochastic Greedy Ascent
TSP	Traveling Salesman Problem
UAV	Unmanned Aerial Vehicle
UCB	Upper Confidence Bound
UUV	Unmanned Underwter Vehicle
VI	Value Iteration
VP	Voronoi Partition

Chapter 1: Introduction

1.1 Motivation

There exist a wide variety of mission scenarios for autonomous vehicles in which the objective is to gather information. One information gathering task is search and rescue, where the goal is to direct ground crews to more efficiently search for survivors. Another is tracking harmful algae blooms in the ocean. One final information gathering task of interest is persistent monitoring, where vehicles select closed tours for partrolling in a littoral environment.

These missions have many challenges however that must be accounted for when operating in a realistic setting. Challenges such as safely avoiding obstacles, accounting for sensor measurement uncertainty, vehicle constraints (such as maximum speed and acceleration), and motion uncertainty due to environmental effects (gusts of wind or swift currents) significantly compound the problem. In addition, these missions often have a limited budget in the amount of computational resources available, the number of vehicles and the mission duration, so vehicles need to decide where to go to gather as much

information as possible.

To address these unresolved challenges, I developed automated mission planners that are capable of maximizing the information gathered for a given mission while satisfying vehicle dynamics and other mission constraints. This is done by searching in the space of dynamically feasible trajectories and assessing the total information content of a trajectory. This is still a difficult task because information based rewards are path dependent, making many state-of-the-art path planning techniques intractable.

Specifically, we investigate three different information gathering scenarios:

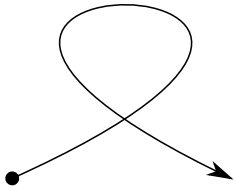
1. Multipass coverage planning to improve search and rescue missions with UAVs

2. Coordinating teams of UUVs for large-scale geospatial tasks

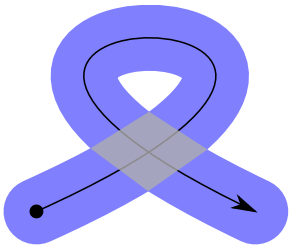
3. Persistent monitoring tasks for ASVs patrolling a littoral environment

1.2 Why Path Dependence of the Reward

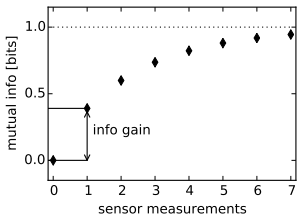
To be able to discuss the goals of this dissertation, we must first outline some of the important concepts, such as path-dependent reward, multipass coverage plans, and mutual information in the context of information gathering systems. Path independence of the reward function differs from the notion of path-independent integrals such as conservation of energy in conserved systems and is defined as follows. Let $c(t) \in \mathcal{C}$ constitute the configuration



(a) An example agent trajectory for time optimal maneuver, where the objective is to minimize the time to move from the start location (circle) to the goal location (arrow) (path independent reward).



(b) An example agent trajectory for coverage sensor planning (path dependent reward). An example application is coverage planning for a vacuum cleaning robot.



(c) The mutual information of noisy, correlated sensor measurements.Multiple observations in the same location increase the amount of information known about the information known, but at a decreasing rate.

Figure 1.1: A comparison of path independent rewards such as time optimal maneuvers (Fig. 1.1(a)) to path dependent rewards such as coverage planning (Fig. 1.1(b)) and maximizing mutual information of multi-pass coverage plans (Fig. 1.1(c)). Path independent rewards (Fig. 1.1(a)) only require local information about the path to evaluate, while path dependent rewards (Fig. 1.1(b)) require global information about the path, making path rewards more difficult to compute.

of an agent at time t in the configuration space $\mathcal{C}$, with $\mathcal{T} = \{c(t),\, t \in [t_a, t_b]\}$ denoting a time indexed curve or trajectory through $\mathcal{C}$. A reward $R(\mathcal{T})$ is path-independent if there exists a function r such that $R(\mathcal{T}) = \int r(c(t))dt$ for all $\mathcal{T} \in \mathscr{T}$, the space of all feasible trajectories. That is, the reward gathered in the next time interval is dependent not only on the present state but on all preceding states in the system trajectory. Many reward/cost functions such as minimizing path length, minimizing fuel costs, minimum time trajectories, and related optimal control problems (even for systems that are not conserved) are "path-independent". Path-independent rewards have a struc-

ture that elicits a principle of optimality, enabling use of techniques such as dynamic programming to speed up the search [1]. Without a more concise definition, a path dependent reward is a reward that is not path-independent and breaks assumptions required for the principle of optimality.

One interpretation of the integral definition of a path independent reward is the reward is only dependent on local information of the path. If the goal is to minimize time of arrival from a start location to a goal location (Fig. 1.1(a)), then letting $\int r(c(t))dt = \int -1dt$ subject to satisfying the boundary conditions defines the path independent reward. For a path dependent reward, consider a robotic vacuum planning to cover an area in Figure 1.1(b), with the sensor radius swept along the path. In many instances, it does not make sense to cover the same area multiple times (grey). Determining that the path loops on itself, however, requires global information of the trajectory, making the reward function path dependent.

In other scenarios, covering the same region multiple times can offer tradeoffs. Consider the task of estimating the probability of a static target occupying a cell. Ideally, one wants the probability of a target being in a cell to be zero or one. Multiple noisy sensor measurements (with false positive and false negative detections) can be combined to reduce the uncertainty. Since the target is either present or absent in the cell and can be represented by a coin toss, there is at most one bit of information (dotted line Fig. 1.1(c)). The mutual information (Fig. 1.1(c), c.f. Section 2.3) is therefore the expected uncertainty reduction of the q correlated sensor measurements when

determining whether or not a target is present in a given cell (black diamond, Fig. 1.1(c)). However, because the sensor measurements are correlated, each subsequent observation contains less and less information. Reward functions such as mutual information are therefore denoted as submodular (c.f. Section 2.4), exhibiting a property of diminishing returns. For a more in depth treatment of this formulation, refer to Section 3.2. This motivates use of developing search and rescue algorithms that can properly reason over the trade-offs of making multiple passes over a region if the presence of targets in the specific region is widely unknown (multi-pass coverage planning, c.f. Chapter 3). Another example of a path-dependent reward includes the mutual information of correlated sensor measurements along a trajectory in a Gaussian Process.

Note that a path-dependent reward function can be transformed into a path-independent problem by transforming the configuration space to encode the space of all paths [2]. For discrete paths and many agents, it is easy to observe the high branching factor and explosion of the number of configurations, making path-independent techniques impractical. Chapters 3 and 4 focus on challenges maximizing path dependent reward, while Chapter 5 formulated persistent monitoring tasks as a locational optimization problem which utilizes path-independent rewards.

1.2.1 Topic 1: Search and Rescue with UAVs

Earthquakes and similar disasters in remote areas pose a challenge for relief efforts when the transportation infrastructure is damaged. Victims in such disasters have a greater chance of survival the sooner they are discovered; conducting an extensive search even just a few hours faster will save many more lives. Delays in search operations can have severe effects on the outcomes of survivors. Survival rates of earthquake survivors drop off dramatically after 48 hours, giving relief crews a narrow time window to search effectively [3, 4]. In the 2013 earthquake that hit Lushan, China, Chinese researchers provided an autonomous rotary-wing UAV to assist search and rescue efforts [4]. In this example, two crew members piloting a UAV were able to search in three hours what a ground crew of 12 people would be able to assess in an entire day, improving the effective speed of the search by a factor of 20.

For the research in Topic 1, consider a scenario where an earthquake or flood causes a building to collapse, trapping survivors who are difficult to access by ground. It would be advantageous to have aerial vehicles autonomously fly at low altitudes navigating over rubble searching for survivors.However, investigating the literature suggests several defficiencies in current techniques that must be resolved to properly deploy autonomous agents in the field. Greedy approaches to informative path planning that only consider all immediate actions are quick to react to changes in the envi-

ronment, but may get trapped [5,6]. However, previous work has shown that for submodular optimization problems, greedy approaches can get to within a constant factor of the optimal solution [7,8]. Similarly, model predictive control based techniques can look multiple time steps into the future [9] further improving performance, but can be trapped similarly. While both discrete [10] and sample based motion planners [11] have been used in information gathering problems and are capable of generating plans of long duration, they have only been applied to relatively simple environments and have not been used to generate multipass coverage plans, with the exception of the authors' previous work in the area [12].

1.2.2 Topic 2: Large Scale Geospatial Tasks with UUVs

Imagine the use of UUVs that can be rapidly and efficiently deployed to various sensing tasks such as tracking oil plumes, intruder detection, or environmental survey. For example, the *Sentry* autonomous underwater vehicle [13] successfully tracked the resulting oil plume from the 2010 Deepwater Horizon disaster. Such a platform however is often transported by surface ship, costing upwards of $30,000 per day [14]. Even small improvements in the amount of information gathered can result in dramatic cost savings. When deployment costs are $30,000 per day, it is rational to spend an additional 24 hours of cloud computing time (costing roughly $110) to find a solution that is on average 0.36% better (c.f. Section 4.6.6). Cost savings come from requiring either fewer assets to complete a mission, or minimizing the time required

to complete the mission by either collecting more informative measurements or eliminating unnecessary mesurements. Put another way, a modest 3% performance improvement for an additional 24hr of compute time per day would result in a cost savings of $767 per day or $280,000 for 365 days of continuous operation.

Some of the main challenges with low-cost long-distance navigation of UUVs are the limitations of and prohibitive cost of underwater positioning systems and underwater communication systems when spanning vast distances, combined with limited battery life limiting propulsion capabilities. We observe that ocean forecast models can in principle be used to control the latitude and longitude of UUVs, even if only the depth of the UUV is actively controlled [15]. One of the profound challenges with this problem is the combined uncertainty of the forecast along with the long duration the vehicles are underwater. Further, research in path planning for Montgolfière balloons on Titan [16,17] found that even having weak propulsion in a strong flow field dramatically reduced transit time to goal locations, being better able to exploit the flow fields.

1.2.3 Topic 3: Persistent Monitoring with ASVs

Consider use of teams of vehicles deployed in the ocean to conduct information gathering tasks in the littoral zone near land. One example application is the monitoring and tracking of algae blooms in the Gulf of Mexico. Algae tends to reside in structures of a given target temperature, and whose re-

gions are highly correlated with the ocean currents [18]. Small autonomous surface vehicles (ASVs) with limited thrust capabilities can significantly reduce operating costs by maximizing endurance. Generating routes for each vehicle to follow that both maximize the amount of information gathered by the team while factoring the ocean currents is a nontrivial task and is still an unresolved challenge. One major challenge to exploiting the ocean currents is being able to account for forecast uncertainty. This requires generating feedback plans that are robust to forecast uncertainty (and the uncertainty of the forecast uncertainty) so that the agents knows how to return to its intended course. Further, swift currents can overpower the control authority of the vehicle, creating regions of imminent collision or failure that must be identified beforehand to stay within the mission area. Consider the scenario where the price of an asset is around $50,000, and the better algorithm reduces the probability of failure/loss per day by 0.01 (e.g. $P_{loss} = 0.03$ vs $P_{loss} = 0.02$). Spending an additional $110 per day for computing better plans would result in an estimated cost savings of $390 per vehicle per day by reducing amortized mission risk costs.

1.2.4 Summary of Topics

While this work aims to solve three distinct tasks, I will now illustrate how these topics are interconnected. First off, the three mentioned topics investigate information gathering algorithms in the context of generating dynamically feasible vehicle plans factoring the physics of the environment. Each

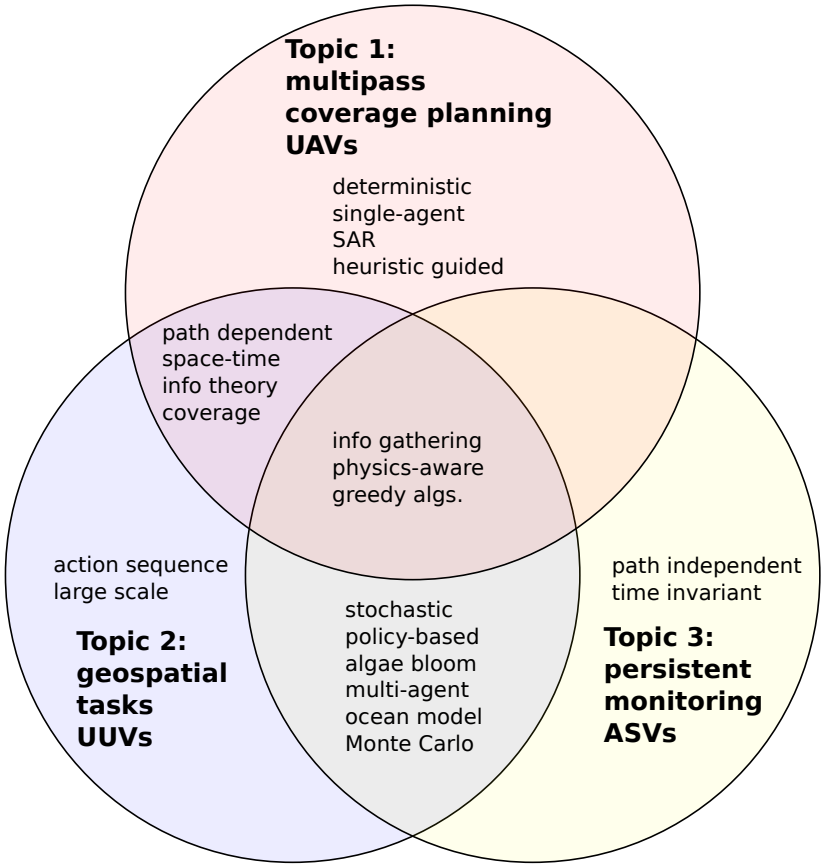


Figure 1.2: Related concepts between the topics in the dissertation.

topic will investigate greedy algorithms; The results in Topics 2 & 3 suggest when greedy methods are advantageous for information gathering tasks. Both Topics 1 and 2 concern themselves with maximizing path dependent rewards and coverage planning. Both topics investigate time varying systems and plan trajectories in space time. Information Theory also plays a critical role in the mentioned topics. Topic 3 differs from Topics 1 and 2 because it focuses on a problem formulation that is path independent. Topics 2 and 3 investigate information gathering challenges related to tracking algae

blooms in the ocean environment using high fidelity ocean forecast models. The mentioned topics utilize stochastic motion models to characterize motion uncertainty, investigate generating feedback policies in the presence of such uncertainty. Topics 2 and 3 consider multiagent systems, and use Monte Carlo simulations to estimate the reward. Figure 1.2 summaries the mentioned overlaps

1.3 Goals

This dissertation sets out to accomplish the following research goals related to trajectory planning for information gathering systems:

1. *Investigate techniques for generating dynamically feasible trajectories respecting platform and environment physics that gather information:* In the majority of research in information gathering systems, the dynamics of the sensing platform are not considered. In scenarios of limited control authority, this omission can cause the planners to generate trajectories that are not feasible. Factoring the platform's actuation capabilities also increases the safety of executing various plans.

    - Planning long duration trajectories requires searching deep solution spaces. Techniques to mitigate this involve generating hierarchical approaches, conducting Monte Carlo rollouts, using more aggressive search techniques to intelligently narrow the search, or

employ greedy methods that only pursue the most immediately rewarding decision.

- Accounting for the environment dynamics (such as ocean currents) can maximize the distance the agent can travel, maximizing how much information is gathered in a given time window. Ideally, physics-aware plans can both increase asset safety while improving total information gathered.
- One major challenges to using high fidelity environment models is the fidelity of the forecast in the future. One must be able to account for forecast uncertainty, and other sources of motion uncertainty. This is critical in avoiding potential obstacle collisions.

2. *Extend state-of-the-art path planning techniques to handle path dependent rewards:* Many path planning problems such as stochastic shortest path (SSP) involve path-independent rewards. Many solvers can exploit this structure using techniques such as Dynamic Programming (DP). However, information gathering tasks are path dependent, making many techniques intractable. Some techniques for path independent rewards can be more readily adapted to path dependent rewards while others cannot.

    - Path dependence of the reward can occur due to the submodularity property, or the property of diminishing returns when sensing measurements are correlated. These tradeoffs can be accurately

modeled using concepts from information theory.

- Such problems with noisy sensor measurements require making multiple passes to further reduce uncertainty, making many current information gathering algorithms unsuitable for the task.
- This makes searching coordination spaces of teams of vehicles even more difficult, as entire trajectories must be coordinated and evaluated for their reward content.

3. *Benchmark testing in simulation of various greedy and globally optimizing techniques in various mission scenarios establishing tradeoffs in solution quality vs. compute time:* Often times in realistic scenarios it is difficult to establish tight bounds on algorithm performance when determining the algorithm that finds the highest quality solution. This is especially true with Sequential Greedy Allocation, which in the most general case only loose performance can be derived yet in practice significantly better solutions can be found. We therefore result to benchmarking a suite of algorithms in different scenarios. We also investigate a deployment cost model to assess the relative costs of computing vs. vehicle deployment, which often favors spending additional computing time to generate more cost effective

Chapter 2: Literature Review

2.1 Overview

Here we discuss relevant material in the literature that solves tasks related to the mentioned topics. Major areas covered include discrete/heuristic guided search techniques, Information Theory, Cross Entropy Method, Submodular function maximization, coverage planning, locational optimization, Markov Decision Processes, and Trajectory Generation. Relevant domains include Search and Rescue, Geospatial Tasks, and Persistent Monitoring.

2.2 Discrete Search and Heuristic Guided Search

Many planning problem representations in robotics discretize the problem into a discrete set of states and actions; the objective of which is to find a sequence of actions that globally maximizes an objective function, while minimizing the computational effort to find the solution. The challenge is overcoming the combinatorial nature of the problem when iterating over all candidate solutions, prioritizing which partial solutions are expanded. Such

techniques employ heuristics[1] such as the "small-is-quick" heuristic (solutions with fewer actions are better) [19]. Another promising quality of discrete search algorithms is they can readily report when the objective is impossible and can exploit many performance tricks when programming computer systems. Many pedagogical treatments of discrete search assume a 4-connected or 8-connected grid, though it is feasible to search abstract graphs such as topological graphs (c.f. Chapter 3) or over a library of predefined feasible maneuvers (c.f. Section 2.12). Abstract graphs typically consist of macro actions or longer duration feasible actions, dramatically decreasing the depth of the search space and improving algorithm performance.

We now identify the discrete search algorithms of relevance to the topics of interest in the context of minimizing path independent costs for paths starting the search from the start state to arrive at a goal state.

The discrete algorithms search the space of solutions by building a tree of partial solutions, storing partial solutions on an open list (solutions with unexplored children) and a closed list (fully explored). Employing different data structures for the open list results in different algorithms, which have different formal properties and qualitative behaviors [20]. Depth First Search utilizes a Last-in-First-Out stack for the open list, quickly building deep

[1]The term *heuristic* in the robotics community has multiple meanings that vary widely given context, often causing confusion. In the most general sense, heuristics use incomplete information about a problem to find a solution quicker. Certain heuristic algorithms tend to discover good albeit suboptimal solutions very quickly (which we denote as *suboptimal* heuristic algorithms), while *admissible* heuristics, when combined with a search algorithm such as Branch and Bound or A*, have formal properties that guarantee the discovery of optimal solutions or discover optimal solutions with a minimal number of iterations.

initial solutions that may be of poor quality. Things get more interesting with priority queues. For path independent rewards, one can decompose the cost of a partial solution ("cost to come"), while employing heuristics to estimate the "cost to go." Dijkstra's algorithm sorts the nodes using the cost to come, expanding nodes along the wavefront of minimum cost to come. Best first search uses the cost to go, instead more aggressively expanding nodes that are closer to the goal. A* utilizes the total estimated cost (cost to come + cost to go), which has been shown to minimize the number of expansions to find the optimal solution. A* has been generalized to efficiently coordinate teams of vehicles in M* by first having each agent plan independently, then expanding the coordination space when needed to resolve conflicts [21].

However, in order to find the guaranteed optimal solution, A* will expand all nodes with a heuristic cost less than the optimal cost, which could dramatically increase the run time for a small improvement in cost [19]. To more speedily find a solution, ARA* uses inadmissible heuristics to more aggressively pick nodes closer to the goal, then reuses previous search effort to improve the solution [22]. In a more aggressive manner, Greedy Best First Search (GBFS) [23] will prioritize nodes that are of lowest cost to the goal state for shortest path problems. Alternatively, it is possible to relax some of the assumptions of A*, resulting in an algorithm that is similar to Branch and Bound (B&B) and capable of maximizing path-dependent reward.

2.2.1 Branch and Bound

Branch and bound is a discrete optimization technique that uses admissible heuristics to guide and significantly speed up the search for the optimal solution compared to e.g., brute force enumeration of all candidate solutions. B&B can either be depth first ([10, 24] or priority queue (similar to A*).

Branch and bound was initially introduced for target search when search effort of a particular region was indivisible [25]. Of particular interest is related work that considers a moving target but seeks to maximize the probability of target detection subject to fuel and risk constraints [26]. For information gathering, branch and bound has been used for feature subset selection [27], and has been used in the similar traveling salesman problem (TSP) [19,28]. Depth First Branch and Bound (DF-B&B) has been combined with Gaussian Processes for informative path planning [10] and is arguably most closely related to our research. However, the bounding heuristic used cannot account for multiple sensor measurements in the same location, for which we must explicitly account. DF-B&B has also been used for multi-pass robotic cleaning [24], but scales poorly to larger environments.

2.2.2 Greedy Algorithms & Heuristic Methods

Throughout this dissertation, we will employ various greedy algorithms, which come in many forms. Greedy algorithms make locally or immediately optimal choices that are irrecoverable. In other words, greedy algorithms

do not backtrack or store multiple candidate solutions. Greedy algorithms are often employed to minimize memory or computational burden for a given loss in solution quality. In certain cases, greedy algorithms are known to converge to the optimal solution (e.g. Kruskal's algorithm [29] for calculating minimum spanning trees [30]). For continuous problems greedy algorithms include gradient descent. For a convex function over a convex set, first order (differential) conditions imply global optimality, making local greedy techniques globally optimal. In other scenarios, greedy algorithms can get within a constant factor of the optimal solution (c.f. submodular function maximization, Section 2.4).

Another way to minimize memory or computational burden is to employ heuristic methods that simplify the problem or solve a related but easy to solve problem. Example include employing Depth First Search or computing Minimum Spanning Trees for coverage planning problems (c.f. Section 2.5). Heuristic methods may or may not have formal guarantees on performance bounds (solution quality with respect to global solution). Note that sometimes performance bounds are loose for a particular domain,

2.3 Information Theory

Information Theory investigates the characterization, transmission and reception of information [31]. Three metrics of interest are the signal entropy, KL divergence, mutual information, and cross entropy. For discrete random

variable X, with probability mass function $f_X(x)$, define the signal entropy as:

$$H(X) = -\sum_{x \in \mathcal{X}} f_X(x) \log f_X(x) \tag{2.1}$$

Assuming that the base of the logarithm in (Eq. 2.1) is two implies that $H(X)$ for $X \sim Bernoulli(0.5)$ (a fair coin toss) is equal to one. This defines the natural unit for information with logarithm base 2 called the shannon, otherwise known and referred through this text as the bit. Note that generalizing signal entropy to continuous random variables takes some care, and gets further complicated when handling joint or conditional entropies.

Kullback-Leibler (KL) divergence offers a means of computing a divergence (in bits) between two probability distributions *in the same sample space*. For two discrete random variables X and Y, with joint probability density function $f_{X,Y}(x, y)$, It holds that $\sum_{y \in \mathcal{Y}} f_{X,Y}(x, y) = f_X(x)$ is the marginal PMF of X while $\sum_{x \in \mathcal{X}} f_{X,Y}(x, y) = f_Y(y)$ is the marginal PMF of Y. The KL divergence is defined as follows for discrete probability distributions.

$$D_{KL}(f_X || f_Y) = -\sum_i f_X(i) \log \frac{f_Y(i)}{f_X(i)} \tag{2.2}$$

A similar definition exists for continuous distributions. KL divergence is not symmetric, i.e. $D_{KL}(p||q) \neq D_{KL}(q||p)$.

Mutual information is a natural choice for modeling uncertainty reduction, as it works for continuous, discrete, and hybrid representations and is compatible with Bayesian search techniques. Unlike KL divergence, mutual

information is symmetric, but it is related to the KL divergence:

$$I(X;Y) = D_{KL}(f_{X,Y}(x,y)||f_X(x)f_Y(y)) \tag{2.3}$$

Or, equivalently for discrete RVs:

$$I(X;Y) = \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} f_{X,Y}(x,y)\log\left(\frac{f_{X,Y}(x,y)}{f_X(x)\,f_Y(y)}\right) \tag{2.4}$$

This implies that the mutual information is equivalent to the information lost by approximating the joint distribution with the product of the marginals. Observe that if $X \perp Y$, then $I(X;Y) = 0$. Note that for mutual information, both X and Y need not be in the same space. In fact, one can be continuous while the other discrete. We now define conditional mutual information. Given random variable Z with realization z ($Z = z$).

$$I(X;Y|Z=z) = \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} f_{X,Y|Z=z}(x,y|z)\log\left(\frac{f_{X,Y|Z=z}(x,y|z)}{f_{X|Z=z}(x)\,f_{Y|Z=z}(y)}\right) \tag{2.5}$$

Note that X, Y, Z can be vector quantities in any space (continuous, discrete). Mutual information [31] is an ideal metric to maximize for improving search. Mutual information measures the dependence of two random variables, such as a noisy sensor measurement in the future and the current best estimate of a target's location. Picking a future sensor measurement that maximizes mutual information will effectively reduce the expected uncertainty of the target's location [32]. Most importantly, assessing the mutual

information of an action allows one to formally assess the trade-offs between searching a region multiple times or searching another region not searched, making it an ideal metric for multipass multi-target search [8].

The mutual information of conditionally independent events has been shown to be submodular [8] (c.f. Section 2.4), which implies that multiple correlated sensor measurements offer diminishing returns in how much information is gathered when summing the total information of each measurement individually. When trying to maximize the submodular reward function of mutual information, the reward is *path dependent* requiring one to plan in the space of trajectories, making planning more computationally burdensome.

The cross entropy between two probability distributions f_X and f_Y is the sum of the entropy of f_X and the KL divergence of f_Y from f_X (Eq. 2.6). The interpretation of the cross entropy is that it is the number of bits required to transmit observations sampled from f_Y when the encoding scheme assumes distribution f_X.

$$H(f_X, f_Y) = H(f_X) + D_{KL}(f_X||f_Y) \tag{2.6}$$

Note that the cross entropy is *not* symmetric.

The use of information theory to improve the gathering of information in target search is widespread. Mutual information is a natural choice for modeling uncertainty reduction, and it works for continuous, discrete, and hybrid representations and is compatible with Bayesian search techniques. However,

computing plans of arbitrary length that maximize mutual information is exponentially complex [8]. Example applications that greedily maximize mutual information include the tracking of magnetic anomolies using UAVs [5] or monitoring of simulated environments [6]. Other work only plans several actions into the future, similar to model predictive control, including tasks such as target search [33–35] or target tracking [9].

2.4 Submodular Function Maximization

One field of interest for maximizing information gathering tasks is the area of submodular function maximization. Given a finite set V and a set function $f : 2^V \rightarrow \mathbb{R}$ that assigns real values to subsets $S \subseteq V$, the objective is to find the set S that maximizes f subject to constraints. For all $A, B \subseteq V$, f is submodular if

$$f(A \cap B) + f(A \cup B) \leq f(A) + f(B) \tag{2.7}$$

Or equivalently, for all $A \subseteq B \subseteq V$ and $s \in V$:

$$f(A \cup \{s\}) - f(A) \geq f(B \cup \{s\}) - f(B) \tag{2.8}$$

Submodular functions are therefore said to reflect a diminishing returns property. One submodular function of interest includes the mutual information of two conditionally independent random variables. The objective is to

find a set S^* that maximizes (Eq. 2.9):

$$S^* = \arg\max_{S \subseteq V} s.t.\ f_C(S) \leq K \tag{2.9}$$

given general constraint $f_C(S) \leq K$. For example, the constraint $f_C(S) = |S| \leq K$ imposes a limit on the number of sensor placements and is the most common. Due to the interactions of selecting different variables on total reward, finding a global maximum by finding a solution subset is a combinatorial problem, making brute force enumeration techniques intractable. Branch and bound solutions exist to improve the run time of exhaustive search [36]. However, one solution of interest is the greedy algorithm [37]. The kth iteration of the greedy algorithm adds the next element $s_{k+1} \subseteq V$ to $S_{k+1} = S_k \cup \{s_{k+1}\}$ selecting s_{k+1} such that $f(S_{k+1})$ is maximized over all remaining elements in V until $k = K$. Nemhauser et. al. proved that the greedy algorithm returns solution $\tilde{S}$ where $f(\tilde{S}) \leq (1 - \frac{1}{e}) f(S^*)$ where $e \approx 2.71828$ is Euler's number [37]. This results in a solution bound that is within 0.63 of optimal.

This seminal work has inspired many extensions making it relevant to multi-agent information gathering path planning problems. This work and the following extensions have inspired many of the listed contributions in this manuscript. Factoring (closed, feasible) path constraints is a nontrivial constraint as opposed to letting $f_C(s) = |S|$ and is similar to the submodular orienteering problem. Chekuri and Pal developed the recursive greedy algo-

rithm heuristic solver which solves the submodular orienteering problem [38]. This algorithm, however, assumes that each state is only visited once, making it unsuitable for our application. Alternate techniques outperform Chekuri and Pal's approach by using randomized algorithms [8]. When coordinating teams of vehicles for information gathering tasks, Sequential Greedy Allocation (SGA) is closely related to greedy submodular function maximization. SGA plans agent paths one at a time in a round robin fashion [39, 40].

Functions such as mutual information (consider Gaussian Processes) have a locality property, where the correlations between sensor readings decrease as the distance between the sensor locations increases. This property has been exploited enabling more efficient search of optimal paths [41]. Finally, use of sequential [greedy] allocation enables multiple agents to maximize submodular objectives with performance bounds [42]. In [43], Fisher information matrices, combined with rapidly-exploring random trees (RRTs), was used for information-rich path planning. There has also been some work where agents conduct information gathering tasks (submodular reward) in the presence of periodic connectivity [44].

2.5 Coverage Planning

In the scenario with uninformative prior information, there is a close resemblance between trajectories that maximize mutual information and coverage plans. A recent survey outlines the various techniques used in coverage plan-

ning [45]. Two approximate solutions are of interest which focus on the 2D case. Zelinsky's algorithm uses dynamic programming to generate a value function, then traverses level sets to cover the entire region [46]. The proposed algorithm can get stuck in the presence of obstacles and must back out to continue its coverage plan. Boustrophedon coverage outlines an approach to partition polygonal environments and suggests using a modified version of depth first search (DFS) to dictate the search order of the various regions [47]. Alternatively from the ox-plowing patterns, spirals can also be used for generating coverage plans [48]. There has also been work on generating coverage plans in partially known environments [49].

For optimal coverage on graphs, coverage planning is equivalent to the traveling salesman problem (TSP), which is NP-hard. Many approximate solvers exist [50]. The multi-agent case is equivalent to the multi-traveling salesman problem (M-TSP). One such approximate solver for M-TSP utilizes genetic algorithms [51]. Interestingly, the solution to the Chinese Postman Problem [52] where the goal is to find the optimal tour covering all edges of a graph can be solved in polynomial time. When factoring in environment physics such as swift ocean currents, strategies that directly account for sensors swept across the environment cover more area than approaches that assign agents to do station keeping at a given waypoint, with all agent waypoints forming a grid [53]. In addition, use of coverage planning has been used for UUVs for target search missions [54].

2.6 Locational Optimization, Voronoi Partitions, & Lloyd's Algorithm

Locational Optimization is a branch of Operations Research that is concerned with finding optimal locations of facilities in a continuous region to minimize transport/service costs [55]. Before proceeding to the relevant objective function, we first define the Voronoi Partition. Given a set of points $P = \{p_i\}_{i=1}^{N}$ and a closed and bounded, simply connected set $X \subset \mathbb{R}^2$, the Voronoi Partition V_i of point p_i is the set of all points in X for which p_i is the closest waypoint in P:

$$V_i = \{q \in X : ||q - p_i|| \leq ||q - p_j||, \forall j \neq i\} \tag{2.10}$$

where $||\cdot||$ is the $l2$ norm. From this definition it is clear that computing the Voronoi Partitions identifies for each point in X the closest point in P. One locational optimization function of interest for information gathering tasks is the sensor placement problem, given the nonnegative sensor function ϕ, minimize the objective function (Eq. 2.11) [56]:

$$\mathcal{H}(P) = \sum_{i=1}^{n} \int_{V_i} l(||q - p_i||)\phi(q)dq \tag{2.11}$$

where $l(\cdot)$ is a penalty function characterizing performance degradation over distance. An example mission scenario would define ϕ as a distribution of target locations letting l denote the cost in detecting targets that are a given

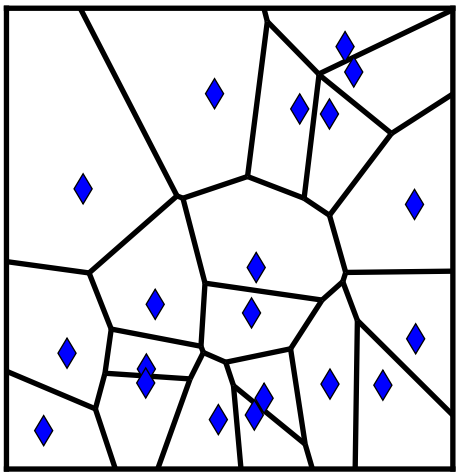


Figure 2.1: An example Voronoi Diagram of 20 points in a bounded volume. Blue diamonds denote the points, while solid black lines are the edges of the regions.

mass	$M_{V_i} = \int_{V_i} \phi(q)dq$
centroid	$C_{V_i} = \int_{V_i} q\phi(q)dq/M_{V_i}$
error	$e_i = C_{V_i} - p_i$

Table 2.1: Properties of the Voronoi Partition.

range from the sensor. Typically, $l(||q-p_i||) = ||q-p_i||^2$ which corresponds to $\frac{1}{r^2}$ falloff in signal intensity with range, reducing the probability of detection.

We now elaborate on discussing various aspects of Voronoi Partitions, prior uses for related research, and various applications of Voronoi Partitions in Locational Optimization problems. In terms of motion planning for autonomous vehicles on or under the water, Voronoi Partitions (VPs) are often used to space agents sufficiently far away from obstacles, generating safe, dynamically feasible trajectories for autonomous surface vessels [57]. Researchers have also applied Voronoi Partitions (VPs) to information gath-

ering tasks to teams of vehicles on the water. It has been noted that planning in strong current flows will strongly impact the reachability graph of the agent when generating coverage plans [58]. To counter this, researchers have extended the notion of VPs by using time varying flow fields to generate non-euclidean cost maps for the distance metric in (Eq. 2.10) to steer ASVs [59]. There has also been work on using VPs to help 2 autonomous underwater vehicle gliders coordinate by dynamically partitioning the environment [60, 61]. VPs have also been used for simultaneous coverage and tracking of targets [62], along with a wide variety of additional locational optimization problems [63].

Optimizing the location of N points solving (Eq. 2.11) is NP-hard and difficult to solve globally. Instead, locally greedy methods have been developed that minimize the objective function given an initial configuration of points. Lloyd's algorithm continually moves the points to the centroid of the respective Voronoi partition until the algorithm converges [64]. Lloyd's algorithm has been employed for mission planning for teams of UAVs to distribute resources [65], and has been augmented into a continuous time formulation used to steer teams of vehicles to regions of interest to maximize reward [56]. In both cases, each vehicle constitutes a single point in the Voronoi Diagram. In terms of persistent monitoring, previous work has adapted such techniques to generating multiple waypoint tours for a single vehicle by incorporating tour arc length costs into the objective function [66].

2.7 Markov Decision Processes

With regard to planning system trajectories accounting for motion uncertainty, Markov Decision Process (MDP) based formulations are standard in the robotics community [67]. Paths generated using this approach have been executed on autonomous unmanned surface vehicles [68, 69]. MDP-based techniques have been used to steer UUVs toward waypoints selected by higher level information-driven planners [70]. Decentralized, multiagent formulations of MDPs exist where agents only cooperate when it affects the reward [71]. We now define the standard MDP formulation.

The state of the agent is a random variable (RV) denoted by S (with realizations denoted by the lower case s), resides in finite state space $\mathcal{S}$. Further, define action A (with realization a) residing in the finite action space $A \in \mathcal{A}(s)$. The MDP defines a stochastic motion model that is Markovian:

$$\begin{aligned} &\Pr(S_{t+1} = s' | S_t = s_t, A_t = a_t) = \\ &\Pr(S_{t+1} = s' | S_t = s_t, \ldots, S_0 = s_0, A_t = a_t, \ldots, A_0 = a_0) \end{aligned} \tag{2.12}$$

where $t \in \{0, 1, 2, \ldots, T\}$ denotes the time index throughout the evolution of the system's dynamics. The probability mass function $\mathcal{P}^a_{ss'} = \Pr(S_{t+1} = s' | S_t = s, A = a)$ defines the transition probabilities for the agent given the present state and action selected. All aspects of the motion model can be time varying by absorbing the time into the state (therefore plan-

ning in space-time). Define the space of execution histories $\mathcal{H} = \{h = s_0 a_0 s_1 a_1 \ldots s_t, h \text{ is feasible.}\}$ to consist of all feasible execution histories (i.e. system trajectory) h.

The associated objective of the MDP is to maximize the (path-independent) reward:

$$J = \mathrm{E}_r[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)] \tag{2.13}$$

with discount factor $\gamma \in (0, 1)$ and path independent reward function $r(s, a)$. Alternatively, one can define the reward of the system path h. $R(h) = \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)$, then (Eq. 2.13 becomes $J = \mathrm{E}_h[R(h)]$ over all feasible execution histories. By the principle of optimality [1], there exists a value function V^* that satisfies the Bellman Equation (Eq. 2.14):

$$\begin{aligned} V^*(s_t) &= \max_{a \in \mathcal{A}(s_t)} \left(\mathrm{E}[r(s_t, a) + \gamma V^*(s_{t+1})]\right) \\ \pi^*(s_t) &= \arg\max_{a \in \mathcal{A}(s_t)} \left(\mathrm{E}[r(s_t, a) + \gamma V^*(s_{t+1})]\right) \end{aligned} \tag{2.14}$$

The most straightforward approach to solve (Eq. 2.13) is value iteration (VI) [67]. In addition to building V^*, VI also builds optimal policy π^* which defines the optimal state to action mapping to maximize reward. VI has been shown to be linear in the number of states [72], which grows exponentially with the number of dimensions. VI is highly parallelizable since all computations use local information (the approach effectively casts the problem into a space in which the greedy solution is optimal) but requires care in sharing global information passed by the Bellman backups, since many com-

putations in a naive implementation are redundant. Early results to speed up the convergence include asynchronous VI [73], which allows states to be visited in any order as long as each state is visited infinitely often [74]. Asynchronous VI allows any previously computed information to be immediately available for the next backup with the backup done "in place", improving convergence and reducing memory limitations [72, 73]. This is sometimes referred to as Gauss-Seidel backups, especially when all states are iterated over in a round-robin fashion. Prioritized value iteration determines the order in which cells are updated minimizing redundant backups facilitating efficient wave-front expansions [75]. Other work has investigated partitioned value iteration where the state space is broken up into partitions, and the algorithm focuses on optimizing one partition at a time [76]. If the state space is allocated in large chunks to different processors, it is likely that most processors will remain idle in regions that don't change much. One application illustrates the importance of combining the mentioned approaches by using partitioned, prioritized, parallel version of value iteration [77]. An alternate approach to solving MDPs, in particular for related reinforcement learning problems, is policy iteration [72].

When extending the MDP formulation to path dependent rewards, VI becomes intractable, which is why the vast majority of techniques to solve MDPs do not solve the various problems investigated in this dissertation. The various techniques considered will treat the value function in fundamentally different ways. These are:

- Estimate V^* using Monte Carlo Rollouts (c.f. Monte Carlo Tree Search)
- ignore the existence of V^* (c.f. Cross Entropy Method)

2.8 Monte Carlo Tree Search

MCTS has garnered widespread attention in the AI community due to its recent success in playing adversarial games such as Go [78]. MCTS combines multi-armed bandit problems [72] with tree-based search techniques to efficiently guide Monte Carlo rollouts to maximize reward. MCTS has been used before for sequential Bayesian optimization tasks for generating information gathering trajectories for a single UUV [79].

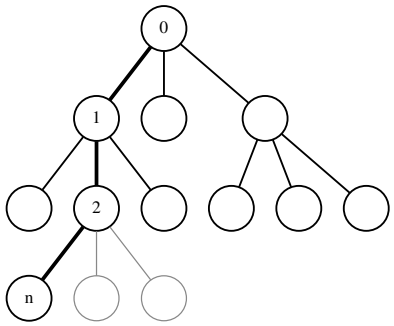


Figure 2.2: A graphical depiction of how MCTS conducts its search. For illustration purposes, assume all the nodes in this figures are for a given action, each layer corresponding to a given time index a given action is selected. Initially starting at node 0, the select function is called recursively until it selects the node 2. The second stage expands (adds) a new node n (greyed out siblings are still waiting to be expanded) and appends it to the tree. From node n, a Monte Carlo simulation is played out using the default policy. The outcome of the simulation is then back-propagated to all parent nodes (thick black edges), including the root node.

Figure 2.2 offers a simplified description of how MCTS conducts its search by building a tree of explored actions. In deterministic settings MCTS explores action sequences, though in probabilistic or adversarial scenarios MCTS is exploring policies based on the history of observed outcomes.

MCTS contains four major stages that it iterates over until the time to compute expires or a given solution quality is met. They are: selection, expansion, simulation, and back-propagation. First, MCTS must (recursively) select the next node among the children of the currently selected node. The most commonly used selection procedure is UCB1 [80]. This selection process generates trees of varying depth guiding the search, and visits nodes infinitely often in the limit to ensure rewards are accurately evaluated.

We now describe the implementation of MCTS (Alg. 1) used in this dissertation which builds a tree data structure, $\mathcal{T}$, consisting of root node *root* and all other nodes expanded during the search. In effect, $\mathcal{T}$ builds a Monte Carlo estimate of value function V^*. For each state in $\mathcal{T}$, the optimal action can be greedily selected that maximizes the value function. *null* is an empty/invalid node, where $root.parent = null$ There are alternative implementations than the one presented here such as UCT using a tabluar implementation that the reader may feel more comfortable with [80]. The tabular implementation will take more memory but can store the outcomes of deeper rollouts earlier in the planning process. To simplify the presentation of MCTS, we will assume that everything is deterministic. In this presentation, the simulator state $S = S_t$ also encodes the time step of the state. At each state, the motion

model can select among the available actions, transitioning to the next state S_{t+1}.

Here, a given node N_S consists of the simulator state S, $N_S.S$, a parent node $N_S.parent$, and a reward estimator, $N_S.reward$ that estimates the reward and variance by using recursive updates. $reward = (N, \mu, \sigma^2)$ tracks the sample mean μ_k and sample variance σ_k^2 for N samples. $reward$ is initialized to $(0, 0, 0)$. The update formulas are in Alg. 2.

After a leaf node has been selected ($SELECT$), new previously unexplored children can be created by employing the motion model. The next step is to add all of the children to the tree ($EXPAND$). Then, a Monte Carlo simulation starts at this node, using the default policy to solve the rest of the problem and continues until termination ($SIMULATE$). Deeper nodes in the tree indicate a more fully defined partial solution (portion of a trajectory) to the planning problem. The default policy may be random, greedy or heuristic but it must be fast to compute and generate results that are effective at guiding the search. Once the reward for the rollout is determined, each node along the path from the expanded node to the root node updates its estimated expected reward by incorporating the most recent rollout ($BACKPROPAGATE$). The selection process for the next rollout starts again and continues until the time limit for computing expires or some criteria for finding a satisfactory solution is met (Alg. 1).

$EXPAND(N_S)$ adds all children of N_S to the tree, using the motion model, and updates N_S to one of it's children (Alg. 4). $SIMULATE(N_S)$

Algorithm 1 $MCTS$

Require: motion model, default policy π, reward, initial state $root$

while termination condition not met **do**
  $N_S \leftarrow root$
  $N_S \leftarrow SELECT(N_S)$
  $N_S \leftarrow EXPAND(N_S)$
  $R \leftarrow SIMULATE(N_S)$
  $BACKPROPAGATE(N_S, R)$
end while
return $\mathcal{T}$

Algorithm 2 $update(reward, x)$

Require: reward data structure $reward$, value x

$N \leftarrow N + 1$
$a \leftarrow (N-1)/(N)$
$\mu \leftarrow a \cdot \mu + x/N$
if $N > 0$ **then**
  $\sigma^2 \leftarrow a \cdot \sigma^2 + (x-\mu)^2/(N-1)$
else
  $\sigma^2 \leftarrow 0$
end if
return $reward$

initializes the simulator to $N_S.state$ and completes the playout until termination, returning total reward R (Alg. 5). $SELECT$ is described in Alg. 3 and $BACKPROPAGATE$ is described in Alg. 6. Here, we denote the motion model with $model$, which has functions that either return the set of possible outcomes (after applying any feasible action) from a given state $model.children(S)$. Further, given state S_t and action a, the resulting outcome is $S_{t+1} = model.next(S, a)$. Note that this is very general and valid model include MDPs and other stochastic motion models with suitable mod-

Algorithm 3 $SELECT(N_S)$

Require: node N_S, selection function sel
1: **if** N_S is not a leaf node **then**
2: $N_S \leftarrow UCB1(N_S)$
3: $SELECT(N_S)$
4: **end if**
5: **return** N_S

Algorithm 4 $EXPAND(N_S)$

Require: node N_S, motion model $model$
1: $child \leftarrow c \in model.children(N_S.S)$ {select one of the children}
2: $N_c \leftarrow null$ {create new child node}
3: $N_c.S \leftarrow child$
4: $N_c.parent \leftarrow N_S$
5: $N_S \leftarrow N_c$
6: **return** N_S

ification.

$SELECT$ employs UCB1 (Eq. 4.8) for the selection procedure:

$$UCB1(N_S) = \underset{a \in \mathcal{A}(S)}{\arg\max} \left(\bar{R}_a + C\sqrt{\frac{\log(\sum_a n_a)}{n_a}} \right) \tag{2.15}$$

$\bar{R}_a$ is the average of all rewards gathered by selecting action a at node N_S computed by $reward$. n_a is the number of times action a was played at node N_S, $\sum_a n_a$ is the number of total plays at N_S. C is a parameter that is experimentally tuned to favor exploration (C larger) vs. exploitation (C smaller).

Because MCTS is simulation based, it is straightforward to extend MCTS to simulate random outcomes or random decisions in a random default pol-

Algorithm 5 $SIMULATE(N_S)$

Require: node N_S, motion model *model*, default policy π_0, reward function $R(\cdot)$

$\mathcal{T} \leftarrow \emptyset$
$\mathbf{U} \leftarrow \emptyset$
while $N_S \neq null$ **do** {recursively follow $N_S.parent$ returning to the *root* node}
  $\mathcal{T} \leftarrow \mathcal{T} \cup \{N_S.S\}$ {build trajectory}
  $\mathbf{U} \leftarrow \mathbf{U} \cup \{N_S.action\}$ {build action sequence}
end while
$S \leftarrow N_S.S$
while $S \neq S_T$ **do** {complete a playout until termination}
  $a \leftarrow \pi_0(S)$ {use default policy to select action}
  $S \leftarrow model.next(S, a)$
  $\mathcal{T} \leftarrow \mathcal{T} \cup \{S\}$ {build trajectory}
  $\mathbf{U} \leftarrow \mathbf{U} \cup \{action\}$ {build action sequence}
end while
return $R(\mathcal{T})$

Algorithm 6 $BACKPROPAGATE(N_S, R)$

Require: node N_S, reward R

while $N_S \neq null$ **do**
  $N_S.reward \leftarrow update(N_S.reward, R)$
  $N_S \leftarrow N_S.parent$
end while
return N_S

icy once the reader fully understands MCTS. Whenever a random outcome is needed (e.g. during $SELECT$ or $SIMULATE$), sample the stochastic motion model. In addition, the children of a node are all possible (action,outcome) pairs. When evaluating action a at node N_S for $SELECT$ or other operations, one must average/sum over all outcomes for a given action (Use the sample mean, not the expected value.).

UCB1 requires experimental tuning of the parameter C.It is also important to note that MCTS is difficult to implement in parallel, but an ensemble of MCTS searches can be merged [81]. MCTS has been applied to MDPs as a form of real time dynamic programming [80,82]. Such formulations form the basis of our application of MCTS to the stochastic path planning problem. We used UCB1 (Eq. 4.8) for the selection procedure of MCTS [80].

2.9 Cross Entropy Method

The Cross Entropy Method (CEM) [83,84] is a general importance sampling strategy that iteratively improves solution quality by tuning the parameters of a probability distribution. CEM has been applied to policy iteration for MDPs [85–88] for path-independent rewards and is particularly suited for the task at hand. Model predictive path integral control (very similar to CEM) has been used to solve model predictive control problems [89]. Starting with a uniformly random policy, CEM simulates many (e.g. $N_p = 5,000$) rollouts of the agent's policy per iteration. Each rollout spans the entire mission duration, and the rollout's reward is evaluated. The rollouts with reward in the $(1-\rho)$ percentile for $\rho \in (0,1)$ are assessed. The policy's parameters are then updated to increase the likelihood of the $(1-\rho)$ percentile trajectories by computing the maximum likelihood estimate (MLE) of all state to action distributions.

We now provide a generic formulation of CEM and will describe applying

CEM to the problem at hand. Since we are concerned with discrete problems, we assume that all distributions are categorical and hence have a finite set of parameters defining them. CEM also works for continuous distributions, especially those in the exponential family.

The objective of CEM is to maximize reward function $Q(x)$ where $x \in \mathcal{X}$ is finite but large

$$x^* = \arg\max_{x \in \mathcal{X}} Q(x) \tag{2.16}$$

Using CEM for optimization assumes that the optimal solution x^* is a rare event when searching over random samples $X \in \mathcal{X}$. As is common with importance sampling, we define $l = \Pr(Q(X) \geq \gamma) = \mathrm{E}[I_{\{Q(X) \geq \gamma\}}]$ with $I_{\{\mathcal{A}\}}$ the indicator function of set $\mathcal{A}$. We define distribution parameters θ. The goal is to update the distribution parameters to bias the search toward the region that contains the optimal solution.

For (deterministic) combinatorial optimization problems, a typical termination criteria is that γ_t does not change in $n = 5$ iterations. Minimizing the cross entropy (Alg. 7 Line 12) is equivalent to generating maximum likelihood estimates of the distribution. However, the MLE update of the policy parameters does not factor in prior knowledge, which is detrimental by forcing probabilities of actions that are not sampled in the kth iteration to zero. The MLE distribution parameters are filtered with a low-pass filter with parameter $\alpha \in (0, 1]$ with $\alpha = 1$ equivalent to not using the low-pass filter in Line 13 in Alg. 7. When the combinatorial problem is deterministic, the best found

Algorithm 7 Cross Entropy Method for Combinatorial Optimization

Require: parametric probability mass function $f(x, \theta)$, initial parameters θ_0, reward function $Q(x)$, number of batched trials N_p, percentile ρ, low-pass filter parameter α.

1: $k \leftarrow 1$
2: **while** termination condition not met **do**
3: $k \leftarrow k + 1$
4: $x_{max} \leftarrow x_{null}$ {x_{null} denotes infeasible solution}
5: $Q_{max} \leftarrow -\infty$
6: $\mathbf{X} \leftarrow$ sample $f(x, \theta_t)$ N_p times. {conduct Monte Carlo rollouts}
7: **if** $\exists x \in \mathbf{X}$ s.t. $Q(x) > Q_{max}$ **then**
8: $x_{max} \leftarrow x$
9: $Q_{max} \leftarrow Q(x)$
10: **end if**
11: $\gamma_k \leftarrow (1 - \rho)$ percentile of $Q(X), X \in \mathbf{X}$. {update rare event}
12: $\phi_k = \arg\max_v \mathrm{E}[I_{\{R(X) \geq \gamma_k\}}] \log f(X, v)$ {minimize Cross Entropy}
13: $\theta_k \leftarrow \alpha\phi_k + (1 - \alpha)\theta_{k-1}$ {filter parameter update}
14: **end while**
15: **if** problem is deterministic **then**
16: **return** (x_{max}, Q_{max})
17: **else**
18: **return** θ_k
19: **end if**

solution in Line 5 of Alg. 7 is a feasible, reproducible solution. However, if the problem is stochastic, x_{max} or Q_{max} may represent an unlikely outcome and is not reproducible with probability 1.

Inclusion of the low-pass filter has been shown to converge to the optimal solution [90]. We further observe that the use of a low-pass filter has a close relationship to the Maximum *a Posteriori* (MAP) estimate of the filter. For the Dirichlet conjugate prior distribution, the Bayes Filter can be formulated as a time varying low-pass filter, where $\alpha \to 0$ due to the accumulation of

information in the prior. However, information from old iterations ought to be eventually forgotten anyway as it represents the rewards of old policies that gathered less reward than the best-known policy. Fixing α to be constant offers a compromise between both MLE and MAP. This is done iteratively until either the value of the $(1-\rho)$ percentile converges, or if a maximum number of iterations has been completed.

CEM has been applied to generating policies for a single beacon vehicle supporting up to two survey AUVs in the presence of a simplified tidal current model (for a team of three vehicles) by minimizing the sum squared position error, which is path independent [86, 87]. CEM is competitive with Dynamic Programming (DP) and both approaches are robust to environmental uncertainties [87]. We further note that the described CEM formulation can in principle handle path dependent rewards and has a means of leveraging ocean forecasts, yet the performance of CEM in these scenarios has not been demonstrated in previous work. The DP based technique will not handle path dependent rewards efficiently, and DP will not scale well to large teams of agents.

For completeness, it is important to note that general formulations of CEM and proofs of convergence for CEM do not extend to solving stochastic problems such as the problem at hand. CEM is merely a stochastic approximation technique for deterministic problems. It is possible to generate stochastic motion models where the $(1-\rho)$ percentile of rollouts is a biased sample of trajectories and does not represent expected system performance.

Techniques exist to generalize CEM to stochastic problems [83, 91].

2.10 Path Independent Approximations to Path Dependent Reward

Given the demonstrated difficulty of solving problems with path dependent rewards, it would be of interest to find alternate means of solving ancillary path-independent objectives that approximate the path-dependent reward well. One approach was to use Markov approximations for transect sampling tasks [92]. This contrasts with the approximation of history dependent policies with Markov policies in Chapter 4, instead changing the reward function being optimized itself. However, the transect sampling task problem formulation enables that the path dependent components decorrelate along the path, so it's applicability to more difficult problems (such as paths that self intersect) is unknown. In addition, other work for UAVs has investigated additive approximation to submodular rewards [93]. For calculating optimal stopping times for path dependent systems, path-wise approximations have been treated as "Taylor expansion" [94].

2.11 Gaussian Processes

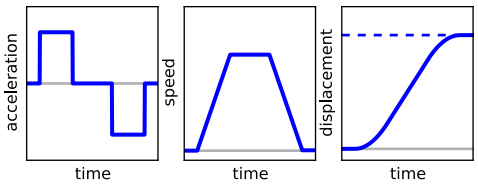
Gaussian Process (GP) Regression based objective functions are in widespread use due to the ability to account for such correlations [70, 95]. GPs also have analytic solutions to metrics such as mutual information or entropy. One

major limitation to GPs is their inability to scale well to large problems. For n sensor measurements, naive GP regression is of $O(n^3)$ complexity. This has spawned many techniques to improve GP regression. The accuracy of sparse approximate Gaussian processes [96] are limited by the number of inducing variables and cannot express complicated sensor deployment scenarios efficiently. Other techniques involve covariance tapering [97] or nearest neighbor Gaussian processes [98] which induce sparse representations while modeling the underlying data accurately.

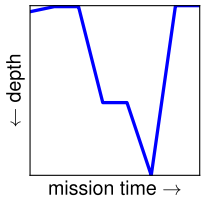
One of the major criticisms of GP regression is that it enforces a uniform spacing of samples [95], suggesting uniform covering strategies as a cheaper alternative to GP regression.

2.12 Feasible Trajectory Generation

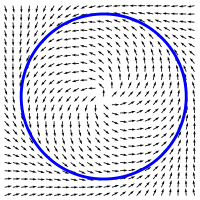
To ensure agile maneuvering, autonomous systems must generate dynamically feasible trajectories that satisfy the vehicle constraints and that the vehicle's controller can reliably track. Example challenges include accounting for limited acceleration (or braking) capabilities to safely avoid obstacles, minimum turning radius, or staying within the flight envelope of the airframe. Path planning algorithms that do not factor these constraints will produce energy consuming, jerky or risky paths [99]. There are a wide variety of approaches taken in the robotics community, many of which are depicted in Figure 2.3. Traditionally, dynamically feasible trajectories are time varying



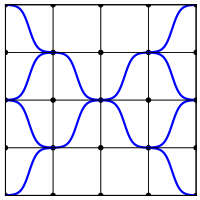
(a) canonical technique: feasible trajectory from BVP solver



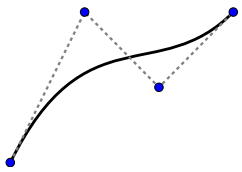
(b) control signal representation



(c) feedback policy



(d) macro actions/lattice



(e) smooth curves (bézier curve shown)

Figure 2.3: Different representations of a dynamically feasible trajectory are possible.

signals of the system evolution (including higher order derivatives) which satisfy the equations of motion of the system, along with the control signals that reproduce the trajectory. For point to point maneuvers, this often requires solving a boundary value problem, and can be solved using a wide variety of control techniques [1, 100, 101] (Fig. 2.3(a)). However, BVP solvers often have trouble accounting for obstacles, so they are combined with various path planning techniques to plan around obstacles. The well known extension of Rapidly Exploring Random Trees (RRT), RRT*, is a sample based motion planner initially intended for combining BVP solutions to find asymptotically optimal trajectories in the presence of obstacles [102]. Further, Kinodynamic RRT* has been developed to plan asymptotically optimal trajectories for sys-

tems with linear dynamics in the presence of obstacles [103]. Sample based techniques tend to suffer due to the expense of solving the BVP many times. In addition, kinematic paths can be converted to dynamic trajectories using back-stepping [104], but this technique cannot generate trajectories that exploit the dynamics to improve performance.

Differentially flat systems are systems that can be fully controlled by defining the trajectories for their flat outputs such as position and yaw [105]. For example, quadrotors require that the flat output trajectories (position, yaw) be smooth up to the 4th order in order to have continuous motor commands for graceful motion [106]. Splines are an effective means of representing feasible trajectories for differentially flat systems [106, 107]. Optimizing splines to satisfy point-wise constraints minimizing snap [106], or minimizing snap and segment time [108] offer practical means of generating feasible trajectories. Similarly, Bézier curves (Fig. 2.3(e)) combined with speed profiles have been used to coordinate teams of UAVs [109].

The use of splines to represent trajectories of quadrotors suggests that alternate representations of the trajectory generation problem may facilitate the search. The planner could instead sample in the control space (Fig. 2.3(b)). Applying the selected control signal to the system with the correct starting condition will generate a dynamically feasible path. Alternatively, one could also search the space of policies, or a state to action mapping. One additional approach is to generate a family of various maneuvers or macro actions called Maneuver Automata (MA) provide an alternative language

for describing dynamically feasible trajectories for vehicles [110], which have been applied to RRT-like algorithms [111]. Other maneuver techniques include using dynamic movement primitives to interpolate optimal control solutions [112]. Stitching maneuvers together to form a lattice (repeating structure, Fig. 2.3(d)) is amenable to searching for algorithm using A* [113, 114]. Another way to generate dynamically feasible trajectories is by developing policies (Fig. 2.3(c)). The MDP based techniques mentioned in Section 2.7 are satisfactory means of generating policies. Also of interest is the use of sequential composition, generating collections of Linear Quadratic Regulator feedback controllers to steer teams of agents [115]

2.13 Literature Related to Topic 1: Search and Rescue

Research interest in probabilistic target search appeared as early as 1956 [116] studying the probability of detection of a target given various sensor conditions. Since then, a large body of work has been developed on the task of target search, including many surveys of the subject [117–119].

First and foremost, our literature survey will focus on problems requiring active search where agents keep moving to locate targets [35, 120]. Several important factors that delineate different approaches in the literature include the number of targets to be tracked, the assumptions about the target motion model, and who is searching for the targets. Some approaches are only for finding a single target, while others are for finding multiple targets. When

working under a Bayesian framework, it is common to track a probability density function of a single target's location [34]. This can be extended to a predefined number of targets in parallel [33, 121], or using random finite sets to track the likelihood of the existence of a random number of targets [5, 120, 122].

Further, whether or not targets are stationary or dynamic affects the search. If targets are known to be stationary (or static), one could divide the search space into cells on a grid and track hypotheses in each cell [5, 34, 123]. Or, one can track the number of targets contained within each cell [124, 125]. Since our work assumes stationary targets, there is a grid of cells, but the approach collects multiple cells into regions for faster evaluation of coverage plans by planning at the coarser granularity of searching regions instead of cells, requiring fewer actions for a full duration coverage plan at the cost of generating suboptimal plans. Targets that react and move according to the agent's position can be assumed to have an adversarial relationship characterized by pursuit-evasion games [118]. Alternatively, target motion can be modeled as a Markov process which is unaffected by the agent's actions [25, 35, 126]. There have been experiments using UUVs in a littoral environment tracking moving targets [127] using a multi-hypothesis tracker [128]. Certain approaches also factor coordination between agents to find the target(s), either implicitly or explicitly [126]. Regarding multiagent algorithms, in particular explicit coordination methods are exponentially complex with the number of agents [126]. Multiagent coordination where the agents share

and fuse observed data is common [33, 35, 121, 129–131]. Multiagent search can also be adversarial where two competing teams are searching for the same target [132].

In contrast to finding trajectories that maximize mutual information, one could plan trajectories that minimize the time to detect the target [48, 133, 134]. Minimizing expected time to detection vs. guaranteeing capture or reducing target location uncertainty will be appropriate in different scenarios [48]. However, the mentioned works are formulated for either a single target [133, 134] or for a fixed number of targets [48], which is unsuitable for widespread disasters on land where the number of survivors is unknown *a priori*.

Hierarchical approaches can be used to improve the representation of the target distribution or of the environment to speed up the search. Quad-tree like structures representing the target distribution have been used [5,135,136]. Alternate approaches take advantage of the structure of the environment (e.g., rooms in an indoor map) [126, 137] but assume that regions are convex and a line of sight sensor is used.

2.14 Literature Related to Topic 2: Geospatial Tasks and Underwater Vehicles

There is a wide body of research regarding using underwater vehicles for geospatial tasks [138]. Using teams of underwater vehicles for information

gathering tasks has been done most commonly with teams of gliders [139], but also with profiling floats [15] and hybrid glider/AUVs [140]. Other work has used Lagrangian drifters providing position information to teams of AUVs tracking oceanographic features [141]. Some earlier work starting in 2007 generated ellipsoidal steering laws to maximize informativeness of the sampled ocean data [142]. Later, there's been field experiments for tracking algae blooms, where researchers investigated Intermediate nepheloid layers (INLs) in the ocean [143]. A time-varying information objective was explored in [11], when using sampling based motion planning for environmental information gathering. Adaptive sampling tasks are scenarios where a nonuniform distribution of sensing measurements improves the cost effectiveness of the mission when, e.g. reducing measurement uncertainty for a given mission. Mixed Integer Linear Programming has been used for formulating adaptive sampling tasks for underwater vehicles [144].

Planning UUV trajectories that exploit the ocean currents in the presence of forecast uncertainty remains an open research challenge. While the effective forward speed of gliders ranges from 25 cm/s (Spray) to 35 cm/s (Slocum) [142], the intended target platform of the research is that of vehicles of severely limited propulsion capabilities (10 cm/s) to maximize endurance. In terms of exploiting ocean currents for energy efficient path planning purposes, graph based techniques have been used to generate time and energy optimal paths for Autonomous Surface Vessels [145]. Ensemble methods in uncertain flow fields based on the Pontryagin maximum principle have been

used [146]. These approaches were developed for waypoint navigation and not information gathering tasks, however. AUV gliders have been able to use the ROMS ocean model for tracking of algae blooms in the Southern California Bight [147]. However, the ROMS ocean model does not quantify model uncertainty, so researchers have developed *post hoc* techniques to estimate model uncertainty by modeling interpolation variance [95]. Use of ensemble methods for ocean forecasting [148] has been used in live experiments in the Gulf of Mexico and significantly improved the prediction accuracy of near surface drifters, outperforming ocean models similar to ROMS such as NCOM or HYCOM [146]. We note that ensemble techniques consist of populations of estimates which can drive statistics on model characteristics. Planners using ensemble methods have been implemented by solving the planning problem as a boundary value problem in the presence of uncertainty [149], but have only been used to solve simple canonical problems such as double-gyre flows so their applicability to models of the ocean is unknown. An alternate planning technique for accounting for environmental distrubances includes using a variant of A* that conducts minimax game-tree search, and has been used successfully with ASVs [150]. However, such game-like nondeterministic formulations can be overly conservative compared to probabilistic approaches. Prior use of MDPs for UUV applications was discussed in Section 2.7.

2.15 Literature Related to Topic 3: Persistent Monitoring for ASVs

Persistent sensing approaches [151] model the information uncertainty in the environment as a field defined over a set of locations and assume that the field increases linearly at locations beyond the sensing range of the robot and decreases linearly at locations within the robot's range. Persistent sensing is distinct from static sensor placement or the art gallery problem [152], where static assets are strategically placed to maximize coverage. The price of static sensors can prove costly in an ocean environment, therefore planning for moving sensors results in lower sensing cost. Persistent monitoring strategies have also been employed for establishing general trends in current flow [153]. Similar work enabling surveillance tasks in a congested littoral enviornment has been done [154].

Control-based approaches involving locational optimization include [56, 66,155]. There also exist information-theoretic approaches that specify which areas are of interest and attempt to maximize the informativeness of a plan. In [156], agents followed the gradient of mutual information to minimize total entropy.

Conceptually, one could pick areas of interest in the environment and find the optimal path connecting fixed nodes, which is equivalent to solving the multiple Traveling Salesman Problem (M-TSP). M-TSP is even more difficult than TSP, though many approximate solutions exist [51]. However,

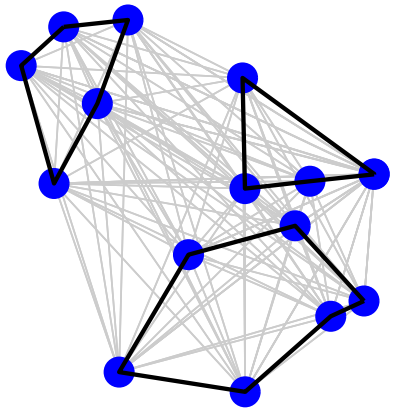


Figure 2.4: An example multiple Traveling Salesman Problem. Nodes to visit are in blue, inter-city connections in grey, with the selected multi-tour in black.

one major limitation to M-TSP is that the areas of interest are fixed and do not adapt to move from high cost areas to low cost areas. Techniques based on locational optimization techniques use Lloyd's algorithm [64] to steer vehicles to regions of interest to maximize reward [56]. Previous work has adapted such work to minimize the arc length of a single tour [66], while our previous work factored in path traversal costs while accounting for motion uncertainty [157]. We now extend our previous work to coordinate multiple vehicles with realistic simulations in a littoral environment. A patrol problem related to persistent monitoring includes the use of decentralized markets to coordinate teams of ASVs in protecting key assets [158].

2.16 Summary

Reviewing the mentioned literature highlights several unresolved issues that must be solved for the topics of interest. The first deficiency lies in how search based algorithms find solutions to information gathering tasks. There is no straightforward "distance to the goal" metric, and there is often greater ambiguity between selecting paths of similar reward for information gathering tasks, exacerbating A* concerted efforts at evaluating all nodes of high possible reward. Conversely, a typical poor to medium quality deep partial-solution will gather more information than the beginning portion of a rare, high quality partial-solution. It is difficult to devise and employ a heuristic that accurately estimates the amount of information that could be gathered with future search effort. This chapter also outlined the MDP formulation, which normally have path independent rewards We mentioned CEM and MCTS as techniques for solving MDPs but given their technique of using simulated rollouts, can extend to path dependent rewards.

Greedy techniques stemming from submodular function maximization and Lloyd's algorithm from Locational Optimization solve relevant information gathering tasks. However, these techniques often do not factor the environment physics, which can generate waypoints or steeering laws that are not reachable/feasible or can cause collisions. Further, coordinating large teams of vehicles for information gathering tasks has been unresolved. Approaches like M* will have great difficulty optimizing path-dependent rewards because

this dramatically increases the size of the coordination space. Techniques like SGA are quadratic in the number of agents, but it is of interest to find efficient strategies that scale well to large numbers of agents.

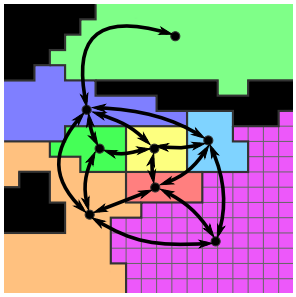
Chapter 3: Multipass Target Search in Natural Environments

This chapter is derived from work presented at the 2017 IEEE International Conference on Robotics and Automation [12] and in the MDPI Sensors Journal Special Issue on Remote Sensing and GIS for Geo-Hazards and Disasters [2].

3.1 Introduction

Consider a disaster scenario where search and rescue workers must search difficult to access buildings during an earthquake or flood. Often, finding survivors a few hours sooner results in a dramatic increase in saving lives, suggesting the use of drones for expedient rescue operations. There are several reasons why generating a multipass coverage plan is advantageous for search and rescue. Scenarios involving noisy sensors or targets that appear or disappear over time require multiple passes to correctly determine whether or not a target is present. Further, given constraints in mission duration, the agent must be able to properly decide where to search, and for how long, before returning to base. To improve the runtimes of the various planners, we

(a) Aerial view of damaged buildings in the disaster ©2015 Associated Press.



(b) Extracted hierarchical graph of the environment.

(c) Multi-pass coverage plan concentrating on areas around the buildings due to high uncertainty.

Figure 3.1: Multipass coverage plan example for the 2015 Nepal Earthquake. 3.1(a) Aerial photography illustrates the extent of the damage due to the earthquake. 3.1(b) Regions (color) with grid depicted in the lower-right region (grid enlarged for illustration). 3.1(c) Resulting multi-pass search over the area at full resolution (color indicates pass count).

suggest using a hierarchical approach. For example, in the 2015 earthquake in Nepal, Canadian relief teams used 3 small MAVs to assess the area and direct relief efforts despite a lack of ground access to villages [159]. For an example of the destruction in the Nepal 2015 disaster, an extracted graph for automated search, and an example multipass coverage plan for SAR, see Fig. 3.1.

Entropy can be used to quantify the generation and resolution of uncertainty. When searching for targets, maximizing mutual information of future sensor observations will minimize expected target location uncertainty by minimizing the entropy of the future estimate. Motion planning for multi-target autonomous search requires planning over an area with an imperfect

sensor and may require multiple passes, which is hindered by the submodularity property of mutual information. Further, mission duration constraints must be handled accordingly, requiring consideration of the vehicle's dynamics to generate feasible trajectories and must plan trajectories spanning the entire mission duration, something which most information gathering algorithms are incapable of doing. If unanticipated changes occur in an uncertain environment, new plans must be generated quickly. In addition, planning multipass trajectories requires evaluating *path dependent* rewards, requiring planning in the space of all previously selected actions, compounding the problem.

This presents several challenges which must be resolved. The algorithm must be able to react quickly to new information that can affect the search plan, suggesting an anytime formulation. An anytime algorithm is an algorithm that quickly finds a feasible solution, and continually improves the solution as available planning time permits.

The goals of this topic is to:

- Identify relevant techniques that extend to multipass coverage planning.
- Develop an any time algorithm capable of producing long duration multipass coverage plans that maximize mutual information and minimize search effort.
- Benchmark the various techniques in natural environment.

This chapter presents an anytime algorithm we developed called $\epsilon-$admissible Branch and Bound (B&B) for autonomous multipass target search for applications in search and rescue. Initial findings tested the various algorithms in maze-like rectangular environments [12]. The work was extended to more realistic environments [2] that account for sensors viewing multiple cells, the physics of the vehicle, and realistic variations in region shapes using state of the art environment decomposition techniques on simulated obstacle occupancy grids generated by gradient noise. The algorithm is capable of generating long duration dynamically feasible multipass coverage plans that maximize mutual information using a variety of techniques such as ϵ-admissible heuristics to speed up the search by more aggressively pruning the search space yet guaranteeing that the final solution is within ϵ of optimal. To our knowledge this is the first attempt at efficiently solving multipass target search problems of such long duration. The proposed algorithm is based on best first Branch and Bound and is benchmarked against state of the art algorithms adapted to the problem, gathering the most information in the given search time.

Experimental results compare the performance of the proposed ϵ-admissible branch and bound algorithm to other state of the art algorithms and also assesses how well in expectation the various algorithms guide human relief crews using a simplified effort allocation model. Findings show that ϵ-admissible branch and bound is able to gather the most information and reduces the expected time for ground crews to search a cell. Such quality plans come

at the cost of requiring the most time to compute, however. We conclude our analysis with a discussion on the low cost of supercomputing capabilities contrasted with the large value of a statistical life.

The chapter is organized as follows, following the road-map in Fig. 3.2. Section 3.1 offers an introduction and motivation to the topics presented. Section 3.2 offers a formal problem formulation for multipass coverage planning for target search. The detailed description of the algorithm presented in this chapter is divided into three related sections. Cross references are provided to establish ties between related concepts; readers may skip the cross references on an initial read. Section 3.3 introduces important preliminary concepts and introduces the benchmark algorithms. Section 3.4 outlines the proposed algorithm in the chapter, and outlines how the analysis will be conducted. Section 3.5 discusses implementation details of the algorithm outlining the simulated natural environment developed for benchmarking the algorithms. Section 3.6 covers the experiments and their results, Section 3.7 discusses the experiment results and Section 3.8 concludes and summarizes the chapter.

3.2 Problem Formulation

We define a common benchmark problem to solve multipass coverage planning problems by defining an environment (sensor model plus motion model) with an objective function to maximize. We are most interested in (1) the

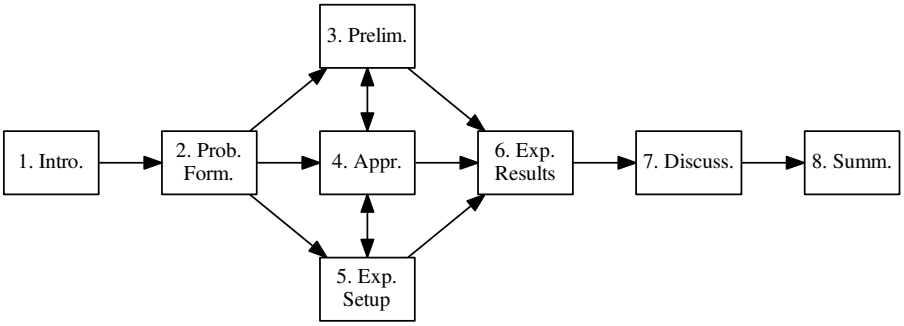


Figure 3.2: Road-map of the organization of Chapter 3.

quality of the solution the algorithm generates (how much information the algorithm gathers which in turn changes how long it takes for relief crews to search a cell for survivors) and (2) how long it takes to compute the solution.

Define $W \subset \mathbb{R}^2$ to be workspace of the robot where $w = (x, y) \in W$. Let $C = W \times \dot{W} \times \ldots \times T$ be the configuration space consisting of time domain T, workspace W and all higher order tangent spaces required to define the robot's trajectory. We assume that the robot starts at point $c_{\text{start}} \in C$. For our application, we are interested in planning in the space of *feasible* trajectories. A feasible trajectory $\mathcal{T}$ is a time indexed curve in C that starts at c_{start} and that the robot's dynamics is capable of reproducing. We denote the space of all feasible trajectories in time domain $s \in [0, t]$ with end time t as $\mathcal{T}_t$. Let $\mathscr{T}_t = \bigcup_{s\in[0,t]}\{\mathcal{T}_s\}$ be the space of all valid trajectories whose end times vary between 0 and t.

Sensor measurements used for target search are stored in a discrete evi-

dence grid $\mathcal{E}$ covering workspace W. Instead of counting the sensor measurements the agent observes, count the sensor measurements observed in each cell separately with $k = 0, 1, 2, \ldots$ Define grid cell $\mathfrak{g}_{x,y} = [x, x + r) \times [y, y + r)$ where

$$x \in \{x_{min}, x_{min} + 1, \ldots, x_{max} - 1, x_{max}\} \subseteq \mathbb{N}$$
$$y \in \{y_{min}, y_{min} + 1, \ldots, y_{max} - 1, y_{max}\} \subseteq \mathbb{N}$$

for user defined resolution r (typically, $r = 1$) where $W = \bigcup_{x,y} \mathfrak{g}_{x,y}$ such that the collection $\mathfrak{g}_{x,y}$ exactly covers the workspace.

For each cell $\mathfrak{g} \in \mathcal{E}$ define X to be the hidden state of whether or not the cell contains a target. X may be discrete or continuous. $Z_k \in \{0, 1\}$ is the kth sensor measurement in $\mathfrak{g}$ and $\mathbf{Z}_{1:k}$ is the collection of the k sensor measurements $Z_1, \ldots, Z_k$. A continuous example is the Bernoulli process where $X \sim Beta(\alpha, \beta)$ and $Z_k \sim Bernoulli(X)$. This can model targets that disappear and reappear from view randomly. Let $\mathbf{z}_{1:k}$ be an observed sequence of measurements (a realization of $\mathbf{Z}_{1:k}$). When it is necessary to denote the particular grid cell in which sensor measurements are taken we shall use superscripts, e.g., $\mathbf{z}_{1:k}^{x,y}$ is the observed sequence of measurements in $\mathfrak{g}_{x,y}$. Each cell in the evidence grid is assumed to be independent from all other cells, that is $X^{i,j} \perp X^{x,y}$ for $i \neq x$ and $j \neq y$. For all x, y, k we assume that $X^{x,y}$ is independent of time and thus $Z_k^{x,y}$ and $\mathbf{z}_{1:k}^{x,y}$ are not affected by the time(s) at which the measurements are taken. The rationale

behind this is that targets with restricted mobility (can only move within one grid cell) can be accurately modeled with a stationary process model, plus removing the dependence on time improves the ability to precompute the mutual information of future sensor measurements. Given the forward sensor model $P(Z_k = z_k|X)$, we compute the inverse sensor model using a recursive application of Bayes' Rule, (Eq. 3.1):

$$P(X|\mathbf{z}_{1:k}) = \frac{P(Z_k = z_k|X, \mathbf{z}_{1:k-1})P(X|\mathbf{z}_{1:k-1})}{P(Z_k = z_k|\mathbf{z}_{1:k-1})} \tag{3.1}$$

in which all sensor measurements are conditionally independent. From this model it is possible to compute the mutual information $I(Z_{k+1}; X|\mathbf{z}_{1:k})$. In practice, this can be computed *a priori* for any process and sensor model, and tabulated on the number of positive and negative sensor measurements observed (see Sec. 3.4 and Table 3.1 in Sec. 3.5.4). This model can account for either static (consider the occupancy grid model) or stationary targets (target's presence at any moment of time is the outcome of a Bernoulli Process, but the target's distribution is independent of time). The outlined approach in this chapter would need to be modified to be used with non-stationary targets (whose target distribution is time dependent such as a Markov chain) or dynamic targets (whose belief distribution may be in more than one cell). We will assume the use of an occupancy grid model [160]. This model has closed form solutions which are obtained by judiciously applying basic properties of probability and information theory; use of other

process/sensor models lies beyond the scope of this chapter. Define submodular function $I(\mathbf{Z}_{k+1:k+q}; X|\mathbf{z}_{1:k})$ to be the total mutual information gathered by a total of q future sensor observations. We note that in this approach, mutual information is used to predict the total information gain of future trajectories conditioned on any previous sensor measurements observed.

Given a particular configuration $c \in C$, the sensor model also defines a sensor configuration footprint, encoding which cells in $\mathcal{E}$ are observed given the perspective (field-of-view), and characteristics of the sensor (consider the projection of the image plane onto the evidence grid for a downward facing camera). This extends to trajectories, defining the trajectory footprint of the sensor following the path of the trajectory (Eq. 3.2).

$$\Phi_{\mathcal{T}} = \{\cup_{x,y} \mathbf{Z}^{x,y}_{k+1:k+q} | \mathcal{T} \text{ makes } q \text{ observations at cell } \mathfrak{g}_{x,y}\} \tag{3.2}$$

It holds that different sensors have different footprints (e.g., downward facing camera vs. scanning LIDAR). $\Phi_{\mathcal{T}}$ is equivalent to a 2D array the same size as the evidence grid by defining $\Phi_{\mathcal{T}}[x, y]$ to be the number of observations in the indexed cell. The cumulative mutual information collected by time t assuming the robot has followed $\mathcal{T}_t$ is given by the sum of the mutual information of the future sensor measurements and the hidden state conditioned on previously observed data within each cell.

$$R(\mathcal{T}_t) = \sum_{\mathbf{Z}^{x,y}_{k+1:k+q} \in \Phi_{\mathcal{T}}} I(\mathbf{Z}^{x,y}_{k+1:k+q}; X^{x,y} | \mathbf{z}^{x,y}_{1:k})$$

The problem of multipass coverage planning for information gathering with mission duration constraints is defined as follows: Find $\mathcal{T}_t^*$, the trajectory of time duration t that maximizes the cumulative mutual information in $\mathcal{E}_t$,

$$\mathcal{T}_t^* = \arg\max_{\mathcal{T}_t \in \mathcal{T}_t} R(\mathcal{T}_t). \tag{3.3}$$

3.2.1 Motion and Sensor Model

We assume a motion model $\mathcal{T}_{t_{l+1}} = \psi(\mathcal{T}_{t_l}, a_l)$ where the current trajectory and next action determine the subsequent trajectory (along with next state c and trajectory footprint). In order to minimize the number of actions required to generate a coverage plan to minimize the depth at which one must search to find the solution, we cluster the cells of the evidence grid into contiguous, connected regions. The agent then has the choice to either traverse between connected regions, or search a given region. See Fig. 3.3 for an example motion model. At a lower level hidden from the planner trying to solve (Eq. 3.3), the agent stitches motion primitives together to generate a trajectory to complete a given action (Section 3.4.3). Previously, we tested the proposed approach in rectangular environments [12]. In this work we extend such findings to more natural simulated Boustrophedon environments as will be covered in Section 3.5.2 and Section 3.5.3.

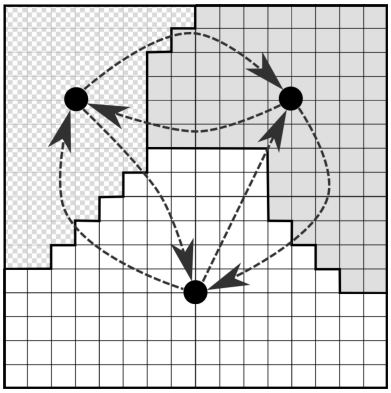


Figure 3.3: Example evidence grid divided up into 3 connected regions or nodes. The agent's motion model consists of different regions (also referred to as nodes). Each region contains a subset of cells (white, grey or checkered). A cell may belong to multiple regions, not shown. Each region has a central point (black circle) contained within the cell. The actions available to the agent at a given region is to either search the current region, or move to an adjacent region along a path connecting central points.

3.2.2 Trajectory Equivalence

The computation time required to solve (3.3) can be significantly decreased vs. a naive search over all $\mathcal{T}_t \in \mathscr{T}_t$ as a consequence of our assumption that $X^{x,y}$ is independent of time for all x, y. Consequently, the order in which observations are recorded in $\Phi_{\mathcal{T}_t}$ does not affect the information gathered. Two trajectories are in the same equivalence class if their footprints and end configurations are equal. Further, if the set of future feasible trajectories is uniquely determined by the current configuration, then two trajectories belonging to the same equivalence class generate the same future feasible trajectories. This implies that we may prune a node in the search tree if it is found to be equivalent to another node in the queue, meaning that we

only need to store one trajectory per equivalence class without affecting the quality of the final solution. In practice, this is accomplished by hashing all $(\Phi_{\mathcal{T}_t}, c)$ that we encounter to see if a new node generates a new equivalence class.

3.3 Preliminaries

Now that we are given a proper problem formulation, we now introduce the benchmark algorithms and additional required concepts before proceeding to the approach.

3.3.1 Benchmark Algorithms

In addition to our proposed algorithm, there are three additional algorithms we would like to benchmark for the described scenario, that we can classify as being either heuristic solvers or planning algorithms. It is reasonable to suppose that heuristic solvers will get solutions faster by solving an easier problem (often in polynomial time), but will get lower quality solutions. Planning algorithms, on the other hand, solve the problem at hand by reasoning over many candidate solutions. Planning algorithm performance can be improved by using heuristics to guide the search. In practice, the use of properly designed heuristics to guide the search reduces the computational effort by a significant constant factor making exponentially complex algorithms tractable, while still obtaining (near-)optimal solutions.

We evaluate two heuristic solvers including (1) a single step greedy heuristic solver, and (2) the depth first search coverage based planner which is used in covering Boustrophedon decomposed environments [47] and has been adapted to generate multipass coverage plans. For planning algorithms, we test both depth first branch and bound and ϵ-admissible branch and bound (ϵ-admissible B&B: Alg. 8 [12]). ϵ-admissible B&B uses improved heuristics to dramatically increase the quality of the solutions. Similar to ARA* [22] and e-admissible search [161], ϵ-admissible B&B uses an inadmissible heuristic to speed up the search time for branch and bound to find a good solution with bounds on sub-optimality.

The greedy heuristic algorithm greedily maximizes total information gathered per unit of time for the *next immediate action*. When the greedy heuristic is selecting the next best action, it is possible (but unlikely in natural environments) that two or more actions result in the same (maximal) amount of information gathered per unit of time. Such ties are resolved by selecting the action that acquires more total information. This is done iteratively until the planned trajectory spans the entire mission duration

For the depth first coverage solver, one uses a modified version of depth first search (DFS) to generate a tree that spans all regions in the environment. Further, the iterative greedy heuristic g computes how many times region i should be searched $S_{g,i}$ to maximize mutual information. The agent then traverses the tree based on the DFS order, keeping track of how many times it has searched each region S_i. To ensure a more uniform coverage of the

environment, the agent searches the current region at most once per visit. Traversing the DFS tree will have the agent visit most nodes multiple times, and the tree can be traversed multiple times until the time expires (provided there is enough time to search). Each region is searched at most $S_i \leq S_{g,i}$.

To illustrate the necessity of using priority heuristics and ϵ-admissible heuristics, we contrast ϵ-admissible B&B to depth first branch and bound (DF-B&B), as this is the more typical implementation of branch and bound (especially when implemented using recursion instead of a queue). Common practices with branch and bound tends to favor a depth first ordering when enumerating partial solutions because it tends to find initial (complete) solutions quicker enabling one to start bounding the space earlier in the search. Best first search strategies, such as A* with good heuristics, will evaluate fewer nodes when looking for the optimal solution however. Note that DF-B&B differs from DFS in that DF-B&B computes the heuristic for every node, where DFS finds a spanning tree of all regions which in turn defines the search order, and only computes the heuristic once to determine how many times each region should be searched. DFS therefore is much faster to compute than DF-B&B.

Note that by convention our motion model returns actions that traverse regions (sorted based on edge information) before actions that exhaustively search regions. Without modification, DF-B&B will only focus on actions that traverse between many regions and find low quality solutions within the allotted time. To generate high quality solutions quickly, the action

set ordering should be randomized (i.e. permuted) such that DF-B&B will deliberate over more diverse action sequences. Such permutation schemes should be complicated enough to prevent cycles in the action space. This modification remains within the scope of the depth first search algorithms since the motion models do not require a preference over the ordering of different actions.

Zelinsky's algorithm [46] is a value function based uniform coverage planning algorithm that plans directly on the 8-connected evidence grid. It may occasionally back itself into a corner, in which case we use Dijkstra's algorithm to plan a route to the nearest unvisited cell, and then continue running Zelinsky's algorithm. We note that Zelinsky's algorithm does not readily extend to sensor models whose footprints extend beyond a single cell in the grid, so it will not be benchmarked in environments with sensor models that observe more than one cell.

We now discuss various algorithms that solve related problems yet are unsuitable for the task at hand. In the absence of good heuristics, Greedy Best First Search (GBFS) [23] is often a good candidate as it will prioritize nodes that are of lowest cost to the goal state for shortest path problems. In terms of information gathering tasks, there is no "distance to the goal" metric, so GBFS corresponds to selecting nodes that have gathered the most information so far. This behavior is similar to how Depth First Search works, since on average deeper solutions will gather more information. More specifically, a typical deep solution will gather more information than the beginning por-

tion of a rare, high quality solution. Therefore, DF-B&B will be an effective stand-in for GBFS. Other algorithms of interest that solve similar problems are not suitable for the problem include:

- submodular orienteering problem solvers are not suitable because these solvers cannot visit the same area multiple times [38].
- MPC based techniques that gather submodular/path dependent reward have too short of a time horizon [120].
- Convex methods are not suitable because the problem is not convex. The domain of feasible trajectories is not a convex set (due to obstacles), therefore the reward function is not necessarily well defined for convex combinations of feasible trajectories which may go through obstacles.
- Brute force methods that enumerate all possible trajectories [10] are intractible for this problem due to the extreme length of the selected action sequences. Even with small to moderate branching factors and the hierarchical structure, deep solution spaces are prohibatively large.
- Monte Carlo based techniques, while good for finding reasonable solutions to large dimensional problems with little known structure, cannot leverage the hierarchical nature of the proposed problem the way search algorithms can. Considering the simple search technique of depth first traversal of a graph, one would be hard pressed to construct a sample

space in which Monte Carlo samples would be more efficient and find better solutions than search based techniques.

3.3.2 Heuristics

Several challenges exist in designing heuristics for information gathering for multipass trajectories that do not exist in path planning for waypoint following:

- All regions must be explored (possibly multiple times), requiring the heuristic to be more computationally demanding and explore a larger portion of, if not the entire workspace.
- The reward function is *path dependent* since mutual information is submodular, requiring planning in the space of trajectories instead of the configuration space.
- There is an increased number of paths of equal or similar reward, further compounding the problem.

To counter these challenges we define the notion of an ϵ-admissible heuristic for branch and bound in previous work, and observe that at significant performance speedup we can more greedily prune the solution space with a bounded loss in optimality [12]. For our application we effectively transform the admissible iterative greedy heuristic g into an ϵ-admissible $\tilde{g}$ (see Section 3.5).

We now describe the iterative greedy heuristic developed for multipass search in previous work [12]. We relax the problem formulation in (Eq. 3.3) by permitting sequences of actions that are not feasible. In other words, regardless of the agent's current state, the agent must select L discrete actions to take, from all N_a actions that are available from any state (Note that to guarantee admissibility N_a is the sum of all actions that transition between regions and all actions that search a given region, Section 3.2.1.).

Define decision variable $a_i \in \{0, 1, \ldots, L\}$ with $i \in 1, \ldots, N_a$ the number of times the ith action is taken, with decision vector $\mathbf{a} = (a_1, a_2, \ldots a_{N_a})$. For the multipass coverage planning problem, actions can be selected multiple times. For example, selecting action a_i j times implies $a_i = j$. Selecting action i the $(a_i + 1)$th time yields the incremental reward of $\Delta R_i(a_i + 1)$. R_i is monotone increasing and concave ($R_i(a + 1) \geq R_i(a)$ but $\Delta R_i(a + 2) \leq \Delta R_i(a + 1)$). The agent's goal is to select $\mathbf{a}$ to maximize the function:

$$\max_{\mathbf{a}} \sum_{i=1}^{N} R_i(a_i) \quad \text{s.t.} \quad \sum_{i=1}^{N} w_i a_i \leq L \text{ and } a_i \geq \alpha_i \geq 0 \tag{3.4}$$

The strategy to find the optimal solution is to sort all actions based on the incremental reward. While picking one action may affect the reward of other actions, those actions affected are a small subset of all N_a actions and can be recomputed, since information content is localized to a given cell, and the information in cells may be accessed by different actions. During each round the action with the most reward is selected. By the concavity of the

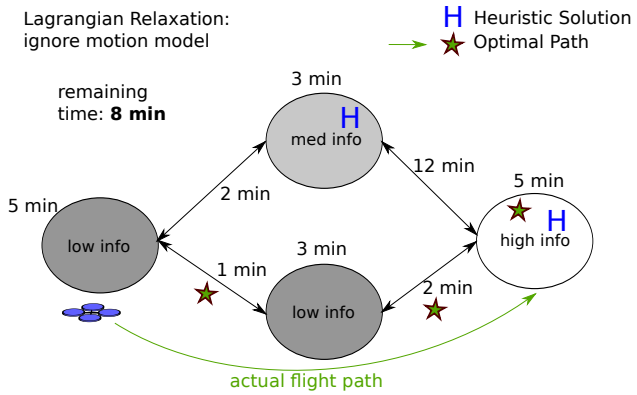


Figure 3.4: An illustration highlighting how the heuristic is an upper bound. Relaxing the motion model is akin to allowing the agent to move freely between regions ignoring the motion model. The heuristic is able to sample the regions with the most info given the mission duration but ignores path constraints, such as having to navigate the 12 minute edge. The optimal solution is to traverse the bottom of the environment.

reward function the reward for taking an action a the ith time is collected before taking action a the $(i+1)$th time [12]. An illustration of how the iterative greedy heuristic can be found in Fig. 3.4.

This formulation is similar to previous work that uses multi-armed bandit (MAB) relaxations for gathering information [162]. However, the MAB heuristic computes dynamic allocation indices [163] in scenarios where reward is uncertain. The iterative greedy heuristic gathers expected information gain, which automatically factors measurement uncertainty, offering a simpler approach for the problem at hand.

One of the main limitations to the iterative greedy heuristic is that it does not account for flight time associated with the agent transitioning between region. This means that many different distributions of information

will have the same heuristic reward even though it will be easier to gather the reward in some scenarios and not in others. Algorithms like A* or best first branch and bound expand all nodes with optimistic reward greater than or equal to the optimal reward, dramatically increasing search time if there are many paths of similar cost. One might consider getting a tighter bound on total information a node can gather with the remaining time available before the mission ends by estimating the total remaining traversal time. One such way to obtain a lower bound is to generate a minimum spanning tree [29] of all regions the heuristic visits. While this approach works for traveling salesman problems [28], such an approach cannot properly bound the information gathered when traversing between regions, making it unsuitable for the problem at hand.

Further, in large maze-like environments, it is often difficult for the planner to explore the search frontier when a candidate solution backs itself into a corner and requires multiple actions to return to the frontier, which is likely to happen when a multipass algorithm is used in large environments.

The iterative greedy heuristic requires the evaluations of many footprints for their information content. In order to speed this up, we first note that changes in cost can be computed locally for the region being evaluated immediately. Second, caching the rewards for footprints is essential to improving the speed of the heuristic. Whenever a region is selected greedily, at the next iteration the rewards only have to be recomputed for the previously selected region (and any other regions it may overlap), resulting in a constant factor

performance improvement with N_r. Caching is therefore a constant factor improvement in the run time of the heuristic.

3.4 Approach

In order to solve (Eq. 3.3), we benchmarked four different algorithms using the same reward and motion model to determine their ability to generate quality solutions and determine how long it takes them to compute the solution. As we did in previous work [12], we observe that $I(\mathbf{Z}_{k+1:k+q}; X|\mathbf{z}_{1:k})$ can be computed from $I(Z_{k+1}; X|\mathbf{z}_{1:k})$ as a 3D look-up table dependent on the number of positive and negative observations for a given cell by applying the chain rule for mutual information [31] and the exchangeability property of Binomial random variables, offering $\mathcal{O}(1)$ mutual information look-up speeds in RAM with as little as 32 kilobytes of memory for a 20x20x10 array of double precision floats.

Our proposed method is ϵ-admissible branch and bound [12], is based off of branch and bound [164] and is summarized in Algorithm 8 with subroutines BRANCH (Alg. 9) and BOUND (Alg. 10). In short, branch and bound works by iteratively partitioning the solution space (branch), and determining if a subspace contains a solution that can potentially beat the current best known solution using an admissible heuristic (bound). Subspaces that can be determined to not contain the best solution are pruned or *fathomed*, speeding the search. When expanding the search tree denoting subsets of

Algorithm 8 ϵ-admissible Branch and Bound

Require: Priority Queue Q_{open}, Queue Q_{closed}, evaluation function $f(\cdot)$, ϵ-admissible heuristic $\tilde{g}(\cdot)$, priority heuristic $P(\cdot,\cdot)$, Start node N_{start}, end time $Tmax$, maximum number of iterations $Imax$, heuristic reward B_0

1: $B \leftarrow B_0$
2: $Q_{open}.insert(N_{start})$
3: $i \leftarrow 0$
4: **while** Q_{open} is not empty or $i < Imax$ **do**
5: $i \leftarrow i + 1$
6: $N \leftarrow Q_{open}.pop()$
7: $Q_{closed}.insert(N)$
8: **if** N is a complete candidate solution and $f(N) > B$ **then**
9: $B \leftarrow f(N)$ {new best solution found. Store it.}
10: **end if**
11: $\mathcal{N} \leftarrow BRANCH(N)$
12: **for all** $N_i \in \mathcal{N}$ **do**
13: **if** $BOUND(N, B, \tilde{g})$ and (N not in Q_{open} and N not in Q_{closed}) **then**
14: $N_i.priority \leftarrow P(N_i, \tilde{g})$
15: $N_i.parent \leftarrow N$
16: $N_i.R \leftarrow f(N_i)$
17: $Q_{open}.insert(N_i)$
18: **end if**
19: **end for**
20: **end while**

the solution space, define a search node N to be a partial solution with a family of candidate solutions, combined with a reference to its parent node, its priority, and other quantities required for improving the search. Search node N is different from a node in the graph of the motion model. DF-B&B can be created by modifying Alg. 8 such that Q_{open} is a stack (LIFO queue), priority heuristic $P(\cdot,\cdot)$ is no longer required, line 14 in Alg. 8 is eliminated, and admissible heuristic g is used in place of ϵ-admissible heuristic $\tilde{g}$.

Algorithm 9 BRANCH(N)

Require: motion model ψ
1: **for all** available actions i from ψ **do**
2: $\mathcal{T} = \psi(N.\mathcal{T}, i)$ {roll out trajectory using motion model}
3: $N_i.\mathcal{T} \leftarrow \mathcal{T}$ {append to new search node}
4: **end for**
5: **return** $[N_1, N_2, \ldots, N_m]$

Algorithm 10 BOUND(N, B, g)

Require: Search node/partial solution N, current best reward B, heuristic $g(\cdot)$
Ensure: determination if N should be expanded.
1: **if** $g(N) > B$ **then**
2: **return** True {current partial solution can potentially beat best known solution}
3: **else**
4: **return** False {current partial solution cannot beat best known solution}
5: **end if**

For completeness, we start off with determining that branch and bound is complete. We then show that ϵ-admissible branch and bound will find a solution that is within ϵ of optimal.

Theorem 1 *branch and bound is complete when X is finite and is guaranteed to find the optimal solution.*

Proof: Branching without bounding will eventually generate all problem instances $x \in X$, effectively becoming exhaustive enumeration of all feasible solutions by setting $g(N) = \infty$, $\forall N \subset X$. Note that the order in which the problem sets N_j are evaluated does not affect the completeness of the algorithm. For $g(N) < \infty$, bounding occurs speeding up the exhaustive search. Suppose the optimal solution is x^* with $f(x^*) = B^*$. Given $B < B^*$ and N_i

s.t. $g(N_i) \leq B$ x^* *is guaranteed to not be within* N_i *otherwise that would violate the admissibility property of* g*. Therefore* N_i *can be fathomed without affecting the completeness of branch and bound. Alternatively, proof for an equivalent formulation can be found in Edelkamp and Shroedl [164].* □

Suppose that instead of using an admissible heuristic for B&B we wish to speed up the search by more aggressively fathoming the search space. Define ϵ-admissible bounding heuristic $\tilde{g}$ which satisfies (Eq. 3.5):

$$\tilde{g}(N) \geq \max_{x \in N} f(x) - \epsilon \tag{3.5}$$

which underestimates the reward $f(x)$ by at most ϵ. This is closely related to Harris's *e-admissibility* criterion for bandwidth search [161]. ϵ-admissible branch and bound is therefore defined as branch and bound that uses an ϵ-admissible heuristic for bounding.

Theorem 2 *Using* ϵ*-admissible heuristic* $\tilde{g}$ *for bounding in B&B guarantees that the solution will be at most* ϵ *less than the optimal solution.*

Proof: The only way this algorithm will not find the optimal solution x^* *is if any subset* N_j*, where* $x^* \in N_j$*, is pruned. Suppose* $\tilde{x}$ *is a feasible solution and* $f(\tilde{x}) = B$*. Given that* N_j *is pruned,* $\tilde{g}(N_j) \leq B$ *but* $\tilde{g}(N_j) \geq f(x^*) - \epsilon$*. Further manipulation yields* $\epsilon \geq f(x^*) - f(\tilde{x}) \geq 0$*.* □

3.4.1 Priority Heuristic

For the priority heuristic in ϵ-admissible branch and bound (Alg. 8 line 14), we use the weighted heuristic that discounts future reward to offer a compromise between best first and depth first ordering. For a node N in the search tree, define priority $P(N, g)$ to be (Eq. 3.6) [12]:

$$P(N, g) = R(N) + \alpha(g(N) - R(N)) \tag{3.6}$$

given current reward $R(N)$, admissible bounding heuristic $g(N)$, and future reward discount factor $\alpha \in [0, 1]$. Setting α towards zero biases the search towards depth-first like behavior.

3.4.2 Agent Model

While this work is targeted at quadrotors, the outlined approach generalizes to other physical systems. Although quadrotors are a differentially flat system [106] requiring 4th order dynamics, we will focus on using a second order integrator subject to velocity and acceleration constraints for this work. We first observe that physical systems have bounds on their acceleration capabilities due to the force output of their actuators, or the maneuvers their controllers can reliably execute, imposing a maximum nominal acceleration on the platform a_{max}. Lower accelerations for quadrotors reduces the maximum attitude angles observed (similarly with their derivatives), which can

affect sensor measurements. Second, we observe that it is often the stopping distance vs. the range of onboard collision detection sensors that dictates how fast a vehicle can safely fly in partially known environments, requiring a hard bound on maximum nominal velocity v_{max}. When traveling at maximum speed v_{max}, decelerating at a_{max}, the stopping distance before the system comes to rest is $d_{stop} = \frac{v_{max}^2}{2a_{max}}$. The vehicle can safely stop if its stopping distance is shorter than its sensing range. Alternatively, if the vehicle wishes to steer around an obstacle d_{obs} distance ahead maintaining forward speed, the distance the robot moves laterally before closing the distance to the obstacle, $d_{steer} = \frac{a_{max} d_{obs}^2}{v_{max}^2}$. d_{steer} must exceed the width of the obstacle to avoid collision. In short, a vehicle that wishes to double its safe forward speed must quadruple its acceleration capabilities or sensing range. Further, analysis of single order integrator systems flying through Poisson forests (distributions of circular obstacles that are generated by a Poisson Process) suggest that above a certain speed, collision will occur with probability one regardless of the planner or controller used [165].

Define $||\cdot||$ as the L2 norm while $\mathbf{r} = [r_x, r_y]'$. Other flat states such as altitude or yaw could be included but are ignored for our work.

$$\ddot{\mathbf{r}} = \mathbf{u} \text{ s.t.} \tag{3.7}$$
$$||\dot{\mathbf{r}}|| \leq v_{max}$$
$$||\mathbf{u}|| \leq a_{max}$$

3.4.3 Motion Primitives

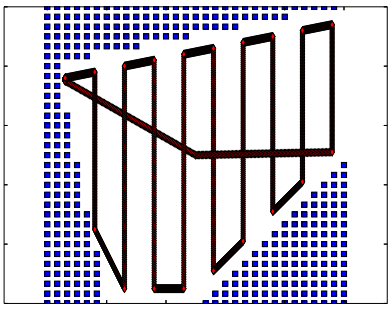


Figure 3.5: Example generation of rectilinear coverage plan for a region with continuous velocities.

In order to execute a high level behavior, sequences of waypoints $\mathbf{r}_i$ are chained together to create multiple point to point maneuvers. For now we assume that a decomposition algorithm divides the environment into contiguous regions that are easy to search (how we do this is covered in Section 3.5.3). When traversing between regions (central point to central point), a simple line connecting the regions may not suffice due to the presence of obstacles. To solve this, we create an alternate motion model that is an 8 connected graph on the underlying evidence grid. A* can generate a sequence of cells minimizing path cost (path length). However, this sequence of waypoints is very dense; many of the waypoints are unnecessary to define the shape of the path and can be eliminated by the following method. One only keeps a waypoint $\mathbf{r}_i$ in the sequence if the angle between vectors $(\mathbf{r}_i - \mathbf{r}_{i-1})$ and $(\mathbf{r}_{i+1} - \mathbf{r}_i)$ is above some threshold, e.g. 5°. Otherwise, remove waypoint i and continue.

In order to search a region, a dynamically feasible coverage plan is needed. The proposed method creates rectilinear uniform coverage plans (Fig. 3.5) with "plow lines" that are parallel to the slice direction defined by the Boustrophedon Decomposition algorithm (Section 3.5.3). The agent transitions between adjacent "plow lines" rectilinearly. In order to compute the start and stop positions of the plow line, the geometry of the (boustrophedon) region is known, and line search is conducted to find the max and min values along a given plow line. Spacing between plow lines is dictated by the maximum sensing radius.

The use of a uniform coverage plan makes the tacit assumption that information is uniformly distributed within the region. This may or may not be the case in practice, reducing the effectiveness of the information gathering algorithm. Having a coverage planner that can directly reason over contiguous nonuniform regions is compatible with the proposed approach, but is to be considered in future work. An alternate approach is to use the distribution of information to guide the selection of regions. The mere act of intelligently decomposing the environment into separate regions can dramatically simplify the problem by solving information gathering problems in smaller contiguous regions. We note that for illustrative purposes, all algorithms benchmarked in this chapter have access to the same motion model, suffering similarly.

Apart from rectilinear coverage plans, one might consider using spirals [48] due to their ability to efficiently cover 2D areas. However, given the

problem specification of having to account for obstacles while generating dynamically feasible coverage plans, The task of covering non-circular/non square regions found from a decomposition algorithm makes spiral based coverage plans impractical due to inefficiencies in covering the edges.

3.4.4 Point to Point Maneuver

For vehicle trajectory $\mathbf{r}(t)$ connecting points $\mathbf{r}_0$ and $\mathbf{r}_1$ for times $t_1 > t_0$, solve the following optimization problem:

$$\begin{aligned} \min_{\mathbf{u}} \; & t_1 - t_0 \text{ subject to (3.7) and:} \\ & \mathbf{r}(t_0) = \mathbf{r}_0 \\ & \mathbf{r}(t_1) = \mathbf{r}_1 \\ & \dot{\mathbf{r}}(t_0) = \ddot{\mathbf{r}}(t_0) = \dot{\mathbf{r}}(t_1) = \ddot{\mathbf{r}}(t_1) = 0 \end{aligned} \tag{3.8}$$

For the system in (3.7), the problem in (Eq. 3.8) is a simple bang-bang control problem [1] minimizing the trajectory time interval $t_1 - t_0$ by selecting optimal control $\mathbf{u}$. The corresponding trajectory can be written in closed form with continuous position and velocity, and piece-wise constant acceleration at a_{max}. The system either has one or two switching times depending on whether the system has enough time to accelerate to its maximum velocity.

Without loss of generality we will consider a scalar model, letting $r = r_x$ with dynamics $\ddot{r} = u$. Let $\Delta r = r(t_1) - r(t_0)$, $\dot{r}(t) = \upsilon(t)$ and $\ddot{r}(t) = a(t)$.

Theorem 3 *Necessary condition: the time optimal control resides within the domain* $u \in \{-a_{max}, 0, a_{max}\}$

Proof: There are two cases, when there exists 1 switching time (when $|v(t)| < v_{max}$ *and* t *is not the switching time or when* $|\Delta r| \leq r_{crit}$*) or when there are 2 switching times (*$|\Delta r| > r_{crit}$*) for* $r_{crit} = \frac{v_{max}^2}{a_{max}}$*. When* $|\Delta r| \leq r_{crit}$*, one can show by using the Pontryagin's Maximum Principle (PMP) [1] that control for the time optimal solution* $u \in \{\pm a_{max}\}$*.*

When $|\Delta r| > r_{crit}$*, the velocity constraint will become active during the trajectory. As a simple illustration, suppose that the system accelerates to the maximum velocity* v_{max}*. To continue accelerating would violate the velocity constraint, so the system must either stop accelerating or slow down. For any system slowing down before the prescribed deceleration at the end of the trajectory, the traversal time would increase, yielding a suboptimal solution to the time optimal problem. Therefore, the system will cruise at* v_{max} *with* $u = 0$ *until it is time to slow down and* $u \in \{-a_{max}, 0, a_{max}\}$*.* □

For a more rigorous proof, consider a formulation of Pontryagin's Maximum Principle with state inequality constraints to get the case when the velocity constraint becomes active. Typical formulations of PMP without inequality constraints will not elicit costate trajectories that have multiple switching times.

With the domain of u and known problem constraints, the switching times can be determined along with the optimal control by inspection (speed up,

cruise, slow down). With known control u, $v(t)$ and $r(t)$ can be solved for closed-form by quadrature. To increase the average speed of the vehicle, it is advantageous to remove the zero velocity and acceleration constraints at the end of each maneuver. Boundary conditions with $\ddot{\mathbf{r}}(t_0), \ddot{\mathbf{r}}(t_1) \neq \mathbf{0}$ are easy to solve for closed form using the mentioned approach when the velocity is in the direction of the vector $\mathbf{r}(t_1) - \mathbf{r}(t_0)$. Two extensions are to use splines (where it is difficult to guarantee hard bounds on velocity and acceleration) or to use Dubbins like trajectories where the use of circles connect adjacent plow lines.

3.4.5 Search Effort Allocation Model

For comparison of results between algorithms, we define a simplified search effort allocation model to see how effective the various multipass coverage planning algorithms are at directing rescue teams. This simplified model can study the practical consequences of having informative trajectories for search and rescue but ignores effects due to finite search resources/time, survivor survival rates, and the time spent transitioning between cells. The search effort allocation model is *distinct* from the sensor model used by the planner based on the different attributes or roles the autonomous agent and human ground crews have during the search and rescue operation. We assume that after executing the autonomous search using one of the benchmark algorithms outlined in Section 3.3.1, the agents generate an occupancy map which become prior probabilities $P_0 = \Pr(X = 1)$ for the human rescue teams on the

ground. Let D be the event that the target is detected within a given cell, while $\neg D$ is the absence of detection. X is the presence or absence of a target in the cell as in the previous model (Eq. 3.1). Variable t denotes the time spent searching, while τ is the detection time constant effectively rescaling the given search time to search effort. Define the probability of detection of a target in a given cell (Eq. 3.9) [32, 166]:

$$\Pr(D|X = 1, t) = 1 - e^{-t/\tau} \tag{3.9}$$

Using Bayes's rule and (Eq. 3.9), we compute the probability the target is still present despite there not being a detection after searching for t seconds (Eq. 3.10):

$$\Pr(X = 1|\neg D, t) = \frac{P_0 e^{-t/\tau}}{1 - P_0 + P_0 e^{-t/\tau}} \tag{3.10}$$

The human rescue teams search the cell for T units of time for a survivor until the survivor is found or $\Pr(X = 1|\neg D, t) < P_{neg}$. If $P_0 < P_{neg}$, the cell is skipped. Using this policy, the maximum search time T_{neg} for the cell is (Eq. 3.11):

$$T_{neg} = -\tau \log \frac{P_{neg}(1 - P_0)}{P_0(1 - P_{neg})} \tag{3.11}$$

Note that the probability density function $f_{T|X=1}(t) = \frac{e^{-t/\tau}}{\tau}$ is due to (Eq. 3.9). We wish to compute the expected time for the search in a given cell to end by letting $\mathrm{E}[T] = \sum_p \mathrm{E}[T|P_0 = p]\Pr(P_0 = p)$ for a given trajectory generated by the algorithm. We start off by computing the expected time to

search the cell given prior probability P_0 $\mathrm{E}[T|P_0]$ by using the law of total expectation over X (Eq. 3.12):

$$\mathrm{E}[T|P_0] = \mathrm{E}[T|P_0, X=0]\Pr(X=0) + \mathrm{E}[T|P_0, X=1]\Pr(X=1) \quad (3.12)$$

$$\mathrm{E}[T|P_0] = T_{neg}(1-P_0) + \left(\tau - e^{-T_{neg}/\tau}(T_{neg}+\tau)\right)P_0$$

This leaves the computation of the distribution of priors P_0. For each cell in the evidence grid, the distribution of outcomes for $P_0 = \Pr(X=1|K=k, \mathbf{z}_{1:k})$ given the sensor data (planned K sensor measurements) can be computed for all realizations. P_0 can be computed directly/deterministically from the known number of sensor measurements and known sensor data $\mathbf{z}_{1:k}$. Since we assume that all cells have the same characteristics, we marginalize over the distribution of sensor measurements (computed from the trajectory footprint) and marginalize over all realizations. We also use the exchangeability property of the sensor measurements Z_i to use a Binomial random variable $M_k = \sum_{i=1}^{k} Z_i$ (Eq. 3.13):

$$\Pr(P_0 = p) = \sum_{i,k} \Pr(X=1|K=k, M_k=m_i)\Pr(M_k=m_i|N=k)\Pr(K=k) \quad (3.13)$$

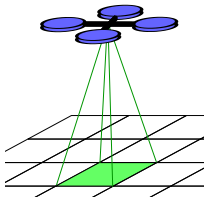
Without loss of generality we assume $\tau = 1.0$, meaning t implies the number of time constants spent searching a given cell.

Having analyzed the search effort allocation model, the trade-offs between

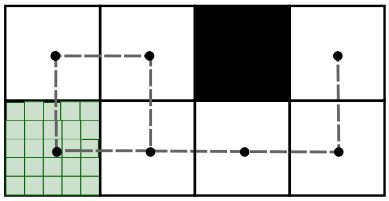
the added cost of computing better paths to the increased rate of rescuing survivors can now be discussed. We observe that the dramatic reduction in costs for supercomputing capabilities in recent years combined with the high value of a statistical life (around $7M in 2005 USD [167]) imply that computer time is extremely cheap relative to human lives. Spending an additional $70,000 renting a sizeable chunk of supercomputing resources is considered cost effective if it increases the expected number of statistical lives rescued by at least 0.01 lives. Increasing the amount of area one is able to thoroughly search within 48 hours by a few percent is instrumental to increasing survival rates. More accurate figures are required for detailed analysis.

3.5 Experimental Setup

We now discuss the formulation for modeling quadrotors with downward facing cameras in natural environments, and discuss experimental setup. When selecting our ϵ-admissible heuristic for ϵ-admissible branch and bound we only want the algorithm to expend effort when there may exist a solution that is at least a factor $1 + \eta$ better than the current best known solution. We let $\tilde{g}(x) = g(x) - \eta B$ where B is the reward of the current best known solution. $B \leftarrow -\infty$ at start up if no heuristic solution is known or computed. Unless otherwise stated we let $\eta = 0.5\%$. Simulations were run on a workstation laptop running 64 bit Ubuntu 14.04 LTS with a Core i7-6920HQ which



(a) sensor model used in the simulations



(b) Example environment denoting regions, the connectivity graph and a portion of the evidence grid.

Figure 3.6: Rectangular Environment sensor model and environment model. 3.6(a): Sensor model observes a single cell in the occupancy grid. 3.6(b): example 7 region environment showing regions (black lines), their nodes (black circles) and edges (gray dashed lines) overlaying the evidence grid. Example portion of the evidence grid is shown in the bottom left region.

has a 2.9 GHz clock with 64 GB RAM. The proposed branch and bound algorithm, the greedy algorithm and the depth first search (DFS) coverage planner were developed in Python, using NumPy [168] for array operations and networkx [169] for graph operations.

3.5.1 Rectangular Environments

We now describe the rectangular environments used for rectangular experiments. The simplified sensor model only makes a single observation in an individual cell in the evidence grid (Fig. 3.6(a)). We procedurally generate environments randomly by creating rectangular regions that produce a tiling on the evidence grid (Fig. 3.6(b)). We randomly remove a fixed set of individual regions while maintaining graph connectivity. Environments will consist of either 12, 24, or 50 regions within the evidence grid. The 12 region

environment started with 4x4 regions, while the 24 and 50 region environments started out with 8x4 and 10x10 regions respectively. Note that in this case, the 12 and 24 region environments have 15k cells accessible for observation, while in the 50 region case there are 10k accessible cells. The agent has the following actions: (1) traverse between connected regions, e.g. move left, down; (2) exhaustively search a region, or search the region until the time horizon expires. Each action has a given time duration and a trajectory footprint where one unit of time is required to traverse and sense one cell, and starts and ends at one of the graph nodes.

3.5.2 Procedurally Generated Natural Environments

In order to procedurally generate natural environments, we use a multiple frequency bands of a gradient noise function (for a single band: Fig. 3.7(a)) to generate elevation maps (Fig. 3.7(b) is a colored elevation map, while Fig. 3.7(c) is a 3D perspective of the colored elevation map in Fig. 3.7(b)), and then threshold that map to create an obstacle grid (Fig. 3.7(d)). In contrast to Perlin noise, Simplex noise (Fig. 3.7(a)) has minimal directional artifacts making it most suitable for natural terrain [170,171] (we use the OpenSimplex algorithm developed by Kurt Spencer [172] and ported to Python by A. Svensson [173]).

The frequency content and hence the complexity of the environment can be controlled by summing multiple frequency components (e.g. Fig. 3.7(a)). OpenSimplex defines the 2D gradient noise function $S(x, y, \text{seed})$

where $(x, y) \in \mathbb{R}^2$ but we restrict to integer grid coordinates given a seed value for the random number generator. $T(x, y, \text{seed}) = \sum_{i=1}^{n_f} w_i S(f_i x, f_i y, \text{seed})$ defines the elevation map. n_f is the number of frequency components in the terrain generator, and f_i is the frequency of the ith band and w_i is the weight of the ith band.

For our experiments, we set $n_f = 2$, $w_1 = 1$ and $w_2 = 0.25$. Define 5 different environment frequency contents: very low freq. (vlf), low freq. (low), medium freq. (med), high freq. (high) and very high freq. (vhf). For (vlf,low,med,high,vhf), the frequencies f_1 correspond to $(0.015, 0.03, 0.045, 0.06, 0.75)$ and the frequencies f_2 correspond to $(0.05, 0.1, 0.15, 0.2, 0.25)$. For each environment complexity, our database of benchmark environments consisted of 20 Simplex environments and 4 random starting locations per environment. Algorithm 11 outlines the approach used to generate Simplex environments. This is contrasted to defining random obstacles over, e.g. randomly generated quadtree maps [174].

3.5.3 Boustrophedon Decomposition

Given an environment, the free space must be decomposed into individual components that can be searched or "plowed" using the *Boustrophedon* (i.e. "the way of the ox") decomposition algorithm [47]. We use a discrete implementation with the following modifications:

- We ignore small obstacles under a given pixel count. We argue that

Algorithm 11 GENERATE_ENVIRONMENTS(T)

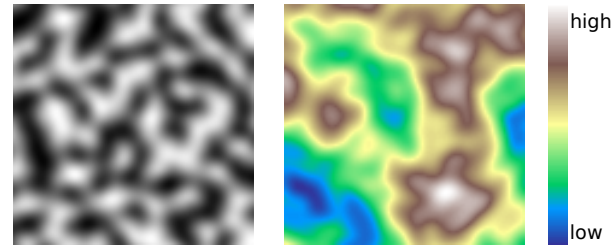
Require: Noise function T
Ensure: set of environments
$Q \leftarrow \emptyset$
for all RNG seed i from 1 to N **do**
  for all coordinates (x, y) in the environment **do**
    $E_i[x, y] \leftarrow T(x, y, i)$
  end for
  Threshold E_i such that 66% of the workspace is collision free.
  Select the largest connected component and set it as free space. Make all other regions obstacles.
  if amount of free space is within tolerance **then**
    Remove obstacles below the minimum size threshold in pixels in E_i.
    add E_i to Q
  end if
end for
return Q

small obstacles smaller than the sensing radius do not adversely affect the coverage plan yet including a small convex obstacle introduces 3 additional regions to the decomposition.

- We impose a greedy merge rule to combine adjacent regions if their boundaries overlap sufficiently due to critical points far away causing otherwise continuous regions to be broken into smaller regions.

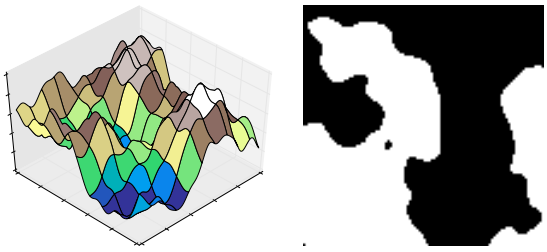
- We also remove regions that are too small to sensibly cover and repair region connections afterwards.

- Since we will be generating trajectories that cover the region, the Boustrophedon algorithm identifies start and end points for each plow line



(a) high frequency Simplex noise

(b) example elevation map with legend



(c) 3D perspective of elevation

(d) binary thresholded environment

Figure 3.7: Example Simplex noise using OpenSimplex noise and illustrations on generating natural environments. Fig. 3.7(a) shows 1 frequency band of Simplex noise, while multiple frequency bands create the environment in Fig. 3.7(b)-3.7(d). The color elevation map in Fig. 3.7(b)

for the trajectory planner.

For the first $N = 40$ computed environments, for the frequency content environments (low,med,high,vhf), the average/standard deviation of number of regions was $(13 \pm 4.3, 28 \pm 5.1, 37 \pm 5.8, 45 \pm 6.8)$. Simplex environments are more complex than previous environments [12] due to increased number of loops, and the fact that region traversals gather a significant amount of reward and create significant overlaps between actions. Observe in Fig. 3.8 that the regions vary widely in size given the complexity of the environment.

Having fewer nodes in the environment is advantageous in speeding up the search. Larger contiguous regions can be searched more efficiently by acceleration limited agents (Section 3.4.2). One disadvantage to having such large regions is that the information content of the region may not be uniformly distributed; the coverage planner may not be able to account for this.

3.5.4 Camera and Sensor Model

For our experiments, we will restrict the sensor model to a binary Bayes static occupancy grid model [160] where $P(Z|X)$ is fully characterized by probability of detection $P(Z = 1|X = 1) = p_d$ and false positive rate $P(Z = 1|X = 0) = p_f$. We let $(p_d, p_f) = (0.85, 0.15)$. Unless otherwise mentioned we represent the target evidence grid $\mathcal{E}$ with a 200x100 grid.

When computing the recursive Bayes filter (3.1) for the binary Bayes static occupancy grid model, define $\theta_k = P(X = 1|\mathbf{z}_{1:k})$ with θ_0 defined using prior knowledge (we typically assume least informative prior $\theta_0 = 0.5$) for new sensor observation z_k, resulting in (3.14):

$$\theta_k = \frac{P(Z_k = z_k|X = 1)\theta_{k-1}}{P(Z_k = z_k|\mathbf{z}_{1:k-1})} \tag{3.14}$$

This results in parameter update equation (Eq. 3.15):

$$\theta_k = \begin{cases} \frac{(1-p_d)\theta_{k-1}}{(1-p_d)\theta_{k-1}+(1-p_f)(1-\theta_{k-1})} & \text{if } z_k = 0 \\ \frac{p_d\theta_{k-1}}{p_d\theta_{k-1}+p_f(1-\theta_{k-1})} & \text{if } z_k = 1 \end{cases} \tag{3.15}$$

In order to compute the multi-step look-up table $I(\mathbf{Z}_{k+1:k+q}; X|\mathbf{z}_{1:k})$ for the binary Bayes static occupancy grid model, we can take a more direct approach using a Binomial random variable to represent the next q observations instead of using the chain rule of mutual information. Let $M_{k,q} = \sum_{i=k+1}^{k+q} Z_i$ be a Binomial RV where:

$$\Pr(M_{k,q} = m|X) = \binom{q}{m} \Pr(Z = 1|X)^m \Pr(Z = 0|X)^{q-m}, \text{ for } m \in 0, 1, \ldots q$$

with $\Pr(X = 1) = \theta_k$ known from the Bayes filter, $\Pr(M_{k,q} = m) = \Pr(M_{k,q} = m|X = 0)\Pr(X = 0) + \Pr(M_{k,q} = m|X = 1)\Pr(X = 1)$ using law of total probability. Mutual information for discrete random variables X,Y $I(X, Y) = \sum_{x,y} f_{X,Y}(x, y) \log_2 \left(\frac{f_{X,Y}(x,y)}{f_X(x) f_Y(y)}\right)$ with $\Pr(X = x, Y = y) = f_{X,Y}(x, y)$ being the joint probability mass function of X, Y and $f_X(x), f_Y(y)$ being the marginal probability mass functions of X, Y. Manipulation yields (3.16):

$$I(\mathbf{Z}_{k+1:k+q}; X|\mathbf{z}_{1:k}) = \sum_{m,x} f_{M|X}(m|x) f_X(x) \log_2 \left(\frac{f_{M|X}(m|x)}{f_M(m)}\right) \tag{3.16}$$

Assuming initial distribution $\Pr(X = 1) = 0.5$, $p_d = 0.85$ and $p_f = 0.15$ we compute the following table $I[nz0, nz1, q]$used in our experiments as shown in Table 3.1.

Observe that when nz0=nz1, the mutual information for $q = 1$ is a con-

Table 3.1: Subset of table of mutual information of q future sensor measurements, indexed by previous number of sensor measurements equal to zero (nz0) and number of sensor measurements equal to one (nz1). Note that in the actual algorithm, the look-up table is computed for values nz0,nz1 ≤ 20 and q ≤ 10.

nz0	0	1	2	0	1	2	0	1	2
nz1	0	0	0	1	1	1	2	2	2
info (q=1)	0.390	0.209	0.050	0.209	0.390	0.209	0.050	0.209	0.390
info (q=2)	0.599	0.347	0.094	0.347	0.599	0.347	0.094	0.347	0.599
info (q=3)	0.737	0.432	0.125	0.432	0.737	0.432	0.125	0.432	0.737

stant 0.390. This is a limitation of the sensor model itself and is unlikely to occur for large nz0+nz1 when the static binary Bayes sensor model accurately models the environment and $p_d, p_f \neq 0.5$. Since X is Bernoulli there is at most 1 bit of information. The table is also symmetric when swapping nz0 and nz1 because $p_d = 1 - p_f$. Note that since mutual information is submodular, $I[nz0, nz1, 2] < 2 \cdot I[nz0, nz1, 1]$

For this work, we assume that the quadrotor has a downward facing camera with a circular field of view and can make observations in multiple cells (Fig. 3.9). We suppose that there exists image processing software that will identify the positive positions of detected targets in the image frame at a near continuous rate (overlap of FOVs of adjacent images processed is significant), while empty positions of the frame are considered negative detections. We assume that the probability of detection p_d and false positive rate p_f are known. These locations are then mapped from the image frame to the evidence grid. To avoid multiple correlated sensor measurements we only record a single observation for a given cell during a particular action

(transition between regions or search a region). This implies that targets in cells within a given distance from the quadrotor can be detected.

For realistic flight scenarios factoring actual hardware capabilities, motion blur on a global shutter camera occurs when camera/environment motion causes the displacements of image features during the image exposure to be greater than one pixel. Both rotational and translational velocities of the quadrotor contribute to motion blur, placing constraints on both the altitude and forward speed of the vehicle to prevent motion blur. To compare, we will consider the IDS UI-3251LE camera with optics that have a 90deg FOV (7mm focal length lens), 1600(H)x1200(V) pixels, flying at 5-20m altitude, and a 1 ms exposure time (suitable for brightly lit indoor environments) and outdoor environments,

Define $v_{g,max}$ to be the maximum speed of the ground in the image plane where $v_{g,max} = |v_{max}| + r_z|\omega_{max}|$ assuming nominal altitude above the ground r_z, and $\omega_{max} = 0.5\frac{\text{rad}}{\text{s}}$ is the maximum attitude angle change expected of the camera during maneuver execution or disturbance rejection.

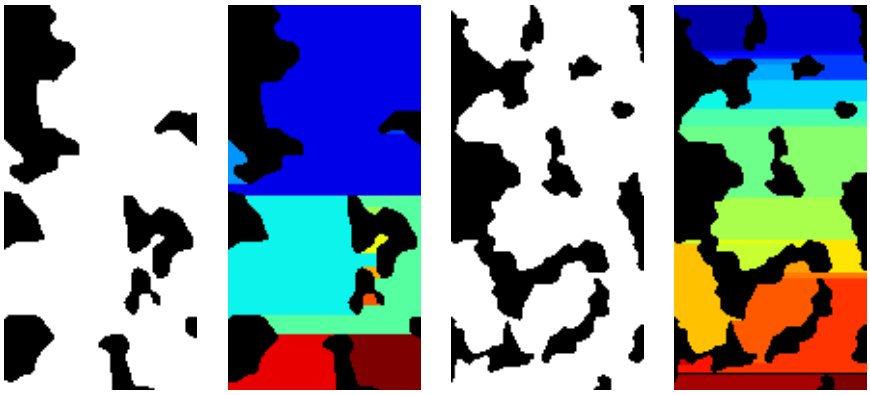
We also note that $FOV = \frac{r_z d_{sen}}{d_{foc}}$ where d_{foc} is the lens focal length and d_{sen} is the size of the image sensor along the direction of motion [175].

Define the displacement δp of the ground plane due to agent motion within the exposure time t_{exp} in pixels (Eq. 3.17):

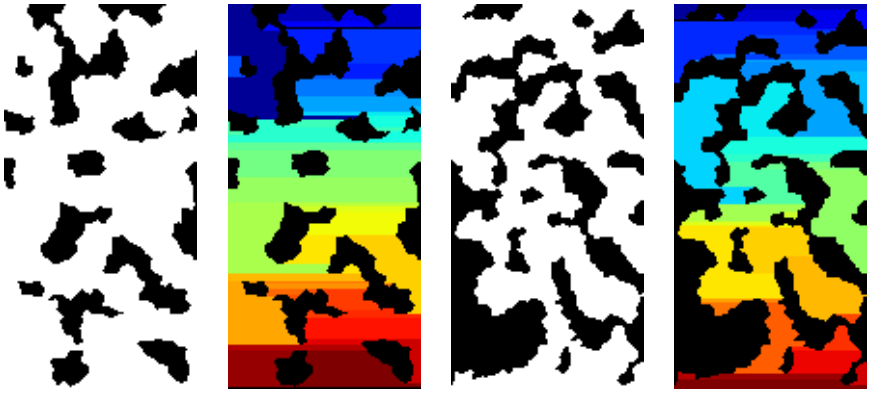
$$\delta p = v_{g,max} t_{exp} \frac{N_p}{FOV} \tag{3.17}$$

with N_p as the number of pixels along the direction of motion and FOV is the field of view in meters along the direction of motion. $\delta p \geq 1$ indicates motion blur. We note that an increase in altitude r_z reduces motion blur by increasing pixel size on the ground but rotational disturbances will dominate for high altitudes. A gimballed camera can reduce rotational disturbances at increased payload costs. This results in the permissible flight envelope by $\upsilon_{max} \in [0, 10], r_z \in [5, 20], \delta p < 1$. For example, setting $(\upsilon_{max}, r_z, t_{exp}) = (10, 10, 2 \cdot 10^{-3})$ results in a $\delta p = 1.8$ causing motion blur while $(\upsilon_{max}, r_z, t_{exp}) = (20, 5, 2 \cdot 10^{-3})$ results in $\delta p = 0.9$ not causing motion blur.

We define cell pitch $r = 2.2$m (not to be confused with camera pixel pitch) mapping cell indices to Euclidean coordinates in meters. We also assume that the camera has a circular 90deg FOV such that the sensing radius is 8.8m. The trajectory footprint of every action in the motion model is created by sweeping the sensor footprint along the trajectory planned by the Boustrophedon coverage planner. Therefore in practice the shape of the field of view doesn't matter as the sensor acts as a sweep sensor with the given sweep radius.



(a) low freq. env. (b) low freq. segmented (c) med freq. env. (d) low freq. segmented



(e) high freq. env. (f) high freq. segmented (g) vhf env. (h) vhf segmented

Figure 3.8: Example environments generated at various frequencies. Black denotes obstacles, and white denotes free space. The white space is divided up into plowable regions by the Boustrophedon algorithm. Each color corresponds to a region.

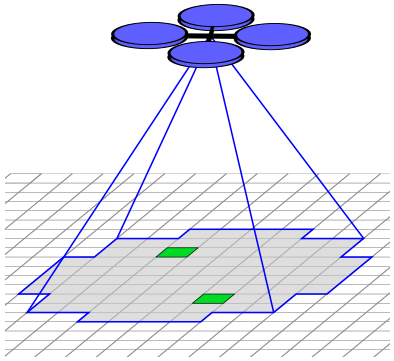


Figure 3.9: Sensor model for downward facing camera. Green/dark squares indicate cells containing detected targets, while grey squares indicate cells within the field of view with no target detections. Targets not shown.

3.6 Experimental Results

For each algorithm we normalize its performance with respect to the iterative greedy heuristic which always overestimates the reward. In other words, one computes the ratio of information each algorithm gathers in an environment by dividing the amount of mutual information (in bits) the algorithm gathers by the heuristic reward (in bits). Since the heuristic is admissible this ratio is $\leq 100\%$. This also normalizes for the variations in how much total information exists in a given environment. We ran 3 experiments; the first two were to understand various aspects about ϵ-admissible B&B.

1. Experiment 1 (rect.): Sweep α, η on the 12 region environment to determine best parameters, and illustrate that the choice of priority heuristic and that ϵ-admissible bounding improve the performance of the planner.

2. Experiment 2 (rect.): Sweep the size of the evidence grid to show (in-)sensitivity to size of evidence grid

3. Experiment 3 (rect.): For set values of α, η, benchmark branch and bound on 12, 24, 50 region environments to greedy approach, DFS and Zelinsky's algorithm in uniform and nonuniform environments. This also shows how the complexity of the algorithm increases for more complex environments.

4. Experiment 4 (Simplex): vary $\alpha \in \{0.2, 0.7, 0.8, 0.9\}$, and $\eta \in \{0\%, 0.5\%,$

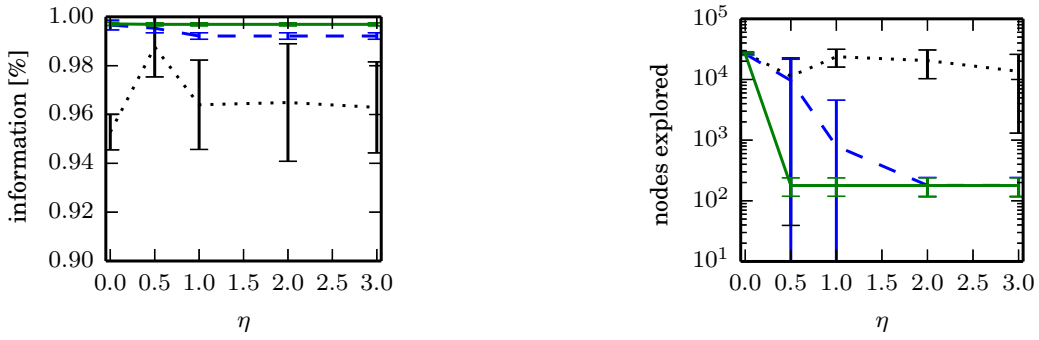
$1.0\%, 2.0\%, 3.0\%\}$ in low frequency environments for ϵ-admissible B&B to identify algorithm performance for different heuristic parameters.

5. Experiment 5 (Simplex): setting $(\alpha, \eta) = (0.8, 0.5)$, vary size of evidence grid in low frequency environments and observe how ϵ-admissible B&B scales with the evidence grid size.

6. Experiment 6 (Simplex): benchmark the four algorithms in Simplex environments, setting $(\alpha, \eta) = (0.8, 0.5)$ for ϵ-admissible B&B.

For uniform environments, all cells are initialized with the most uninformative prior of $P(X = 1) = \theta_0 = 0.5$. By inspecting the mutual information look up table, we observe that cells that have two or more prior sensor measurements have low enough information content that a nonuniform covering will gather more information than a uniform covering. When generating nonuniform environments, we want the geometry of low entropy regions to not correspond with the geometry of the obstacles. Therefore the nonuniform environments generate a distribution of low entropy regions using Simplex noise thresholded at 50% (i.e. 50% of the environment is low entropy, while the remaining environment is set to high entropy).

For the environments (low,med,high,vhf), define the low entropy regions with the following frequency parameters (vlf,vlf,low,med).



(a) information gathered

(b) nodes expanded

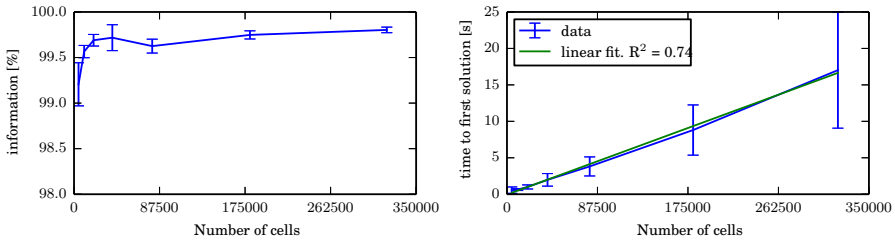
Figure 3.10: Results for Experiment 1. $(\alpha, \eta) = (0.9, 0.5)$ reduces the number of nodes expanded while not affecting solution quality. Green/solid is $\alpha = 0.9$, blue/dashed is $\alpha = 0.7$ and black/dotted is $\alpha = 0.5$.

3.6.1 Experiment 1

We set $\alpha \in \{0.5, 0.7, 0.9\}$ and $\eta \in \{0.0, 0.5, 1.0, 2.0, 3.0\}$ in the 12 region environments. For these trials, the planner continues to plan until the priority queue is empty. Fig. 3.10(a) shows what percentage of information the planner is able to collect for given parameter set (α, η). We see that for $\alpha \geq 0.7$ collected information is insensitive to choice of η (this stems from the fact that for higher values of α, the first discovered solution is usually the best). In Fig. 3.10(b) we observe how tweaking the heuristic used for ϵ-admissible bounding dramatically reduces the number of nodes expanded. For example setting $(\alpha = 0.9, \eta = 0.5)$ only explores 0.6% of the number of nodes explored when compared with setting $(\alpha = 0.9, \eta = 0.0)$.

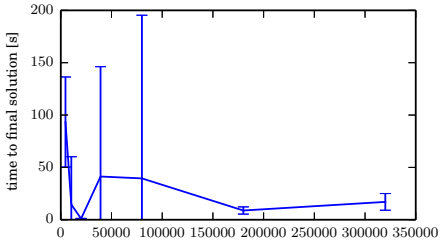
3.6.2 Experiment 2

The results of Experiment 2 show that the time to the first solution is linear with the number of cells in the evidence grid (Fig. 3.14(d)) without affecting solution quality (Fig. 3.11(a)). This is due to the hierarchical nature of the formulation instead of planning directly in the evidence grid (which would be exponential in the number of grid cells). Time to algorithm termination (Fig. 3.11(c)) is not linear with the number of cells in part due to the effects of introducing caching in the heuristic. Further, the quality of the heuristic improves in larger environments since traversal times become insignificant with larger regions with longer search times.



(a) information gathered

(b) time to first solution



(c) time to terminate algorithm

Figure 3.11: Results for Experiment 2. Runtime is linear with cell count and does not affect total information gathered by the proposed algorithm.

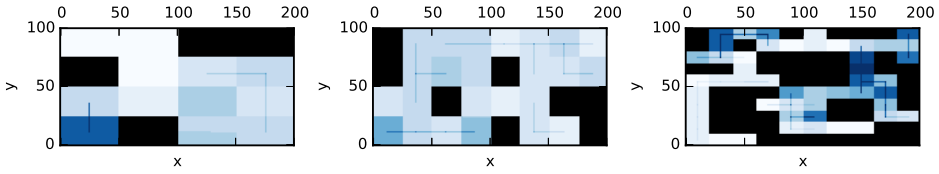
3.6.3 Experiment 3

Fixing $\alpha = 0.8$ and $\eta = 0.5\%$ unless otherwise stated, we will benchmark the proposed branch and bound algorithm to other algorithms in the 12, 24, and 50 region environments. Branch and bound will run until the first solution is found, or until 10,000 or more iterations have completed. We would like to note that Zelinsky's algorithm was implemented in the programming language Julia, which has speeds comparable to C and is much faster than Python. Since we are pushing the proposed algorithm to the limits, it may be possible that branch and bound will not be able to improve upon the solutions discovered by heuristic solvers such as Zelinsky's algorithm or the greedy algorithm within the allotted time. We consider a trial a success when branch and bound finds a solution in the allotted time. We anticipate that future improvements in heuristic design will dramatically improve algorithm performance for the more difficult environments.

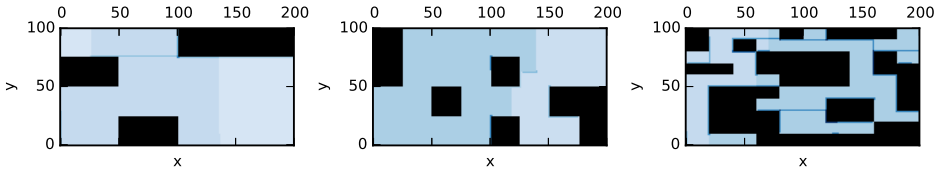
Tables 3.2 and 3.3 summarize the statistics for the time to first solution in seconds (time) and percentage of total available information gathered (info) displaying averages, standard deviations, and success rates.

3.6.3.1 Uniform Environment

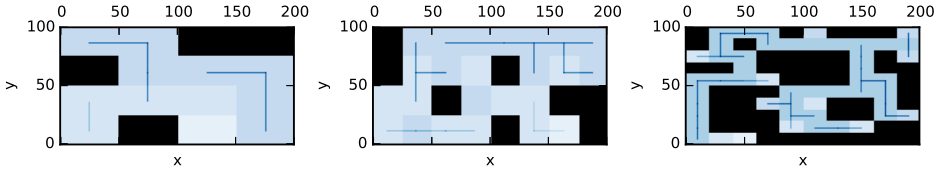
Figure 3.12 shows example footprints that closely resemble *median* performance of each of the algorithms for the uniform environment. The footprints are the same size as the evidence grid, where white cells are unexplored by



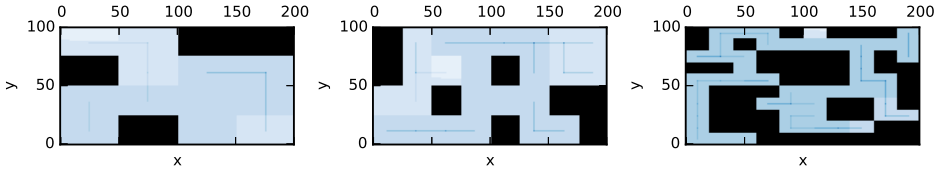
(a) greedy algorithm



(b) Zelinsky's algorithm



(c) DFS algorithm



(d) B&B algorithm

Figure 3.12: Coverage plans in rectangular environments with uniform information. Different algorithms (rows) with respect to region count (columns). Even coverage is desired. The proposed branch and bound method, DFS and Zelinsky's algorithm perform well.

the trajectory, blue cells as being visited regions (darker cells have been visited more often), and black regions denote obstacles. Table 3.2 summarizes the statistics for the uniform, static environments.

Table 3.2: Uniform rectangular environments, mean of 40 trials

Regions	Branch and Bound			
		info	time	effort
12		99.2% ± 0.1%	1.15 ± 0.4	1.30 ± 0.003
24		99.2% ± 0.1%	6.5 ± 6.9	1.31 ± 0.003
50	(83% succ.)	99.2% ± 0.1%	128 ± 135	0.98 ± 0.003

Regions	DFS		
	info	time	effort
12	97.5% ± 0.6%	0.17 ± 0.01	1.34 ± 0.01
24	97.0% ± 0.2%	0.21 ± 0.01	1.35 ± 0.006
50	96.5% ± 0.2%	0.37 ± 0.01	1.04 ± 0.009

Regions	Greedy		
	info	time	effort
12	72% ± 11%	0.2 ± 0.03	1.73 ± 0.15
24	84% ± 9%	0.3 ± 0.02	1.44 ± 0.04
50	52% ± 19%	0.94 ± 0.18	1.85 ± 0.33

Regions	Zelinsky		
	info	time	effort
12	99.7% ± 0.08%	$< 0.1^*$	1.30 ± 0.002
24	99.5% ± 0.1%	$< 0.1^*$	1.30 ± 0.004
50	78% ± 0.1%	$< 0.1^*$	1.00 ± 0.002

3.6.3.2 Nonuniform environment

We observe that branch and bound is able to handle the nonuniform environments as equally well as the uniform environments, while Zelinsky's algorithm suffers. Results are summarized in Table 3.3.

Table 3.3: Nonuniform rectangular environments, mean of 40 trials.

Regions	Branch and Bound			
		info	time	effort
12		99.3% ± 0.2%	1.0 ± 0.5	0.90 ± 0.006
24		99.3% ± 0.1%	9.6 ± 16	0.90 ± 0.002
50	(80% succ.)	99.3% ± 0.1%	79 ± 80	0.73 ± 0.004

Regions	DFS		
	info	time	effort
12	96.9% ± 1%	0.16 ± 0.01	0.93 ± 0.02
24	96.7% ± 0.8%	0.23 ± 0.01	0.93 ± 0.01
50	95.5% ± 0.6%	0.37 ± 0.01	0.82 ± 0.01

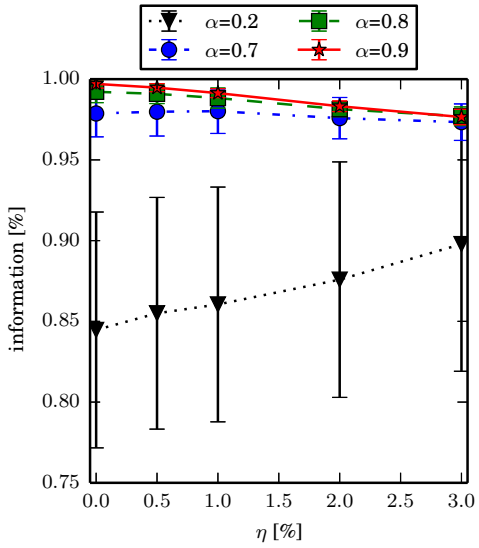
Regions	Greedy		
	info	time	effort
12	70% ± 15%	0.19 ± 0.02	1.31 ± 0.19
24	84% ± 9.4%	0.32 ± 0.03	1.11 ± 0.12
50	72% ± 13%	0.72 ± 0.13	1.59 ± 0.3

Regions	Zelinsky		
	info	time	effort
12	90% ± 1.6%	$< 0.1^*$	1.07 ± 0.03
24	90% ± 1.2%	$< 0.1^*$	1.07 ± 0.02
50	93% ± 0.4%	$< 0.1^*$	0.9 ± 0.006

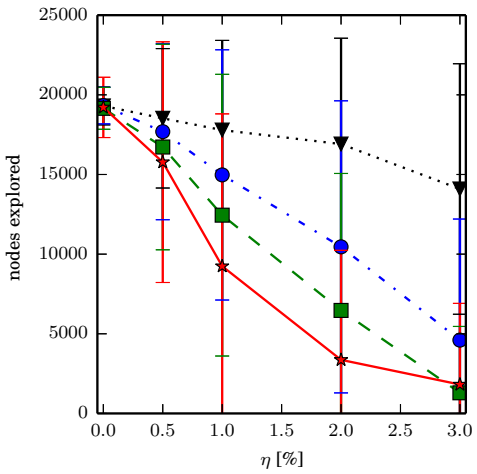
3.6.4 Experiment 4

We tested the branch and bound algorithm in the low environment, terminating if the priority queue is empty or after 6,000 iterations have occurred. We set $\alpha \in \{0.2, 0.7, 0.8, 0.9\}$, and $\eta \in \{0\%, 0.5\%, 1.0\%, 2.0\%, 3.0\%\}$ and observe total information gathered, and number of nodes explored. Fewer nodes explored implies more nodes in the search space are pruned. As both α and η increase, the number of nodes explored decreases due to better prioritization or pruning. Results are summarized in Fig. 3.13. In terms of the reduction in nodes expanded, for $(\alpha, \eta) = (0.9, 1.0\%)$ on average only expands 48% of the nodes that are expanded when $(\alpha, \eta) = (0.9, 0.0\%)$ suggesting that ϵ-admissible branch and bound can speed up the search by about a factor of two while finding solutions that are on average within 0.6% of the optimal solution.

In determining the parameters for future experiments, $(\alpha, \eta) = (0.8, 0.5)$ was chosen for the following reasons . Setting $\eta = 0.5\%$ ensures that ϵ-admissible B&B finds solutions within 0.2% of the optimal solution. While $\alpha = 0.9$ expands the fewest nodes in the lowf environments, in larger environments, setting $\alpha = 0.8$ instead of $\alpha = 0.9$ decreases the time to finding the first solution which significantly increases the success rate of the algorithm and reduces memory usage (Exp. 3).



(a) Information gathered for ϵ-admissible B&B in low freq. environments. This decreases with increasing η and decreasing α.



(b) Nodes explored in low freq. environments for ϵ-admissible B&B.

Figure 3.13: Results for Experiment 4

3.6.5 Experiment 5

For Experiment 5 we varied the size of the evidence grid to consist of $N_{cells} \in \{4608, 10368, 20000, 39200, 80000, 180000, 320000\}$ where each environment has a 2:1 aspect ratio. In addition, the resolution of the cells changes such that the area of the environment in meters is the same in all trials. Also, the sensor footprint is the same in meters in all environments. For the Experiment 5 Simplex environments, the following parameters are: $(f_1, f_2) = (0.0075, 0.1)$ and $(w_1, w_2) = (1, 0.0625)$.

For each environment size we generated 10 environments with 4 starting locations for a total of 40 trials. Environments are generated using Simplex

noise for environments with 320000 cells (or 800x400 cells) and are down-sampled to smaller environment sizes.

After the environment is re-scaled, Boustrophedon decomposition generates the discrete graph before running the proposed branch and bound algorithm. The algorithm also scales parameters for Boustrophedon decomposition such as the minimum area of an obstacle before it is removed. The environments don't scale perfectly however, and the number of regions per environment can still vary (Fig. 3.14(a)). Note that the size of the environment in cell count varies by a factor of 70, while the number of regions varies by a factor of 1.3, implying that the increase is insensitive to increase in cell count. Also note that T_{max}, defined by the total time to cover each region twice, is insensitive to the number of cells and its standard deviation over all cell sizes is about 750 seconds with a mean of 9980, varying by about 7.5% (Fig. 3.14(b)). Branch and bound runs for 6000 iterations or until Q_{open} is empty. The time to the final solution is approximately linear (Fig. 3.14(e)) when the number of cells expanded remains constant Fig. 3.14(c).

One thing of interest is that the quality of the solution degrades as the size of the environment increases (Fig. 3.14(f) and Fig. 3.14(g). This is not a limitation of the algorithm itself; Fig. 3.15 shows that the quality of solutions found using DFS on the same environments degrades similarly with evidence grid size. One possibility that explains this is that the environment is not scaling perfectly. When the sensor footprint is discretized in a coarser grid (when N_{cells} is smaller resulting in larger grid resolutions), the footprint tends

to be more square. This can introduce artifacts in the total area swept by a moving sensor by a factor at most $\sqrt{2}$ when moving diagonally, dramatically increasing how much information per unit of time is gathered when traversing between regions. Traversals in environments with fewer cells would therefore get a slight performance boost.

Note that in certain environments, many of the resulting statistics are heavily skewed and non-Gaussian so the error bars (Fig. 3.14(d)) which represent one standard deviation dip below 0. None of the trials violate constraints such as $t < 0$.

The results of Experiment 5 demonstrate that ϵ-admissible B&B scales reasonably well with increasing environment size due to the hierarchical nature of the algorithm. Experimental results demonstrate that the ϵ-admissible heuristic is able to significantly speed up branch and bound with bounded loss in optimality.

3.6.6 Experiment 6

For Experiment 6 we benchmarked the proposed branch and bound algorithm with $(\alpha, \eta) = (0.8, 0.5\%)$ against DFS and the greedy algorithm. The algorithm terminates after either the priority queue is empty or $N = 6,000$ iterations have completed. All algorithms are benchmarked with both uniform and nonuniform prior distributions.

To get an understanding of how the various algorithms perform in different environments, we illustrate trajectory footprints representing median al-

gorithm performance from ϵ-admissible B&B (Fig. 3.16), greedy (Fig. 3.17), DFS (Fig. 3.18) and DF-B&B (Fig. 3.19) in uniform environments. Similarly, Figs. 3.21, 3.22, 3.23 and 3.24 show the median performance of their respective algorithms in nonuniform environments. Fig. 3.20 shows the entropy in example environments with information non-uniformly distributed in the environment.

The trajectory footprints in Figs. 3.16-3.19 and Figs. 3.21-3.24 are an array representation of the best trajectory discovered by the mentioned algorithm, $\Phi_{\mathcal{T}}[x, y]$. Areas not visited are white, obstacles are black, and blue regions indicate how many times the areas have been visited (see the legend in Fig. 3.16(e)). Multipass coverage plans that are more uniform should be more uniform in color. Notice that all cells in a given region are not the same color; dots and lines shown within the regions show the information gathered between region traversals as these are visited additional times along commonly traveled paths between regions. Other lines are due to sensor footprints overlapping between adjacent regions.

One can observe that branch and bound covers the environment most uniformly (resulting in the highest quality solution) while the greedy solver can get stuck or miss regions. DFS's performance falls in between ϵ-admissible B&B and the greedy heuristic solver while DF-B&B has the worst performance. Results are summarized on the uniform environments in Table 3.4 and on the nonuniform environments in Table 3.5. N_{sol} denotes the number of solution improvements found during the search. The described effort allo-

cation model computes the expected time to search a cell in number of time constants (e.g. with $\tau = 1$), denoted as "effort" on Table 3.4 for the uniform environment using detection threshold $P_{neg} = 1\%$. Fig. 3.25 and Fig. 3.26 summarize key results in Tables 3.4 and 3.5 respectively.

Comparing the two algorithms that gather the most information on average in Experiment 6, ϵ-admissible B&B gathers 2.3% more information on average than DFS search taking 1-3 orders of magnitude longer to compute. To get a better sense of the real world performance improvement, the solutions generated by ϵ-admissible B&B searches uniform environment at a rate that is 2.8% faster than the DFS coverage planner (where the search rate is inversely proportional to time to search a given cell), while ϵ-admissible B&B searches nonuniform environments at a rate that is 0.8% faster than DFS. We think that this decreased performance improvement in the nonuniform environments is a limitation of the Boustrophedon decomposition algorithm and/or the coverage planner. Generating region coverage plans that factor in the information content (consider an extension of the weighted Voronoi Diagram that accounts for obstacles) would alleviate this and allow ϵ-admissible B&B to better reason about nonuniform distributions of information.

3.7 Discussion

Since branch and bound reasons over all possible solutions, it discovers the highest reward path but takes the longest to compute. The ϵ-admissible

Table 3.4: Experiment 6 results for uniform Simplex environments, mean of 80 trials

env.	succ. rate	ϵ-admissible Branch and Bound					
		1st info	1st time	N_{sol}	final info	final time	effort
low	100%	92.8% ± 5.3%	5.8 ± 9	10 ± 4.1	99.1% ± 0.6%	316 ± 169	1.4 ± 0.033
med	98.8%	95.9% ± 0.2%	27 ± 47	9.3 ± 4.5	98.5% ± 0.5%	648 ± 175	1.36 ± 0.030
high	93.8%	96.8% ± 1.1%	76 ± 77	7.4 ± 4.2	98.2% ± 0.5%	961 ± 308	1.31 ± 0.028
vhf	98.8%	96.6% ± 0.9%	129 ± 143	8.9 ± 4.7	98.0% ± 0.9%	1185 ± 273	1.27 ± 0.030

env.	Greedy			DFS coverage planner		
	info	time	effort	info	time	effort
low	85% ± 7.8%	0.3 ± 0.1	1.61 ± 0.01	97% ± 3%	0.2 ± 0.02	1.48 ± 0.064
med	81% ± 11%	0.57± 0.12	1.61 ± 0.15	97% ± 1.0%	0.3 ± 0.03	1.40 ± 0.033
high	82% ± 11%	0.78 ± 0.26	1.56 ± 0.18	97% ± 0.7%	0.42 ± 0.04	1.35 ± 0.029
vhf	84% ± 10%	0.87 ± 0.15	1.47 ± 0.16	96% ± 1%	0.5 ± 0.04	1.30 ± 0.18

env.	Depth First Branch and Bound					
	1st info	1st time	N_{sol}	final info	final time	effort
low	71.2% ± 10%	8.5 ± 4.7	42 ± 17	78.1% ± 8.8%	235 ± 85	1.72 ± 0.11
med	66.3% ± 11.4%	31 ± 12	38 ± 16	71.2% ± 10.1%	466 ± 107	1.75 ± 0.14
high	61.9% ± 11.5%	69 ± 21	34 ± 15	65.5% ± 10.6%	712 ± 160	1.80 ± 0.15
vhf	61.6% ± 11.8%	107 ± 31	30 ± 15	64.6% ± 11.6%	862 ± 216	1.77 ± 0.17

Table 3.5: Experiment results for nonuniform Simplex environments, mean of 80 trials

env.	succ. rate	ϵ-admissible Branch and Bound					
		1st info	1st time	N_{sol}	final info	final time	effort
low	100%	93.2% ± 4.3%	6.8 ± 20	11 ± 5.8	98.6% ± 0.1%	282 ± 137	1.21 ± 0.05
med	98.8%	97% ± 0.4%	31 ± 51	9.5 ± 4.3	98.7% ± 0.4%	596 ± 271	1.27 ± 0.04
high	100%	96.6% ± 1.5%	111 ± 141	7.5 ± 4.1	98.1% ± 0.6%	760 ± 264	1.23 ± 0.02
vhf	95%	96.3% ± 1.2%	151 ± 199	8.7 ± 5.1	97.8% ± 0.8%	965 ± 288	1.20 ± 0.03

env.	Greedy			DFS coverage planner		
	info	time	effort	info	time	effort
low	83% ± 11%	0.3 ± 0.01	1.34 ± 0.10	95% ± 3.3%	0.19 ± 0.03	1.25 ± 0.05
med	79% ± 16%	0.51 ± 0.13	1.41 ± 0.09	96% ± 1.3%	0.31 ± 0.03	1.26 ± 0.03
high	80% ± 10%	0.69 ± 0.01	1.35 ± 0.08	96% ± 1.0%	0.42 ± 0.04	1.24 ± 0.02
vhf	81% ± 12%	0.83 ± 0.16	1.31 ± 0.10	95% ± 1.3%	0.5 ± 0.04	1.20 ± 0.02

env.	Depth First Branch and Bound					
	1st info	1st time	N_{sol}	final info	final time	effort
low	70.2% ± 13.9%	7.3 ± 4.4	36 ± 16	77.3% ± 12.6%	222 ± 78	1.4 ± 0.1
med	62.6% ± 13.0%	29 ± 12	38 ± 16	68.2% ± 11.6%	448 ± 124	1.43 ± 0.09
high	60.9% ± 11.2%	68 ± 21	30 ± 14	64.6% ± 10.6%	675 ± 169	1.45 ± 0.08
vhf	61.3% ± 11.4%	101 ± 31	28 ± 10	64.7% ± 10.7%	790 ± 210	1.42 ± 0.09

heuristic shows that having a tighter bound dramatically reduces the number of nodes expanded later in the search. One limitation to ϵ-admissible B&B is that the heavy reliance on the iterative greedy heuristic and all the heuristic's limitations (see Sec. 3.4.1) can cause the planner to continually expand too many candidate solutions at a fixed depth preventing the planner from finding a solution within the allotted time and memory constraints. Memory efficient search techniques such as beam-stack search [176] could alleviate such bottlenecks. On the other hand, we observe that ϵ-admissible B&B is an anytime algorithm. In the vhf uniform environments for example, around 90% of the time the first solution is discovered within 4.5 minutes (in contrast with about 90% of the trials terminating around 24 minutes).

Zelinsky's algorithm is effective in uniform environments as expected but suffers in nonuniform environments. The heuristic solver DFS finds the second highest quality solutions in the shortest amount of time offering an effective trade-off in solution quality vs. speed to compute. We note that the quality of the DFS solutions degrades in environments with many more (smaller) regions due to the fact that more time is spent traversing between regions vs. searching the individual regions. The poor solution quality of DF-B&B despite the long time to compute the solution highlights the importance of generating effective heuristics to guide branch and bound to find high quality solutions. This is a finding that is not always recognized in the literature (usage of DF-B&B in place of heuristic guided B&B has occurred [10]).

However, for *very large* problem instances that we did not investigate in

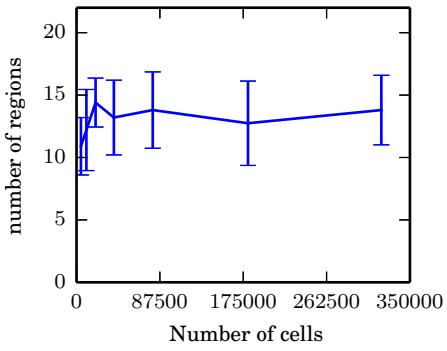
this chapter (hundreds to thousands of regions, or tens of millions of cells), the findings of this study must be reinvestigated. Using branch and bound techniques may become intractable for very large problems, and heuristic techniques may offer the only feasible solution that can be computed in realistic time, with significant drops in solution quality.

3.8 Summary

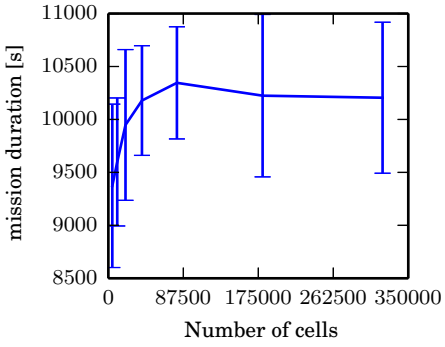
In this chapter we identify the task of generating multipass coverage plans for target search, as agents searching for targets must assess the trade-offs of searching a given region multiple times. This relatively unexplored area of research requires generating plans that visit most if not all of the workspace multiple times, where even seemingly small problems require long action sequences to solve. Given the difficulty of the problem, we created the algorithm ϵ-admissible branch and bound (ϵ-admissible B&B) [12] which uses heuristics to improve the quality of the solution found in the given time. We benchmarked four different algorithms in Simplex environments assessing various performance metrics such as how long it takes to compute the solution and the quality of the solution (information gathered minimizing uncertainty in bits and expected time for a relief crew to search a cell).

The two most promising algorithms are ϵ-admissible B&B and the Depth First Search (DFS) heuristic solver [47]. ϵ-admissible B&B finds the highest quality solutions (gathers 2.2% more information than DFS and generates

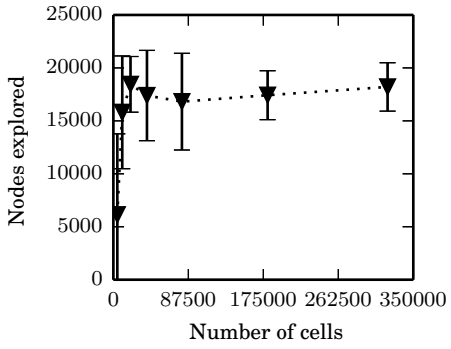
plans that search uniform environments 2.8% and nonuniform environments 0.8% faster than DFS) but takes the most time to compute. DFS on the other hand finds the second highest quality solutions but takes the shortest time to compute. Access to cheap supercomputing capabilities with the high value of a statistical life suggests that even small improvements in solution quality can be cost effective.



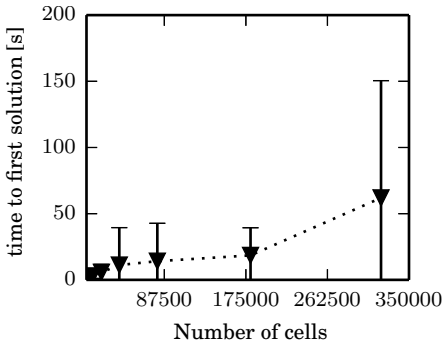
(a) Number of regions for given environment size



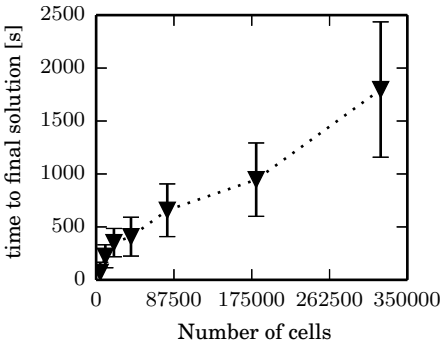
(b) mission duration



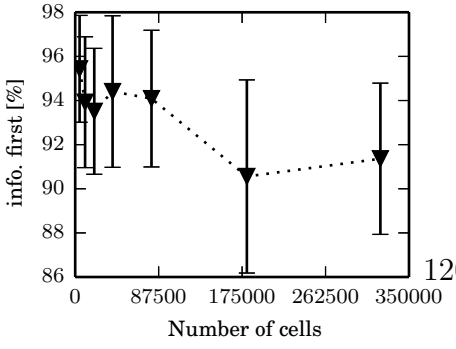
(c) nodes expanded



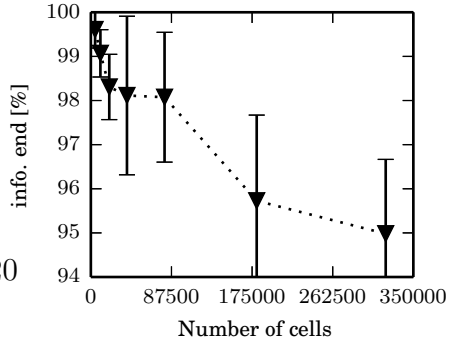
(d) Time to first solution



(e) Time to last solution



(f) Information gathered from first solution



(g) Information gathered from last solution

Figure 3.14: Results for Experiment 5.

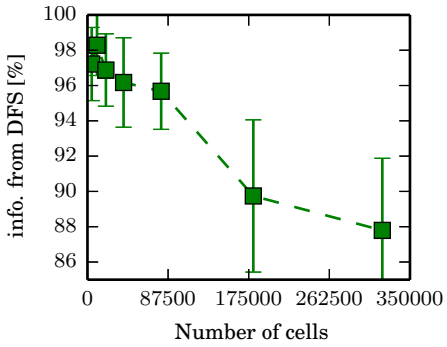
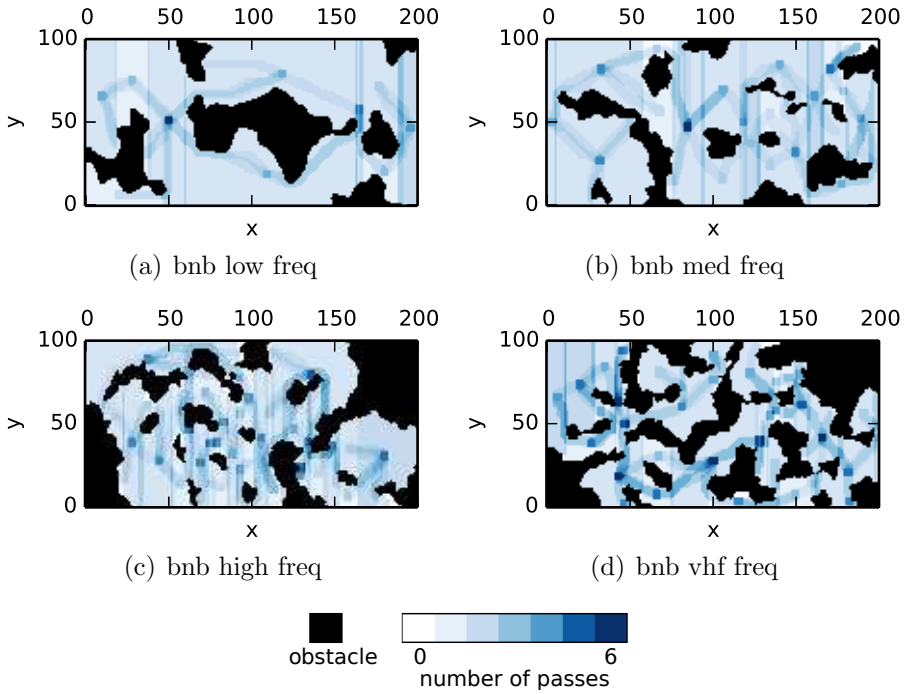


Figure 3.15: Information gathered from DFS. This is a benchmark to demonstrate that solution quality degrades with N_{cells} and is not dependent on the algorithm.



(a) bnb low freq

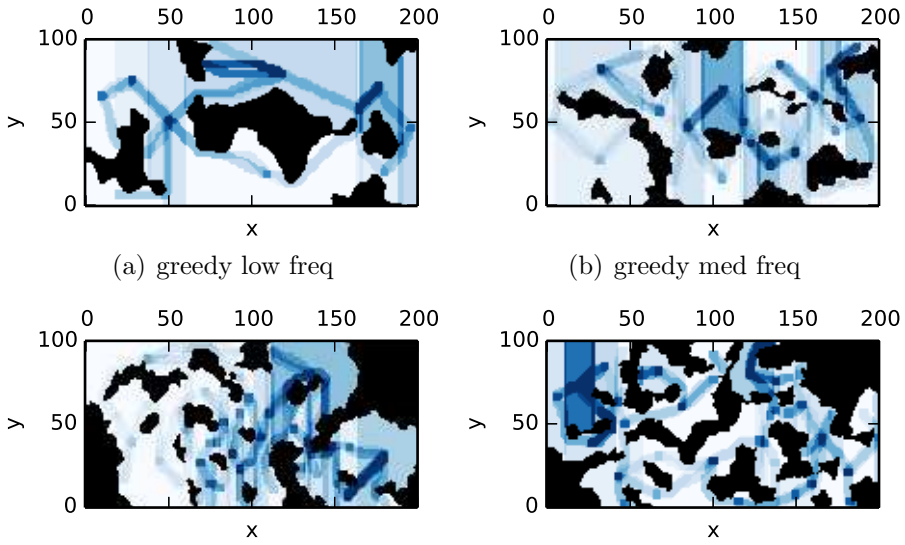
(b) bnb med freq

(c) bnb high freq

(d) bnb vhf freq

(e) footprint legend

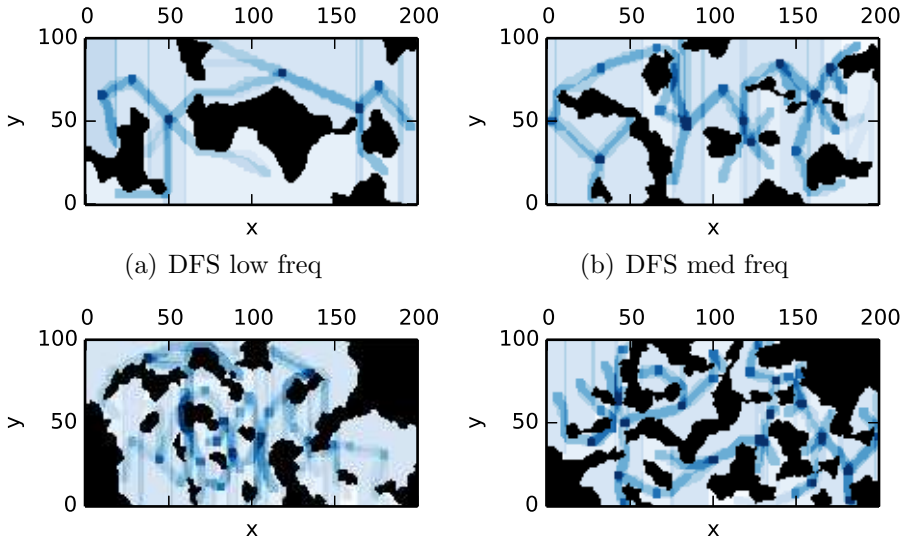
Figure 3.16: multipass coverage plans in environments with uniform information for the ϵ-admissible B&B algorithm



(a) greedy low freq (b) greedy med freq

(c) greedy high freq (d) greedy vhf freq

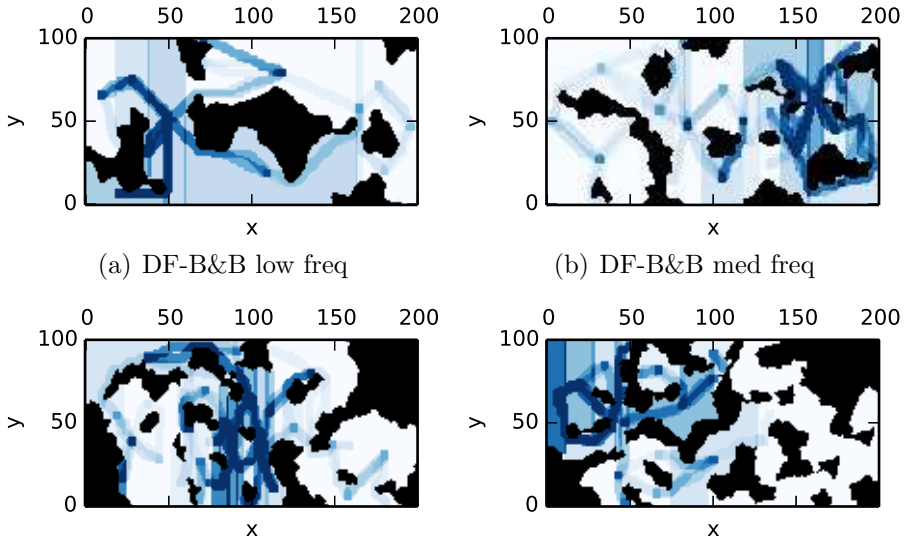
Figure 3.17: multipass coverage plans in environments with uniform information for the greedy heuristic algorithm



(a) DFS low freq (b) DFS med freq

(c) DFS high freq (d) DFS vhf freq

Figure 3.18: multipass coverage plans in environments with uniform information for the DFS coverage planner algorithm



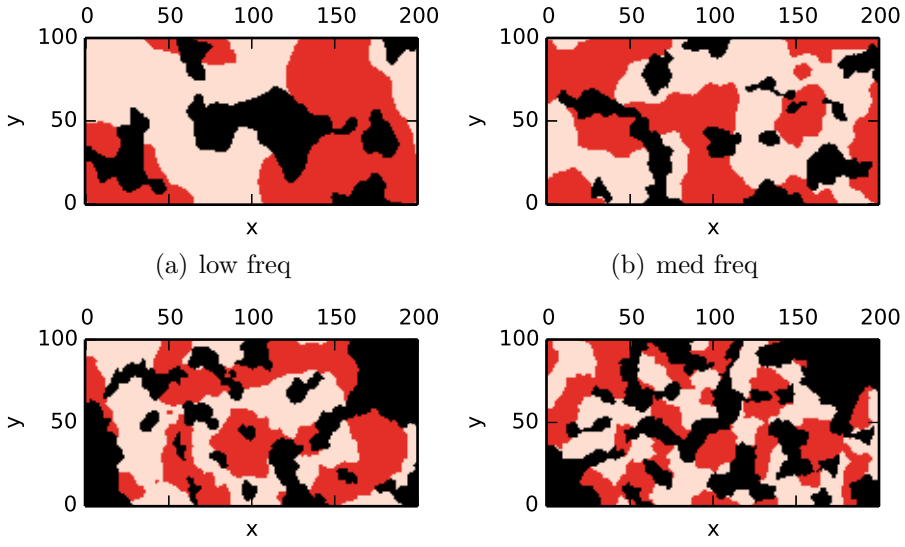
(a) DF-B&B low freq

(b) DF-B&B med freq

(c) DF-B&B high freq

(d) DF-B&B vhf freq

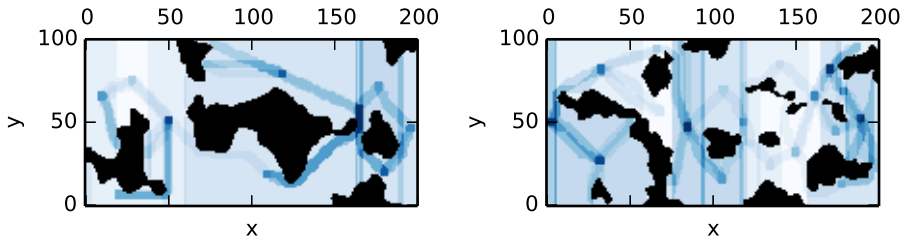
Figure 3.19: multipass coverage plans in environments with uniform information for the DF-B&B coverage planner algorithm



(a) low freq (b) med freq

(c) high freq (d) vhf freq

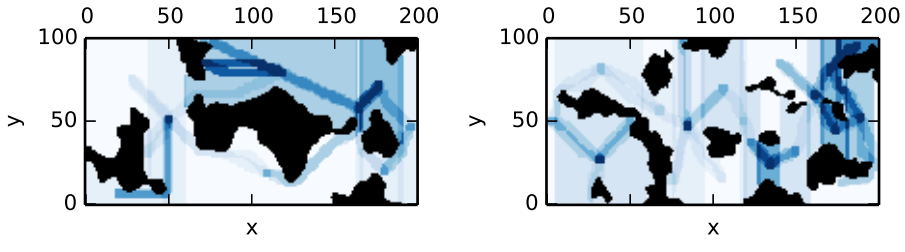
Figure 3.20: Example nonuniform environments showing obstacles (black), and the entropy of the cells before the search. The dark red indicates high entropy regions while the light red indicates low entropy regions.



(a) bnb low freq (b) bnb med freq

(c) bnb high freq (d) bnb vhf freq

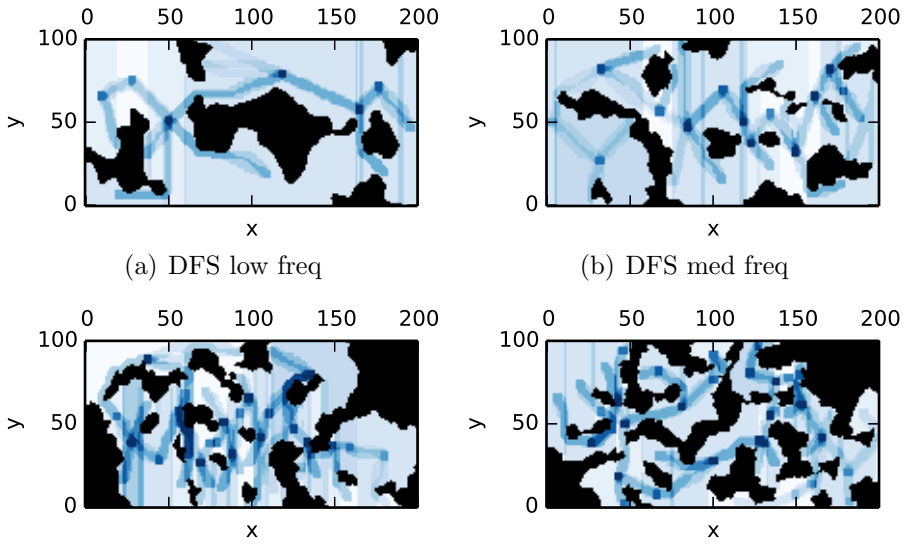
Figure 3.21: multipass coverage plans in environments with nonuniform information for the ϵ-admissible B&B algorithm



(a) greedy low freq (b) greedy med freq

(c) greedy high freq (d) greedy vhf freq

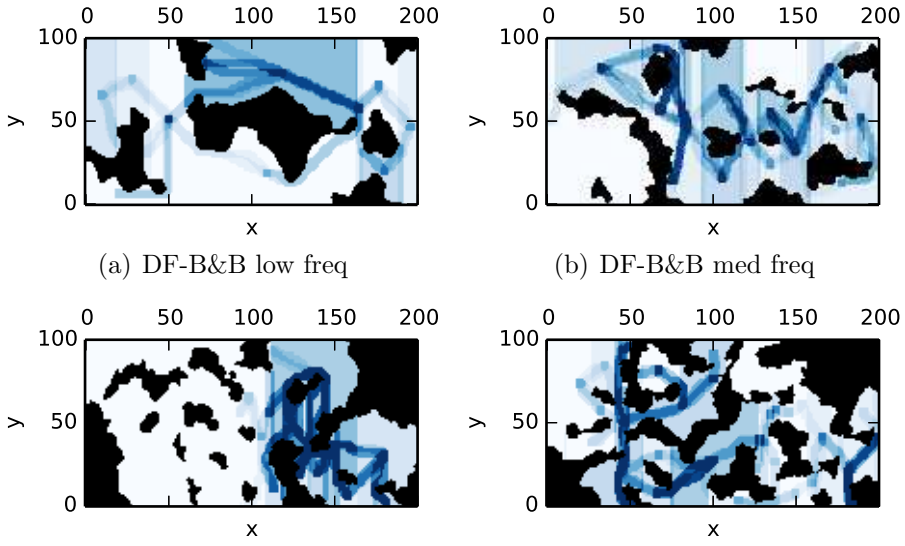
Figure 3.22: multipass coverage plans in environments with nonuniform information for the greedy heuristic algorithm



(a) DFS low freq (b) DFS med freq

(c) DFS high freq (d) DFS vhf freq

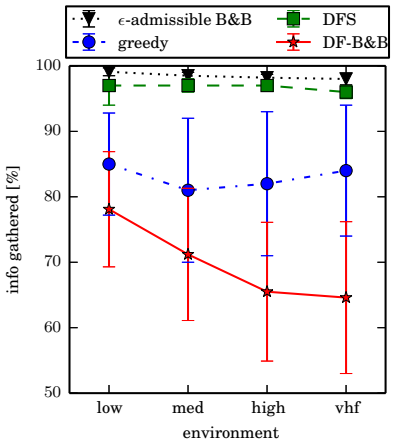
Figure 3.23: multipass coverage plans in environments with nonuniform information for the DFS coverage planner algorithm



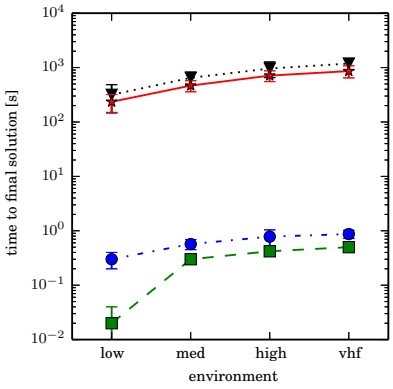
(a) DF-B&B low freq (b) DF-B&B med freq

(c) DF-B&B high freq (d) DF-B&B vhf freq

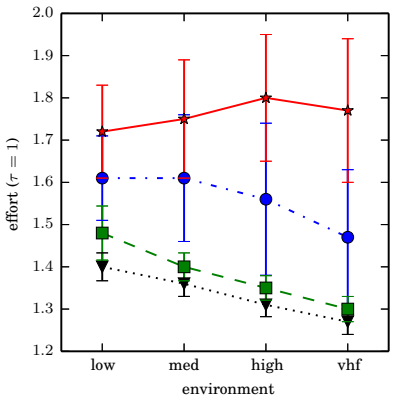
Figure 3.24: multipass coverage plans in environments with nonuniform information for the DF-B&B coverage planner algorithm



(a) info gathered in final solution

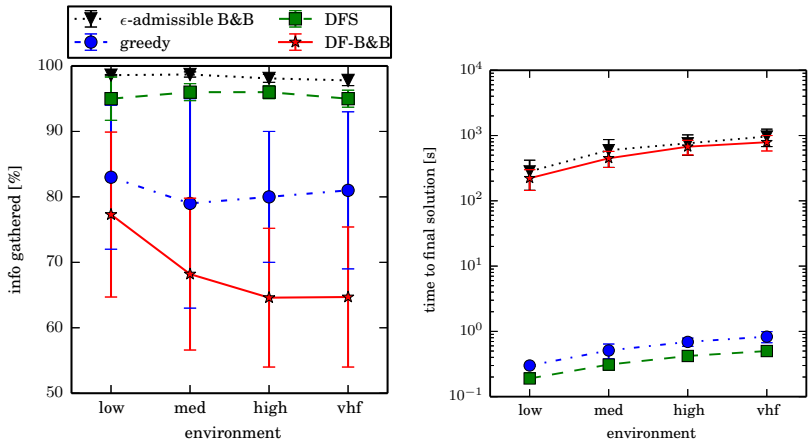


(b) time to final solution

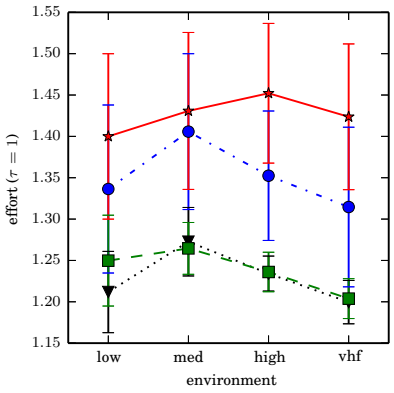


(c) expected search effort per cell

Figure 3.25: Results for Experiment 6 in uniform environment. See Table 3.4 for more detailed figures.



(a) info gathered in final solution (b) time to final solution



(c) expected search effort per cell

Figure 3.26: Results for Experiment 6 in nonuniform environment. See Table 3.5 for more detailed figures.

Chapter 4: Coordinating Underwater Vehicle Teams to Conduct Large-Scale Geospatial Tasks

Portions of this work are to be presented at the 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems [177].

4.1 Introduction

In order to maximize endurance while minimizing mission cost, consider use of underwater vehicles with weak propulsion that can exploit the ocean currents. Missions spanning hundreds of kilometers require coordination of many vehicles, yet many state-of-the-art approaches do not scale well to such large scenarios. This approach also requires being able to account for forecast uncertainty, which poses yet another unresolved challenge for mission planning. Finally, rewards of interest such as adaptive sampling or coverage planning are path dependent because overlapping sensor measurements reduce the total information gathered. The path dependence of the reward makes many well-known techniques such as dynamic programming intractable for the problem at hand.

To get a sense of the scale of problems of interest, Figure 4.1 demonstrates a hypothetical large scale fleet exercise in the Gulf of Mexico where UUVs are required to track algae blooms by tracking regions of a given temperature. While doing so agents must coordinate their forecasted 7 day trajectories to minimize overlap while exploiting the ocean currents. The region is 84°W to 93°W, and 21°N to 30°N, resulting in an environment that is 935x997km. Any agents that leave the region are considered lost. A total of 256 agents are deployed uniformly at random from 85°W to 92°W and 22°N to 29°N. Selecting trajectories for all agents that jointly maximizes reward gathered is a monumental task. Even limiting the decision space of each individual agent to 15 candidate trajectories results in a joint decision space containing $15^{256} \approx 10^{301}$ elements! Naturally, this decision space cannot be searched exhaustively. One option around this is to instead search a significantly smaller decision space, e.g. either that of sequentially coordinated trajectories, or that of a shared policy that does not distinguish between individual agents.

In this chapter we consider large teams of unmanned underwater vehicles (UUVs) conducting large-scale geospatial tasks such as information gathering or coverage planning. We investigate planning techniques that can evaluate path dependent rewards, account for the ocean forecast, and efficiently coordinate plans for many agents. Two formulations investigated either search the space of action sequences or the space of feedback policies to find dynamically feasible trajectories. We present what we believe to be the first

application of the Cross Entropy Method to coordinating large teams of 8-128 UUVs. We also develop a novel iterative greedy method that further refines the best discovered constant action sequences to improve other greedy techniques. The iterative greedy method gathers the most information on average, scales well to large problems and is the most cost effective means of deploying large teams of agents by gathering 3%-8% more reward than other techniques. The goals of this topic are:

- identify existing state-of-the-art algorithms relevant to the task of coordinating large teams of UUVs for geospatial tasks in the presence of ocean forecast data:
- Extend techniques for solving MDPs to path dependent rewards
- Propose novel greedy methods capable of efficiently coordinating large teams of agents.
- Benchmark the identified algorithms in the geospatial tasks of adaptive sampling based on temperature and coverage planning in three experiments to determine how the techniques scale to teams of 8-128 agents.

The layout of this chapter starts off with preliminaries in Section 4.2 before introducing the problem formulation in Section 4.3. Section 4.4 describes the two representations used by methods in this paper for restricting the search to the space of feasible trajectories. Algorithms that exist in the

prior art and are capable of solving the mentioned problem are covered in Section 4.5. The novel methods constituting the primary contribution of this paper are described in Section 4.6. Section 4.7 describes the experimental setup for the benchmark environment and Section 4.8 covers the results of the algorithms. The results are discussed in Section 4.9 and Section 4.10 concludes the chapter.

4.2 Preliminaries

Before proceeding to the problem formulation in Section 4.3, we must discuss some preliminaries. Geospatial tasks such as adaptive sampling or coverage planning are path-dependent which increases the difficulty of the problem.

4.3 Problem Formulation

Define the configuration of an agent to be $c = (lat, lon, d, t) \in \mathcal{C}$ where lat is the latitude, lon is the longitude, d is the depth beneath the ocean's surface, and t is the time since the epoch (mission start). A trajectory $\mathcal{T}$ is a time indexed curve $c(t) \in \mathcal{C}$ defined for time domain $t \in [t_a, t_b]$ Assume that the vehicle is capable of controlling its depth d. For lateral and longitudinal motion, the agent is capable of generating thrust velocity vector v. v is in the local frame with respect to local flow fields and in meters per second. Further note that superscript (i) with $i \in \{1, 2, \ldots, N\}$ denotes agent number with $N = N_{agents}$.

Given the flow field, starting configuration c_0 and control signals $u(t) = (v(t), d(t))$ it is possible to construct trajectory $c(t) = \phi(c(t_a), u(t))$ from the ocean forecast data. When applying various algorithms, trajectories and actions will be discretized to facilitate use of numerical methods or discrete algorithms. A trajectory $\mathcal{T}$ is feasible if there exists a control action sequence $\mathbf{u}$ that can reproduce the trajectory. Similarly define a multi-trajectory $\mathcal{M}_{\mathcal{T}} = \{\mathcal{T}^{(i)}\}_{i=1}^{N}$ that is generated by applying the multi-action sequence to the team of agents. The space of all feasible multi-trajectories is $\mathcal{M}_{\mathscr{T}}$.

We define a path-dependent reward function $R : \mathcal{M}_{\mathcal{T}} \to \mathbb{R}$ mapping multi-trajectories to real values. The objective of the problem is to find the feasible trajectory satisfying system dynamics that maximizes the reward:

$$\mathcal{M}_{\mathcal{T}}^{*} = \underset{\mathcal{M}_{\mathcal{T}} \in \mathcal{M}_{\mathscr{T}}}{\arg\max}\, R(\mathcal{M}_{\mathcal{T}}) \tag{4.1}$$

Assume that all trajectories consist of a discrete set of configurations $\mathcal{T} = \{c_\kappa\}_{\kappa=0}^{T}$. All planners assume macro actions where a constant depth, constant velocity command $u_\kappa = (d_\kappa, t_\kappa, v_\kappa)$ is executed until the time t_κ passes. Define control action sequence $\mathbf{u} = \{(d_\kappa, t_\kappa, v_\kappa)\}_{\kappa=1}^{T}$ to be a sequence of T subsequent commands. For the multiagent problem, define a multi-action sequence $\mathbf{U} = \{\mathbf{u}^{(i)}\}_{i=1}^{N} \in \mathcal{M}_{\mathcal{U}}$ to be a collection of control action sequences for each agent.

4.3.1 Reward functions

For this work, we investigate two different reward functions: coverage with a finite sensing radius, and environmental sensing to select regions in a given temperature range.

4.3.1.1 Coverage

Area coverage of a multi-trajectory with a finite sensor footprint is of interest. The sensing radius can be defined either by using the sensor range (e.g. range of sonar for target search tasks) or by assuming a feature length over which sensor measurements decorrelate. One natural example includes the Rossby Radius [178], which defines length scales of correlated sensor measurements when modeling ocean processes. The Rossby Radius in the Gulf of Mexico varies but is about 30-40km in the Gulf of Mexico [179]. When combined with Nyquist's theorem this suggests that sensing measurements be spaced 15-20km apart. Area coverage is path dependent because overlaps reduce total area covered. Consider a sweep sensor of fixed radius that is swept between waypoints. This generates a capsule (a rectangle with semi-circle "caps" at both ends) which can be well-approximated with a rectangle (Fig. 4.2).

4.3.1.2 Environment Sensing: Temperature

Missions intending to sample locations based on temperature include tracking algae blooms [180], which often favor a specific range of temperature conditions [18] (See Fig. 4.3). Supposing there is some desired temperature T_0, one can generate a reward function that favors sampling locations nearest T_0, where T is the temperature at configuration c:

$$I(c) = \begin{cases} e^{-a_p(T-T_0)} & \text{if } T \geq T_0 \\ e^{-a_n(T_0-T)} & \text{if } T < T_0 \end{cases} \tag{4.2}$$

a_p is a scale factor for the positive case and a_n is a scale factor for the negative. Letting $a_p \neq a_n$ allows one to ensure that the region $\{c : I(c) \geq b\}$ for $b \in (0, 1)$ is a guaranteed fraction of the environment, η. Decreasing η increases the difficulty of the problem by having fewer locations of higher reward.

For a distribution/population of temperatures T, define the percentile $P(T)$ which denotes the percentage of values below T. We wish to find a range of percentiles $[P^-, P^+]$ containing $P(T_0)$, and $P^+ - P^- = \eta$. We can then solve for T^+ and T^- by solving the equations $P^+(T^+) = P(T_0) + \frac{\eta}{2}$ and $P^-(T^-) = P(T_0) - \frac{\eta}{2}$. Starting with the first case statement, compute a_p by setting $e^{-a_p(T^+-T_0)} = b$ resulting in $a_p = \frac{-\log b}{T^+-T_0}$. Similarly, $a_m = \frac{-\log b}{T_0-T^-}$. This choice of parameters guarantees the requirement that $\{c : I(c) \geq b\}$ is a fraction of the environment.

When evaluating the reward a multi-trajectory gathers, multiple measurements at the same location reduce overall information gathered. To penalize multiple coverings, discretize the region into a finite set of configurations C on a grid covering the region of interest. Draw a ray between waypoints; all cells touching the ray are considered visited once by the agent during the trajectory. Sensing a cell the nth time accrues $\frac{1}{n}$th of the reward. The total reward gathered is the sum of all rewards from all cells visited. For a given region, one can define the total information content of the environment. Total information content can vary wildly within a larger data-set, so the reward content of a region must be normalized. Define the total information content of the region $I_{total}(C) = \sum_{c \in C} I(c)$. Note that this does not account for multiple passes of many agents, and the normalized reward can exceed 100%.

4.4 Representations

The problem formulation described in Section 4.3 does not impose a given method for enforcing that the solution (a multi-trajectory) is dynamically feasible. Before proceeding on to mentioning existing algorithms (Section 4.5) or the proposed approach to solving the problem (Section 4.6), discussing the representation of the space of dynamically feasible trajectories $\mathcal{M}_{\mathcal{T}}$ is needed to put the algorithms to be described in proper context. In order to solve (Eq. 4.1) numerically satisfying feasibility constraints, some assumptions must

be made. It is sensible to only search in the space of feasible trajectories instead of having to actively enforce feasibility constraints, but there are two fundamentally different ways of doing that. To simplify the formulation further guaranteeing searching in the space of feasible trajectories, $R(\mathcal{M}_\mathcal{T})$ can be recast as a function of a multi-action sequence, $\tilde{R}(U) = R(\phi(c(t_0), U))$ (Eq. 4.3),

$$\mathbf{U}^* = \arg\max_{\mathbf{u} \in \mathcal{M}_\mathcal{U}} \tilde{R}(\mathbf{U}) \tag{4.3}$$

or as a function of a feedback policy that maps multi-trajectories to multi-actions, $\hat{R}(\pi) = R(\phi(c(t_0), \pi(c(t))))$ (Eq. 4.4):

$$\pi^* = \arg\max_{\pi \in \mathcal{M}_\Pi} \hat{R}(\pi). \tag{4.4}$$

Both of these techniques search in the space of feasible trajectories, but require varying degrees of additional assumptions or simplifications to work well in practice. Before describing the techniques that can solve (Eq. 4.1), we now describe these representations in detail.

4.4.1 Action Sequence Representation

To ensure that all multi-trajectories are feasible, one could recast the optimization problem (Eq. 4.1) to search over all action sequences (Eq. 4.3). Given any multi-action sequence $\mathbf{U} \in \mathcal{M}_\mathcal{U}$, the resulting trajectory is feasible. Even when restricting the problem to macro actions, action sequence

parameters are continuously valued. Restricting the available multi-actions to a finite set enables use of discrete optimization techniques such as the brute force method that enumerates over all candidate solutions. The task of an action sequence based solver is to find the multi-action sequence that maximizes total reward.

4.4.2 Policy Based Representation

Another way to restrict the solver to the set of feasible multi-trajectories is to search in the space of policies (Eq. 4.4). This extends the notion of optimizing over $\mathcal{M}_{\mathcal{U}}$ by optimizing over families of action sequences when considering disturbances. Representing policies requires additional care in the formulation so we now define the discrete stochastic motion model to enable use of a discrete policy representation. Consider that there are N agents, where agent i resides at the configuration $s^{(i)}$ in a finite configuration space $\mathcal{S}^{(i)}$. Since all agents are homogeneous and collisions between agents can be safely ignored, all agents share a common motion model (state space, motion model, transition probabilities). At each state s, the agent may select from a finite action set $a \in \mathcal{A}(s)$. Define the joint state space of all agents $\mathcal{S} = \mathcal{S}^{(1)} \times \mathcal{S}^{(2)} \times \ldots \times \mathcal{S}^{(N)}$ and joint action space $\mathcal{A} = \mathcal{A}^{(1)} \times \mathcal{A}^{(2)} \times \ldots \times \mathcal{A}^{(N)}$. For discrete algorithms subscript $t \in \{1, 2, \ldots, T\}$ denotes the time index. For iterative algorithms use the subscript k to denote the kth iteration. The motion models of concern are Markovian systems, where the Markov property

is satisfied:

$$\Pr(S_{t+1} = s' | S_t = s_t, A_t = a_t) = \tag{4.5}$$
$$\Pr(S_{t+1} = s' | S_t = s_t, \ldots, S_0 = s_0, A_t = a_t, \ldots, A_0 = a_0)$$

The probability mass function $\mathcal{P}^a_{ss'} = \Pr(S_{t+1} = s' | S_t = s, A = a)$ defines the transition probabilities for the agent given the present state and action selected. All aspects of the motion model can be time varying by absorbing the time into the state (therefore planning in space-time). We are interested in larger problem domains where, e.g. $|\mathcal{S}| \geq 100 \times 100 \times 10 = 100,000$, $N \geq 10$, and $|\mathcal{A}| \approx 10$. Further, define the trap state s_T that captures all failure states (collisions) and terminations (going beyond the maximum time limit). For any action, s_T transitions to itself with probability one. Define the space of execution histories $\mathcal{H} = \{h = s_0 a_0 s_1 a_1 \ldots s_t, h \text{ is feasible.}\}$ to consist of all feasible execution histories. Given the execution histories of $M \leq N$ robots, the joint reward is defined at termination. A multi-history is a set of multiple histories of N agents: $\mathcal{M}_{\mathcal{H},N} = \{\mathcal{H}_{i=1}^{(i),N}\}$. The reward function maps multi-histories to real numbers $R : \mathcal{M}_{\mathcal{H},N} \to \mathbb{R}$. Note that portions of the described formulation are similar to typical Markov Decision Process formulations mentioned in Section 2.7, but differ in the reward structure.

While discrete rollouts of the stochastic system may not be feasible trajectories, they approximate the continuous system's performance well enough for optimization purposes, and discrete stochastic policies can control the

continuous system in the following manner. First, the location of an agent in continuous coordinates is converted to a discrete state. An action is sampled according to the policy and the discrete state. The action is applied to the continuous system and the sequence is repeated, with the reward of the rollout determined at the termination of the trial. This technique generates feasible paths and enables fair comparison between discrete and continuous based methods.

The goal of a policy-based solver to find a good policy π of the form $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that maximizes the expected reward $\mathrm{E}_\pi[R]$. Alternatively, a policy can also be dependent on the execution history: $\pi : \mathcal{M}_{\mathcal{H},N} \rightarrow \mathcal{A}$. A joint policy maps joint state spaces to joint action spaces. Alternatively, agent i can have its own independent policy where $\pi^{(i)} : \mathcal{S}^{(i)} \rightarrow \mathcal{A}^{(i)}$.

4.5 Existing Algorithms

We now discuss state-of-the-art algorithms that are capable of coordinating teams of underwater vehicles for geospatial tasks. All the algorithms described in this section are action sequence based approaches and include the brute force method, the greedy method, and Stochastic Gradient Ascent (Sto.GA). These algorithms comprise a set of competitive algorithms against which we benchmark our proposed methods.

4.5.1 Brute Force Method

Similar to the discrete techniques, we will discretize the space of all action sequences, and have the agents exhaustively search the entire action space. To sidestep searching the joint action space, agents plan sequentially using sequential greedy allocation. Agent i tests all action sequences in the action space and evaluates their reward, accounting for all other agents that have previously selected action sequences. The action sequence with the highest reward is stored and used in planning the action sequences of all subsequent agents.

The main challenge is further limiting the action space to ensure that the search is tractable. For example, for 15 available actions planning action sequences of length 7 results in $15^7 \approx 171M$ action sequences per agent! However, having the agent select an action every 48hr instead of 24 hr requires generating action sequences of length 4, resulting in searching a space of $15^4 = 50,625$ action sequences per agent, which is quite feasible.

Forecast uncertainty can be accounted for by using Monte Carlo rollouts when generating and evaluating multi-trajectories from the selected action sequences but lies beyond the scope of this paper.

We further note that since brute force method significantly restricts the number of actions selected every 48 hrs, it is possible for other action sequence methods to find better solutions than brute force by more intelligently searching a much more expressive space of action sequences.

4.5.2 Greedy Method

A more dramatic variation of the brute force method is to have the method select a single action that lasts for the entire duration of the 7 day horizon. This generates such a small action space such that good coordinated solutions can be generated quickly. This is denoted as the greedy method. In effect, the greedy solver is using a greedily (i.e. sequentially) coordinated constant action sequence heuristic to search over the discrete set of multi-actions $\mathcal{M}_{\mathcal{A}}$ in $\mathcal{M}_{\mathcal{U}}$. Due to the lack of branching for the greedy solver, the space of actions for each agent $\mathcal{A}$ can be much larger than for the brute force method. The strength in this method lies in its ability to quickly explore an expressive space of coordinated action sequences.

4.5.3 Stochastic Gradient Ascent

Rather than discretizing the space of all action sequences, we can instead attempt to plan in the continuous space of action sequences. To keep this problem tractable we can consider using a stochastic gradient-based technique which iteratively improves an initial set of action sequences until the solution converges. We consider a method termed Stochastic Gradient Ascent (Sto.GA), which we will briefly explain here.

Sto.GA starts by initializing $\mathbf{U}$ using a desired initialization technique. In this work we initialize Sto.GA using the greedy solver. To plan for multiple agents, Sto.GA uses sequential greedy allocation by optimizing the ac-

tion sequence for one agent while holding all others constant. One of the major difficulties in using an optimization based technique is in calculating the gradient update to the action sequence. There is no clear analytic technique to calculate this gradient in the action-sequence-space when the rewards functions we are considering are defined over the state-space. Sto.GA solves this problem by utilizing techniques similar to that from Stochastic Trajectory Optimization for Motion Planning (STOMP) [181], which uses sampling to approximate the gradient of a non-differentiable cost function. Energy-Efficient Stochastic Trajectory Optimization (EESTO) further extends STOMP to consider time-variations between waypoints [182]. By sampling perturbations to the individual actions of the agents' action sequences, the agents can construct a set of action sequences. Sto.GA then evaluates each of these sequences, assigning a score to each, and then uses a weighted recombination of the sampled perturbations in combination with the sequence evaluations to approximate the gradient. Sto.GA iteratively continues this process until the solution converges or a maximum number of iterations is reached. Sto.GA includes multiple matrix operations which are already implemented in parallel in many matrix libraries, including the ones used in this work.

We now discuss various algorithms mentioned in the Literature Review (Chapter 2), determining which algorithms are unsuitable for the task at hand:

- Coordination techniques such as M* [21] are for path independent re-

wards. Agent interactions are more common for path dependent rewards due to increased occurence of path overlap instead of collisions.

- Heuristic guided search techniques, including the $\epsilon-$admissible B&B developed in Chapter 3 are also unsuitable because generating effective heuristics that account for the effects of the ocean currents is intractible.

- Coverage based planning algorithms (Section 2.5) are unsuitable since they require strong propulsion capabilities to reliably execute coverage patterns.

- Locational Optimization techniques are not considered in this scenario as they are used to optimize path independent rewards of a given form and do not optimize the objective functions of interest. However, a similar scenario employing Locational Optimization techniques will be investigated in Chapter 5.

4.6 Approach

We now outline the techniques we created to solve the geospatial tasks at hand, which are the primary contributions of the paper. Starting with policy based approaches, we first discuss the default policy. We then describe how we extend Monte Carlo Tree Search (MCTS) and Cross Entropy Method (CEM) to improve upon the default policy. Subsequently, we describe CEM applied to action sequences (CEMAS) and the iterative greedy method. The

section finishes with a discussion on how to rationally compare tradeoffs in time to compute vs. solution quality.

4.6.1 Policy Representation

To make implementation feasible, policy based approaches must search a discrete space of policies. We assume that a policy π is implemented as a combination of look-up tables, hash maps and lists that map states or execution histories to (distributions of) actions. Consider the use of arrays or hash-maps. Because N is fairly large ($N \geq 10$), policies defined in the joint space are computationally intractable. We therefore favor the use of single agent policies where $\pi^{(i)} : \mathcal{S}^{(i)} \rightarrow \mathcal{A}^{(i)}$ are independent of other agents. Policies may be Markovian (only dependent on present state) or history dependent. Policies may be deterministic by storing a single action for each state, or stochastic by maintaining a distribution of actions for each state. Policies may either be shared by all agents, or each agent may have its own independent policy.

The Markov random policy is of the form (Eq. 4.6):

$$\pi_j(s) = \Pr(A_t = a_j | S_t = s) \tag{4.6}$$

where $\pi(s)$ denotes a probability distribution over all actions for state s. If $\forall s \in \mathcal{S}, |\mathcal{A}(s)| \leq N_{actions}$, then π is a look-up table with $|\mathcal{S}| \times N_{actions}$ parameters. Note that time varying policies are possible when the discrete

state S encodes the time as well as system configuration.

History dependent random policies are of the form (Eq. 4.7):

$$\pi_j(h) = \Pr(A_t = a_j | H_t = h) \tag{4.7}$$

Use of a look-up table for history dependent policies is intractable. Instead, we use a hash-map. We use the most recent state in the history as a hash, with hash collisions stored in a list that remains short in practice. If all agents share the same policy, a single policy π is all that is required. Otherwise, agent i has policy $\pi^{(i)}$. A deterministic policy is similar to the random policy, with the only difference that instead of storing probability distributions over all actions for each state/history, a single action is stored.

4.6.2 Default Policy

Consider first the set of all random Markovian policies shared by all agents. The choice of a shared Markov policy may seem unsound for solving multi-agent, history dependent reward structures but it has proven surprisingly effective. Random Markovian shared policies have the fewest number of parameters to tune and consume the least amount of memory with the fastest sampling rate making them suitable for fast Monte Carlo rollouts. In a large system of homogeneous coordinating agents in the deep ocean, the reward function exhibits an exchangeability of agent locations. It doesn't matter which agent is where, but given the same scenario, each agent ought make

the same decision (or sample from the same distribution of decisions). In previous work, multiple agents have shared and executed *the same policy* for solving MDP problems [183], enabling coordination in a large swarm. One of the major challenges in the MDP case is that deterministic policies tend to collapse the distribution of agent locations to a small subset of states that maximize the reward (consider SSP problems [67]). This is undesirable for rewards such as coverage planning, suggesting use of random policies instead. Consider a well-designed random Markovian policy that generates a limiting distribution (analogous to the stationary distribution of Markov Chains) of populations of agents, evenly distributing the population in such a manner that increases dispersion while also maximizing reward. Controlling distributions of agents has been employed successfully for various coordination problems [184, 185]. Actively controlling the stationary distribution is computationally burdensome as computing the stationary distribution requires expensive (sparse) linear algebra operations. We instead prefer to indirectly approximate distributions using efficient Monte Carlo rollouts.

In the absence of any additional information, a good initial policy is to select from the set of valid actions uniformly at random, which is denoted as the default policy. Starting with a uniform distribution over all actions, certain actions (such as diving beneath the ocean floor) are guaranteed to cause the agent to enter the trap state s_T and should be pruned by setting the action's probability to zero.

When the uncertainty of the stochastic motion model is purely introduced

by the agent due to positional uncertainty and not due to motion/forecast uncertainty, the performance of the random policy controlling the continuous system can be boosted by sampling the policy many times and selecting the continuous rollout with the highest reward. This performance boosting technique pertains to any random policy generated by any algorithm.

4.6.3 Monte Carlo Tree Search

We now extend MCTS as it was presented in Section 2.8 to be able to coordinate large teams of agents gathering path dependent reward. Given a node N_S in the tree, with present state S, MCTS employs the following modified version of UCB1 (Eq. 4.8) for the selection procedure:

$$UCB1(N_S) = \underset{a \in \mathcal{A}(S)}{\arg\max} \left(\frac{\bar{R}_a}{R_{max}} + C \sqrt{\frac{\log(\sum_a n_a)}{n_a}} \right) \tag{4.8}$$

R_{max} is the largest reward ever observed. $\bar{R}_a$ is the average of all rewards gathered by selecting action a at node N_S. n_a is the number of times action a was played at node N_S, $\sum_a n_a$ is the number of total plays at N_S. C is a parameter that is experimentally tuned to favor exploration vs. exploitation. The only significant modification here is the inclusion of R_{max} which normalizes all reward averages to the range $[0, 1]$ facilitating better tuning of C.

4.6.4 Cross Entropy Method

We extend previous work using CEM (Section 2.9) to maximize the reward a policy gathers for a MDP [85] to path dependent reward problems in a multiagent setting. We denote this extension as CEM applied to policies (CEMAP). To do so, we apply CEM to the UUV coordination problem, letting $\theta = \pi$. and $\phi = \hat{\pi}$. Let $\mathcal{P}_{\mathcal{H},N}$ be the set of the $(1-\rho)$ percentile best multi-histories for N agents at the iteration. Define observation set $\mathcal{O}$ that generates state/history-action observations from $\mathcal{P}_{\mathcal{H},N}$ for the maximum likelihood estimate. A single agent history $h_t = s_0 a_0 s_1 \dots a_{t-1} s_t$ generates Markov $o = (s, a) \in \mathcal{O}_s$ observations $\{(s_\tau, a_\tau)\ 0 \leq \tau \leq t-1\}$ or generates history dependent $o = (h, a) \in \mathcal{O}_h$ observations $\{(\prod_{\sigma=0}^{\tau}(s_\sigma, a_\sigma)s_{\sigma+1}, a_{\sigma+1}), 0 \leq \tau \leq t-1\}$. For individual policies, $\mathcal{O}^{(i)}$ is generated from agent i's corresponding execution histories in $\mathcal{P}_{\mathcal{H},N}$, or $\mathcal{P}_{\mathcal{H}^{(i)}}$. Each agent updates its own policy with its own execution histories, yet coordination occurs by globally selecting the best multi-histories. For shared policies, outcome set $\mathcal{O}$ batches all trajectories together and does not distinguish observations from various agents.For Markov random policies, let:

$$\hat{\pi}_j(s_k) = \frac{\sum_{o \in \mathcal{O}_s} I_{\{(s_k, a_j)\}}}{\sum_{o \in \mathcal{O}_s} I_{\{(s_k, a)\}}} \tag{4.9}$$

where $\hat{\pi}_j(s_k)$ is the relative rate of action a_j being selected in all pairs $o \in \mathcal{O}_s$ compared to all actions selected at state s_k. For history dependent random

policies, let:

$$\hat{\pi}_j(h_k) = \frac{\sum_{o \in \mathcal{O}_h} I_{\{(h_k, a_j)\}}}{\sum_{o \in \mathcal{O}_h} I_{\{(h_k, a)\}}} \tag{4.10}$$

The resulting policy parameters $\hat{\pi}$ are filtered through the low-pass filter, since convex combinations of probability distributions are valid probability distributions.

CEM can also be applied to searching the space of action sequences, which we denote as CEMAS. Let agent i's action at time k be random variable $A_k^{(i)}$. We assume that $A_k^{(i)} \perp A_l^{(m)}$for $k \neq l$ and $i \neq m$ with $\Pr(A_k^{(i)} = j) = p_{j,k}^{(i)}$. Let $\theta = [p_{j,k}^{(i)}]$ and apply Algorithm 7.

CEM is general in that the independence assumption of $A_k^{(i)}$ can be broken, but that dramatically increases the number of parameters for CEM to tune. Similar to the brute force method, CEM can be configured to have actions last 48 hours instead of 24 hours (denoted as CEMAS-2 for selecting actions that last 2 days). This will offer a benchmark to see how well CEM does in comparison to the brute force method. CEM in any of its forms can be readily parallelized by distributing the N_p rollouts to multiple M processes, speeding up the compute time. Little data is shared between processes, where the updated policy is transmitted to all workers and each worker returns the rollouts.

4.6.5 Iterative Greedy Method

The greedy method can be further improved upon by selecting constant actions that are of shorter and shorter duration in an iterative fashion. Starting with $k = 1$, take the best $k-1$ actions previously discovered, and set all "future" actions $k, k+1, \ldots, T$ in the action sequence to a constant $\mathbf{a} \in \mathcal{A}$. Store the action $\mathbf{a}$ that maximizes the total reward gathered over all $\mathcal{A}$. Increment k by one. This continues until $k > T$ or the termination condition is met. This is the iterative greedy solver and is described in Algorithm 12.

The iterative greedy solver is guaranteed to find a solution of equal or greater reward than the greedy solver. The first iteration with $k = 1$ generates the same solution that the greedy solver generates. The best known multi-action is only changed if an improvement is discovered. The iterative greedy solver is therefore an anytime algorithm. After the $k = 1$ iteration completes, every agent has an action sequence that is feasible and spans the entire mission duration. Subsequent iterations offer additional improvements. Various termination conditions include running out of time to compute the best solution, or requiring that R_{max} improves by a factor $\epsilon > 0$ between iterations of k. For example, the second mentioned termination condition will terminate the algorithm after the $k = 2$ iteration if the iterative greedy solver does not improve the greedy solution. Refer to Fig. 4.4 for an example diving sequence that is improved upon by the iterative greedy solver.

The iterative greedy solver can be parallelized by dividing up the actions

at each time step amongst multiple processors, but this smaller batch size compared to CEM for example will experience smaller performance gains from parallelization.

4.6.6 Assessing tradeoffs in compute time vs. solution quality

In order to assess the tradeoffs of using more time consuming algorithms, we wish to assess when it is rational (cost saving) to spend additional time to compute a better solution. We note that in 2018 dollars, a 48 physical core machine with 384 GB RAM in a cloud computing environment can be rented for 24 hours at about $110 per day. Run times will be for single process algorithms and we will ignore the fact that the various algorithms can be parallelized, meaning that computing costs are overestimates of actual costs. To underestimate daily vehicle deployment costs to more conservatively estimate against the cost effectiveness of longer compute times, we take the minimum of $200 per day per UUV or $30,000 per day for the surface ship monitoring the fleet.

Suppose Algorithm A took t_A seconds to compute a solution of reward R_A while Algorithm B took t_B seconds to compute a solution of reward R_B. Suppose that Algorithm A finds a better solution than Algorithm B ($R_A > R_B$) but takes more time to compute ($t_A > t_B$) with positive $\eta = \frac{R_A - R_B}{R_B}$. Suppose that a mission terminates after a given threshold of information is gathered. To save vehicle deployment costs, Algorithm A will complete the mission faster or with fewer vehicles, implying that algorithm B will take a

factor of $(1+\eta)$ more total vehicle deployment cost to complete the mission. Supposing that deploying Algorithm A on the vehicles costs C_{deploy} while Algorithm B costs $C_{deploy}(1+\eta)$ due to less efficient deployments. Total effective deployment costs for Algorithm A are $C_A = C_{deploy} + t_a C_{compute}$ while the total costs for Algorithm B are $C_B = (1+\eta)C_{deploy} + t_b C_{compute}$.

Given these constraints, there is a break-even point for the minimum performance improvement required by the algorithm for a given amount of compute time by dividing the performance improvement. For 128 UUVs, the \$200/day operating costs amount to \$25,600. It is considered rational to spend an extra 24 hours of compute time on a better algorithm if the better algorithm is $\eta = 0.43\%$ better. This is in line with the observation that computational resources are cheap and deployed robotics hardware is expensive.

4.7 Simulated Experiments

4.7.1 Ocean Environment

For this work, we use the Navy Coastal Ocean Model (NCOM) to model the ocean forecast for the Gulf of Mexico (GoM), which is contained in a bounding box spanning latitudes 18°N-30.8°N and longitudes 80.25°W - 98.0°W [186](this corresponds to the same area charted in Figs. 4.1 and 4.3). For each day, the NCOM model describes ocean current velocity, temperature and salinity forecasts at 72 depths, at 2km resolution in latitude and

longitude, and 3hr temporal resolution for 7 days.

The Gulf of Mexico is divided up into non-overlapping $3^\circ \times 3^\circ$ regions (roughly 300km on a side). All experiments will use forecast data from the Gulf of Mexico from Dec 08, 2017. Deployment zones are placed in the center, and vary in size (20km or 200km) depending on the experiment. For 200km deployment zones, if the deployment zone is less than 2/3 water, the region is discarded. For 20km deployment zones, many deployment zones near the center are tested until one is found that is in open water. Otherwise the region is discarded. Starting locations are placed uniformly at random inside the deployment zone. To ensure that agents coordinate, up to four agents are placed at a starting location until all agents have been assigned a starting location. All algorithms benchmarked are tested in the same environments with identical deployments and reward functions.

4.7.2 Reward Functions

For the temperature-based reward function, T_0 is set to the mean temperature for the entire GoM region (this is consistent across all $3^\circ \times 3^\circ$ regions, so the information content varies widely between regions), and parameters for the reward function are tuned such that $b = 0.5$ and $\eta = 20\%$. The grid size is 2km. The information content of each $3^\circ \times 3^\circ$ region varies based on local variations in temperature and can vary by over 2 orders of magnitude. Therefore, temperature-based rewards are normalized by the total information content of the region I_{total}.

For the coverage-based metric, computing the coverage area is then the task of taking the union of overlapping polygons, and computing the area. This requires an efficient polygon union operation. The Cascaded Union from the GEOS library [187] is an efficient union operation that is capable of processing many polygons, often for Geospatial Information Systems. The Cascaded Union packs the polygons into an R-tree using the Sort-Tile-Recursive algorithm [188], and takes the union of the individual polygons in a bottom-up approach, resulting in dramatically faster run times than a naive binary union operation. The sensing radius is set to 15km, which is half of the average Rossby radius in the GoM (30 km [179]).

4.7.3 UUV model

We assume that the UUVs have active buoyancy control and can maintain their desired depths. To minimize energy consumption, we assume that each UUV has enough thrust to sustain a speed of at most 0.1 meters per second relative to the local frame that is moving with the currents. In other words, the platform has limited control authority and may not necessarily overpower the ocean currents. For reference, forecast data suggests that the average current speed for a given depth does not go below 0.15 meters per second, meaning that ocean currents will typically dominate vehicle thrust capabilities.

4.7.4 Stochastic Motion Model

Discretizing the ocean forecast model into a discrete model requires discretizing the action space and the configuration space. We select a discrete set of depths over which to plan. Next, we select a discrete set of thrust velocities. Taking the Cartesian product of these sets generates the full action set.

Planning in space-time accounts for the time varying effects of the field. Each discrete coordinate is assigned a unique state number S and is associated with the configuration $c = (lat, lon, d, t)$. Each state consists of a corresponding cell in (lat, lon) $g_S = \{(\theta, \phi) \in [lat, lat + \delta lat] \times [lon, lon + \delta lon]\}$. Any transitions from the evolution of the continuous system into the cell g_S is associated with discrete state S. In order to generate the outcome probabilities, particles are distributed on a uniform grid inside the state's cell. The selected action is applied to all particles, the flow field data is integrated over the given time interval and the resulting distribution of particles is recorded, approximating the transition probabilities from cell to cell when issuing a given command at the present state. This data is then stored in a look-up table for future queries.

Due to discretization, the motion model introduces positional uncertainty of the agent's location, as the stochastic motion model is indifferent to the exact location of the agent within a cell. This in turn propagates through the flow field dynamics (Fig. 4.5). It is also possible to introduce forecast uncertainty into the motion model by including additional particles and Monte

Carlo rollouts of uncertain forecast outcomes. This is compatible with ensemble techniques that handle forecast uncertainty [148]. We assume that forecast uncertainty will dominate positional uncertainty since the size of the cells is relatively small and particle movement of adjacent particles is highly correlated. Positional uncertainty is small relative to total distance traveled. Calculating statistics on particle populations shows that the deviation of the particles for each cell is around 3.5km while the particles travel around 20km on average.

For this work the stochastic motion model uses 4-km cells, with 9 particles to approximate the transition probabilities given the state and action to be simulated. While 9 particles are used for each (state,action) pair, many particles transition to the same state and can be combined, so on average 4-5 unique outcomes are generated. With $v_{max} = 0.1$, define the set of thrust velocities $\{(0,0),(v_{max},0),(-v_{max},0),(0,v_{max}),(0,-v_{max})\}$ along with diving depths $d \in \{100, 500, 1000\}$, resulting in 15 actions for the agents to select. Each discrete action lasts for 24 hours.

4.7.5 Algorithm Configurations

CEM will conduct at most 100 iterations of policy improvement. CEM will terminate if the last 5 values of $\hat{\gamma}_t$ do not fluctuate by more than 0.5% since the standard convergence criteria do not apply for the problem at hand due to the stochastic nature of the problem. CEM parameters α, ρ will need to be tuned using training data for each experiment. CEMAP will set $N_p = 5,000$

while CEMAS will set $N_p = 2,500$. Since MCTS is an anytime algorithm, it is often given a time budget in which it can compute a solution. To maintain better repeatability between trials, MCTS conducts 30,000 rollouts when generating a policy. Because the default policy (Sec. 4.6.2), CEMAP and MCTS use the stochastic motion model, they select amongst 15 actions that last 24 hours each. The brute force method must maintain a small solution space so while it uses the same 15 actions in the stochastic motion model, actions last up to 48 hours.

In contrast to the limited action set for the policy based approaches, both the greedy method and iterative greedy method can explore all 72 depths that define the NCOM model and 9 thrust velocities for a total of 648 actions per agent, far exceeding the 15 actions every 48 hours that the brute force method can exhaustively search. Sto.GA is not limited to exploring a discrete set of actions and can vary action values (including the time interval) continuously.

4.7.6 Experimental Setup

We benchmark the described algorithms in 3 experiments:

- Experiment 1: tune/benchmark various algorithms for 10 agents for environment sensing task
- Experiment 2: tune/benchmark various algorithms for 10 agents for coverage task
- Experiment 3: benchmark scaling the number of agents timing for

environment sensing task. Use parameters tuned for in Exp 1

Table 4.1 outlines all the selected algorithms benchmarked with the following abbreviations: M. is Markov, H.D. is History Dependent, det. is short for deterministic while rand. is short for random. Also, the term "shared" is for a single shared policy between all agents and while the abbreviation "ind." is individual policies for each agent. The column abbrev. in Table 4.1 denotes the shorthand or abbreviated name of the algorithm. CEMAS-2 (i.e. CEMAS with actions that last 2 days) is for reference and will only be tested in Experiment 1.

Table 4.1: The various algorithms benchmarked

abbrev.	policy rep.	solver	coordination
default	M. rand. shared	random	–
CEM	M. rand. shared	CEM	centralized
CEM h.d.	H.D. rand. shared	CEM	centralized
MCTS	M. det. ind.	MCTS	Seq. greedy alloc.
MCTS h.d.	H.D. det. ind.	MCTS	Seq. greedy alloc.
CEM ind.	M.R. ind.	CEM	centralized

abbrev.	solver	coordination
greedy	greedy/best constant depth	Seq. greedy alloc.
i.greed	iterative greedy	Seq. greedy alloc.
Sto.GA	Stochastic Gradient Ascent	Seq. greedy alloc.
bfm	brute force method	Seq. greedy alloc.
CEMAS	CEM applied to action sequences	centralized
CEMAS-2	CEMAS with 2 day actions	centralized

In terms of performance evaluation, benchmark algorithms will be evaluated for how much reward they gather, and how long it takes to compute the solution.

Evaluating a fixed action sequence for how much reward it gathers is straightforward by feeding the action sequence into the (now deterministic) simulator and observing the reward the resultant trajectory gathers. For the discrete policy based approaches, after a policy is derived, it is evaluated for how much information it gathers by using 5,000 Monte Carlo rollouts. Sample averages of discrete Monte Carlo rollouts sample the discrete motion model used during the optimization process and do not represent continuous system performance. However, since the discrete policy can drive the continuous system, continuous Monte Carlo rollouts are feasible by sampling different discrete actions while using the continuous system dynamics. As mentioned in Section 4.6.2, motion uncertainty is due to positional uncertainty imposed by the discrete policy and not due to the underlying system. Therefore, Monte Carlo estimates of the average and best continuous rollouts are of interest. In the presence of forecast uncertainty, the average of the continuous rollouts estimates the expected reward gathered by the policy. Otherwise, we retain the best continuous rollout during the Monte Carlo estimate, as this is a dynamically feasible trajectory and is a fair comparison to the benchmarked techniques that generate action sequences instead of policies.

Experiments 1 & 2 offer the primary benchmark of the various algorithms for two different reward functions for 10 vehicles. Experiment 1 will use the temperature-based reward function for evaluating multi-trajectory rewards, while Experiment 2 will use the coverage-based reward. To force collaboration amongst vehicles, vehicles are deployed in two teams of 4 and one team

of 2. Each team starts at the same location and must plan to disperse. There is a 20km deployment zone in each $3^\circ \times 3^\circ$ regions in which teams of vehicles are deployed uniformly at random. All mentioned algorithms in Table 4.1 will conduct the same trials with the same starting configurations.

The algorithms will train in three regions with 20 trials each to tune the parameters α, ρ, and C for a total of 60 training trials. For tuning CEM, let $\alpha \in \{0.3, 0.4, 0.5, 0.6, 0.7\}$ and $\rho \in \{0.15, 0.2, 0.25\}$ for 15 total parameter configurations. Tuning CEMAS requires different parameter selection from CEM, therefore we let $\alpha \in \{0.5, 0.6, 0.7\}$ and $\rho \in \{0.05, 0.1, 0.15\}$ for 9 total parameter configurations. Similarly, for MCTS, we let C be 17 logarithmically spaced points between 10^{-3} and 10^{1}. e.g. $C \in \{1e-3, 1.8e-3, 3.2e-3, \ldots, 1.8, 3.2, 5.6, 10\}$. For validation the algorithms plan in five different regions with 20 trials each using parameters selected to maximize reward gathered by a subject matter expert, for a total of 100 trials.

Experiment 3 will vary the number of agents planned to observe how the algorithms scale to larger problems. The number of agents will vary among the set $N_{agents} \in \{8, 16, 32, 64, 128\}$. The deployment region will be 200kmx200km and centered inside the $3^\circ \times 3^\circ$ region. Agents will be deployed in groups of 4 uniformly at random. There will be 5 trials in 5 different regions, for a total of 25 trials.

Due to the extended run times for the larger problems, the best parameters identified from training in Experiment 1 will be applied to Experiment 3. Also, many solvers have different variants due to policy representation,

so only the best performing policy representation when validating results in Experiment 1 for the respective solver (MCTS or CEM) will be selected.

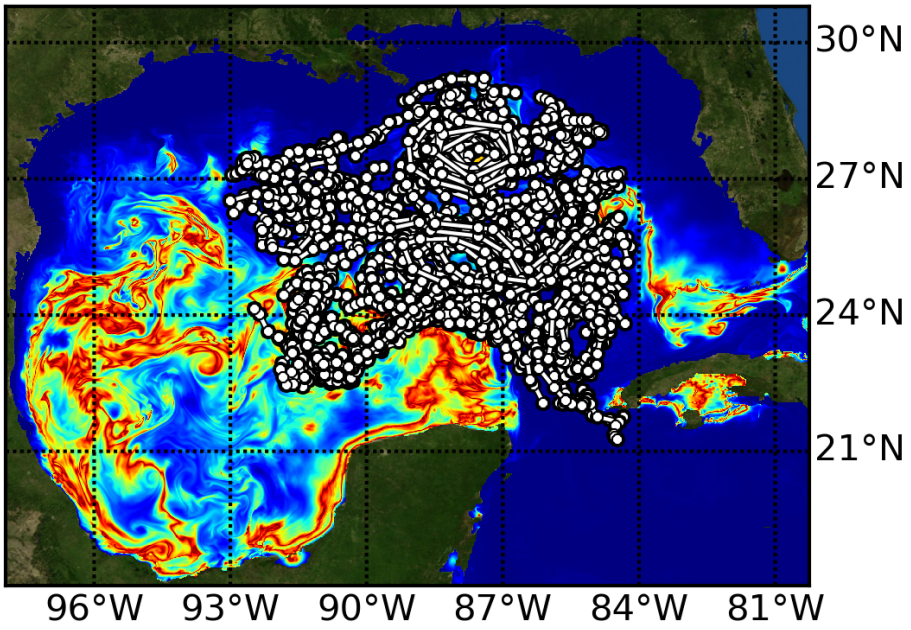


Figure 4.1: Hypothetical large deployment scenario of 256 agents in the Gulf of Mexico that motivates use of efficient coordination strategies for information gathering tasks in the ocean. Many of the trajectories overlap, compounding the evaluation of the plan. Even selecting amongst 15 distinct possible trajectories per agent results in $15^{256} \approx 10^{301}$ joint decisions, making many state-of-the-art techniques intractable.

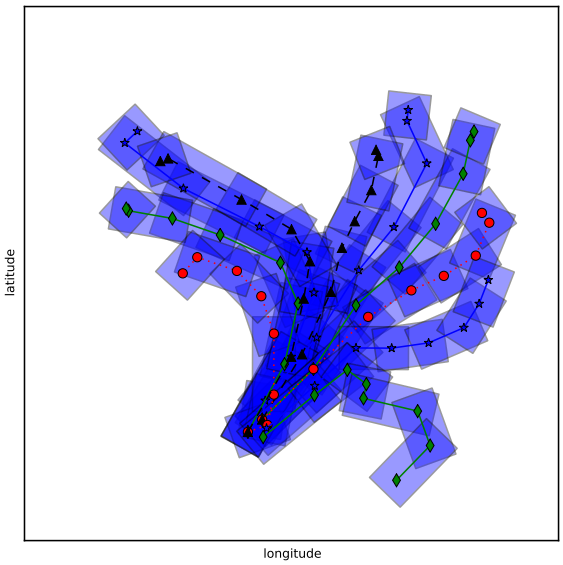


Figure 4.2: Example multiagent trajectory with polygonal approximation to the area covered by a sweep sensor. Here, 10 agents start close to each other and must coordinate to reduce overlaps (darker blue) to maximize coverage. The rectangle approximation is of equal area to the capsule generated by a finite sensor radius swept between two waypoints.

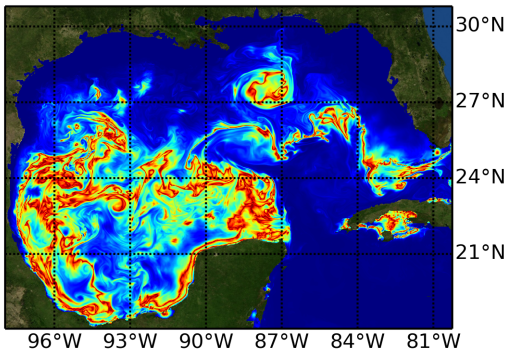


Figure 4.3: Temperature field in the GoM as defined by (Eq. 4.2) with $T_0 = 26.3°\text{C}$.

Algorithm 12 Iterative Greedy Solver

Require: discrete action space $\mathcal{A}$, reward function based on discrete multi-actions $\tilde{R}$

$\hat{\mathbf{A}}^* \leftarrow [null, null \ldots, null]$ {populate with null actions}
$R_{max} \leftarrow -\infty$
$k \leftarrow 1$
while $k \leq T$ and termination condition not met **do**
  for $i \in \{1, \ldots, N_{agents}\}$ **do** {Seq. Greedy Alloc.}
    $\mathbf{a}_i \leftarrow \hat{\mathbf{A}}^*[i]$ {get first $k-1$ actions from best known action sequence}
    for $a \in \mathcal{A}$ **do**
      $\mathbf{A} \leftarrow \hat{\mathbf{A}}^*$ {get best known action sequences for other agents}
      for $l \in \{k+1, k+1, \ldots, T\}$ **do** {set action k and all future actions to a}
        $\mathbf{a}_i[l] \leftarrow a$
      end for
      $\mathbf{A}[i] \leftarrow \mathbf{a}_i$ {update agent i's action sequence}
      $R \leftarrow \tilde{R}(\mathbf{A})$ {evaluate multi-action sequence for reward}
      if $R > R_{max}$ **then** {store best known action sequence}
        $R_{max} \leftarrow R$
        $\hat{\mathbf{A}}^*[i] \leftarrow \mathbf{a}_i$
      end if
    end for
  end for
  $k \leftarrow k + 1$ {shorten length of constant action sequences}
end while
return $(\hat{\mathbf{A}}^*, R_{max})$

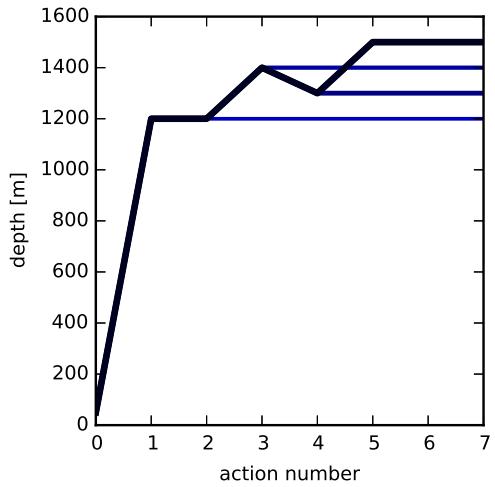


Figure 4.4: The iterative greedy solver iteratively improves initial greedy solution (thin blue), resulting in the thick black line. In this example, only the diving depth sequence for a single agent is shown for clarity.

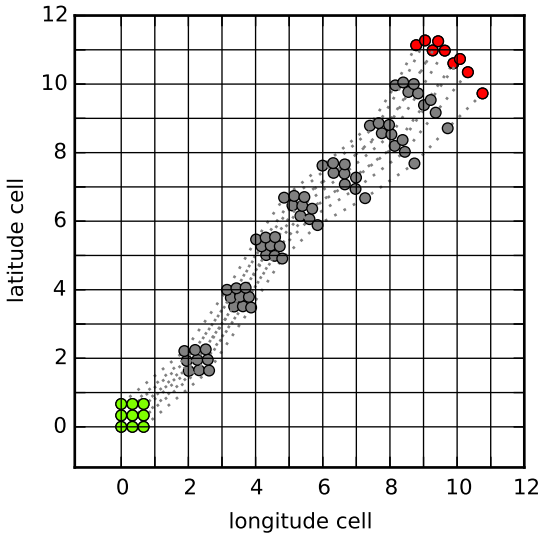
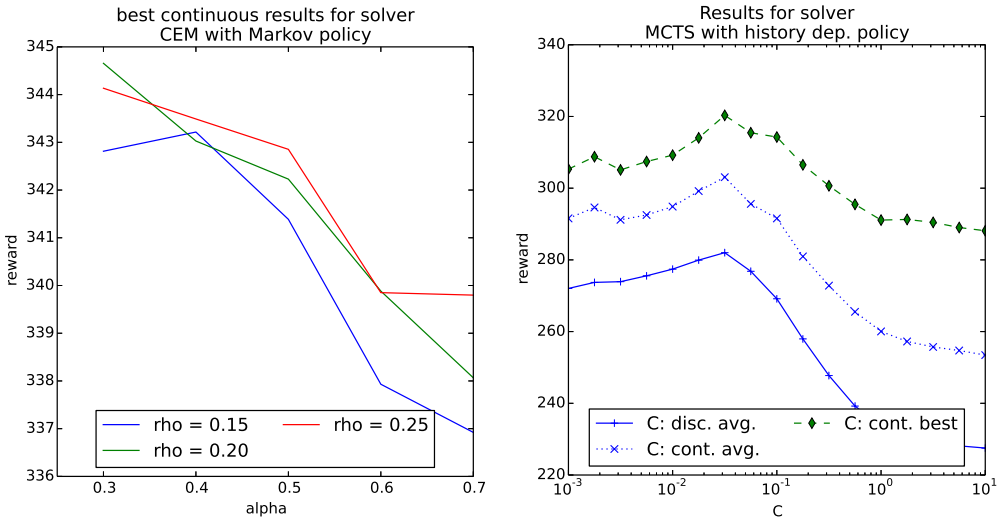


Figure 4.5: Example demonstration of building the stochastic motion model from ocean forecast data. Cell sizes are 4km square whose boundaries are denoted by the black grid lines. Nine particles start out uniformly covering the starting cell (0,0) in the lower left corner (green) and are allowed to drift freely with the ocean surface currents for 24 hours. The position of each particle is shown every three hours (grey), until the 24 hr time limit (red). Particles end up in five different cells in the upper right corner at the end of the action, resulting in 5 outcomes for the given cell and action

4.8 Results

4.8.1 Experiment 1



(a) Training results tuning parameters α, ρ for the Cross Entropy Method.

(b) Training results tuning parameter C for MCTS.

Figure 4.6: Training Results for Cross Entropy Method with Markovian policies for Experiment 1. The performance of MCTS and CEM vary with respect to the choice of parameters and are tuned with training data to improve algorithm performance. Parameter selection for CEM and MCTS is domain specific, and no method for calculating the best parameters *a priori* is known. Despite CEM having more parameters, its performance varies less due to parameter selection (2.4% variation) than MCTS (8.6% variation). Charts such as these are used to select parameters for validation runs and selected parameters are summarized in Table 4.2.

Training results in the first 3 regions for Experiment 1 suggested using the algorithm parameters in Table 4.2. Fig. 4.6 shows the resulting parameter sweeps for CEM with shared Markov Policies (4.6(a)) and MCTS with history

dependent policies (4.6(b)). CEM experience at most a 2.4% variation in reward during the parameter sweep, while MCTS experienced roughly an 8.6% variation, implying that CEM is less sensitive to parameter selection than MCTS.

The policy based solvers such as CEM or MCTS use discrete rollouts when generating the policy. However, the discrete policy drives the continuous system which has different dynamics from the discrete system. Further, these continuous rollouts are deterministic since the action selection is now fixed, hence each trajectory is dynamically feasible and reproducible. The best solution often has better coordination among the agents and is often around 10% better than the average reward. As an example, Fig. 4.6(b) shows the statistics for the various rollouts (discrete average, continuous average, continuous best) for MCTS.

Table 4.2: parameters selected for Experiment 1 & 3: temperature-based reward

parameter	CEM	CEM h.d.	CEM ind.	CEMAS
α	0.4	0.5	0.3	0.7
ρ	0.25	0.2	0.15	0.05

parameter	MCTS	MCTS h.d.
C	0.03	0.03

The results for Experiment 1 show that the iterative greedy method gathers the most reward (Fig. 4.7(a)). The brute force method beats all of the policy based methods yet only takes 76% longer to compute than the slowest policy based method (CEM ind.). CEMAS searches in a dramatically larger

action space that is $15^3 = 3,375$ larger than the action space that the brute force method searches, yet only takes 2.2 times longer to compute than the brute force. It is noted that the run time of CEMAS is highly sensitive to tuning parameters, where setting $(\alpha, \rho) = (0.5, 0.15)$ results in run times that are 76% slower than selecting $(\alpha, \rho) = (0.7, 0.05)$ with a mere 2% difference in solution quality. Reducing $N_p = 5,000$ to $N_p = 2,500$ cuts down CEMAS's run time by more than half with less than $\approx 2\%$ decrease in solution quality.

Running CEMAS-2 alongside the brute force method shows how CEMAS-2 finds solutions within 98% of the brute force method, yet CEMAS-2 takes considerably longer to compute than the brute force method. In smaller search spaces, it is more likely that CEMAS will sample the same action sequence many times, wasting computational effort. This suggests that CEMAS is finding near optimal solutions in the much larger action space in which we cannot compute solutions via brute force reasonably (solutions could take as long as 27 days to compute).

In terms of policy based solvers, CEM with the shared Markov policy gathers the most reward, even beating CEM with individual Markov policies or a shared history dependent policies. This goes against the intuitions that history dependent rewards require policies that are history dependent. We suppose that having fewer parameters to tune in the policy enables the shared Markov policy to make most efficient use of experience.

In terms of daily running costs, the iterative greedy solver has an expected daily operating costs of around \$2,000 and offers a performance improvement

$\eta = 3.2\%$ over Sto.GA. computing costs for iter. greedy amount to \$0.13 per day while computing costs for Sto.GA around \$0.10 per day. On the other hand, using Sto.GA has a daily operating costs of \$2,064 due to reduced solution quality, making it rational to spend the extra computational time to run iterative greedy to generate the better solution.

4.8.2 Experiment 2

Training results in the first 3 regions for Experiment 2 suggested using the algorithm parameters in Table 4.3. Fig. 4.8 shows the resulting parameter sweeps for CEM with shared Markov Policies (4.6(a)) and MCTS with history dependent policies (4.6(b)). CEM experienced at most a 4.5% variation in reward during the parameter sweep, while MCTS experienced roughly 9.4% variation, implying that CEM is less sensitive to parameter selection than MCTS.

Table 4.3: parameters selected for Experiment 2: coverage-based reward

parameter	CEM	CEM h.d.	CEM ind.	CEMAS
α	0.3	0.5	0.3	0.7
ρ	0.2	0.2	0.15	0.05

parameter	MCTS	MCTS h.d.
C	0.05	0.05

When validating the algorithms, we note that CEMAP takes somewhere between 1-2 times longer to compute than MCTS. This is due to the increased cost of evaluating the reward of various discrete rollouts, which is the dominant run time cost. Unless it convergence criteria is satisfied, CEMAP eval-

uates at most 500,000 multi-history rollouts (100 iterations of 5,000 rollouts) per deployment scenario, while MCTS evaluates 300,000 (30,000 iterations per agent times 10 agents) multi-history rollouts per scenario, most of which have fewer than 10 agents and are therefore faster to compute on average per rollout.

Because the iterative greedy solver offers a performance improvement of $\eta = 8.1\%$ over Sto.GA, expected deployment costs are also reduced. Total daily operating costs for the iterative greedy algorithm are around \$2,000 while Sto.GA has a daily operating costs around \$2,162. Computing costs for iter. greedy are around \$0.29 per day while computing costs for Sto.GA are around \$0.12 per day. It is therefore rational to use the iterative greedy solver over Sto.GA and all other algorithms tested.

4.8.3 Experiment 3

The run time results of Experiment 3 show that CEM and CEMAS have run times that are linear in the number of agents, while brute force, greedy, iterative greedy, and Sto.GA have run times that are quadratic in the number of agents (Fig. 4.10). The sublinear dip at the end of CEM's time in Fig. 4.11 and CEMAS's time in Fig. 4.10(b) is most likely due to overcrowding agents resulting in more efficient batch updates.

The two biggest competitors for gathering the most reward in the large $N_{agents} = 128$ scenario are Sto.GA and the iterative greedy solution (Table 4.4). Iterative greedy gathers 7.2% more reward than Sto.GA on average.

We note that the variation in the reward gathered over all trials is heavily dependent on the deployment scenario, making use of standard deviations or error bars misleading. Instead, a pairwise comparison between how algorithms perform on an identical trial is required for more in depth analysis. Consider the fact that in all 25 scenarios tested iterative greedy found better solutions than Sto.GA. We further note that the iterative greedy method took about 4 times longer to compute. In terms of total deployment costs for 128 agents, the iterative greedy method has deployment costs of $25,613 (computing time costs around $13) while Sto.GA costs around $27,447 to deploy due to generating solutions that gather less reward. Using iterative greedy in place of Sto.GA therefore results in 6.7% cost savings in total deployment costs.

Table 4.4: Experiment 3 results for 128 agents

algorithm	avg. reward	std. reward	avg. time [s]
default	0.42	0.08	–
CEM	0.58	0.092	7,540
MCTS	0.50	0.091	37,305
CEMAS	0.67	0.10	47,000
bfm	0.68	0.098	75,000
i.greed	0.80	0.095	10,000
Sto.GA	0.74	0.090	2,730
greedy	0.67	0.082	443

4.9 Discussion

From the results in Experiments 1-3, we note that on average the proposed iterative greedy method in this paper finds better solutions than Sto.GA and results in cost savings despite taking longer to compute. However, the presented results use the greedy method to initialize Sto.GA. It is possible to use the iterative greedy method to initialize Sto.GA to further improve upon the iterative greedy solution resulting in a combined approach. The greedy solver and the iterative greedy solver exploit the heuristic that restricting the search to (sequentially) coordinated constant action sequences will better guide the search. This heuristic deserves greater treatment and merits further investigation.

We now wish to compare the general trends in the results of Experiments 1 & 2 to get a sense of how well the approaches perform relative to one another when changing the objective function. Some general statements are:

- the action based solvers outperform the policy based solvers. We note that this is in the absence of forecast uncertainty.
- CEMAS outperforms brute force due to searching in a more expressive search space.
- iterative greedy outperforms Sto.GA, CEMAS and the greedy method.
- CEM outperforms MCTS.

- CEM outperforms CEM h.d.

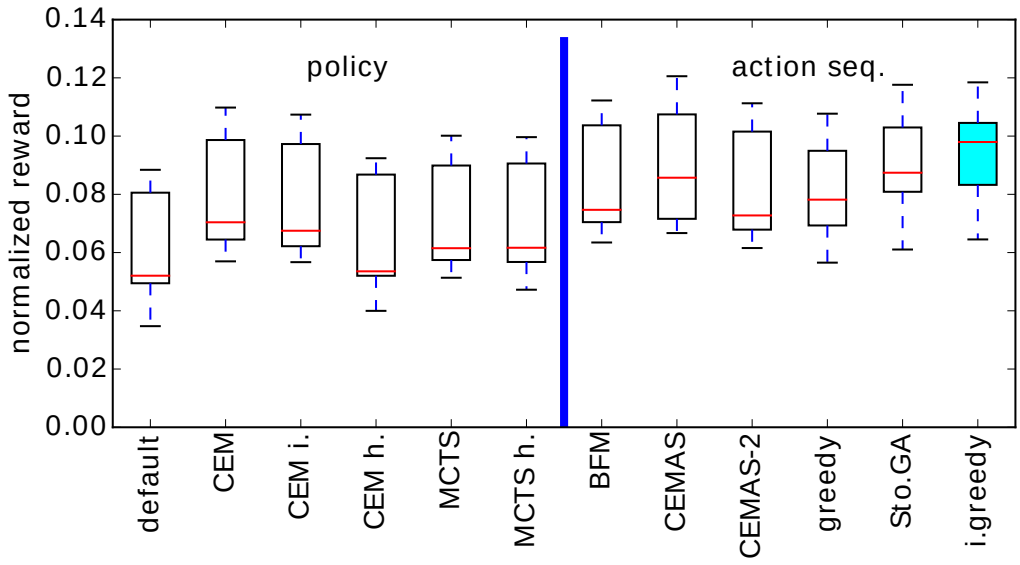
In terms of differences, Sto.GA's performance in Exp. 2 drops relative to iterative greedy, CEMAS and greedy. Sto.GA's reduced performance in Exp. 2 is perhaps due to the fact that the coverage reward function is not smooth, unlike the temperature-based reward which may be hindering accurate estimations of the gradient.

In terms of the longer run times between Experiment 1 and 2, evaluating the reward constitutes around 90% or more of the compute time for CEM. This is exacerbated by the cascaded union being more expensive to compute. While action sequence based approaches such as Sto.GA or the iterative greedy method beat policy based techniques, they do so in the absence of forecast uncertainty. Forecast uncertainty will significantly dominate positional uncertainty. It is often the case that simulators will predict features such as gyres, yet their exact locations will be off significantly resulting in correlated forecast errors. The only way to account for such errors is to utilize an ensemble of predictions. Generating policies may turn out to be the only approach that can handle such uncertainty.

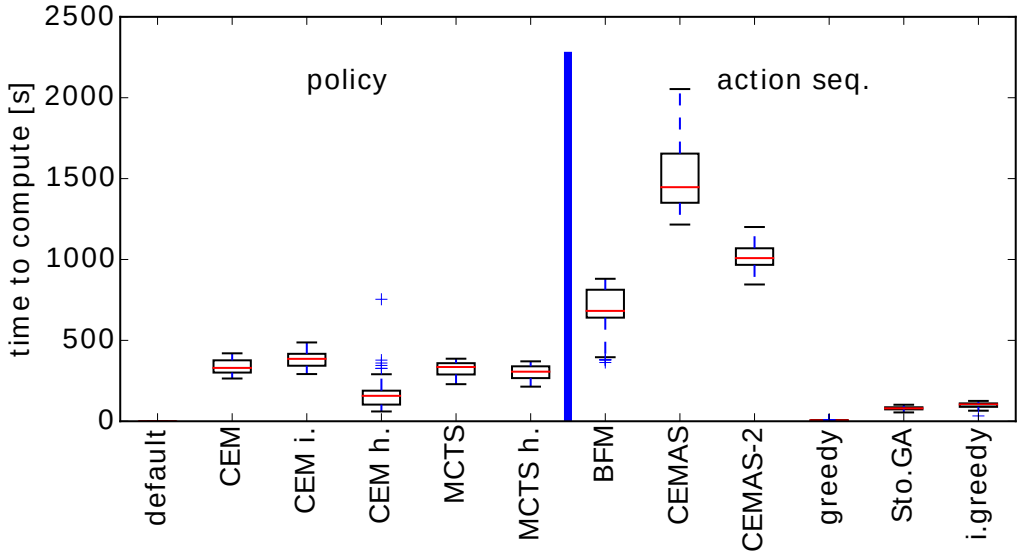
For Experiment 3, the major findings are that for the larger problems CEMAS and MCTS do not find solutions that are better than brute force method due to the exponentially increasing size of the solution space. Techniques that exploit the sequentially coordinated constant action sequence heuristic (greedy, Sto.GA, iter. greedy) find solutions that scale incredibly well to large problems.

4.10 Summary

We investigated two large scale geospatial tasks: adaptive sampling of temperature fields and coverage planning for teams of 8-128 underwater vehicles in large 300km by 300km environments while exploiting the ocean currents to maximize endurance. Generating feasible trajectories involves restricting the search either to the space of action sequences or in policies mapping states to actions. We benchmark six methods with different variations on their implementation and observe that the iterative greedy method offers the most cost effective solution outperforming Sto.GA by 3%-8% in the different experiments. The iterative greedy method restricts the search by employing the coordinated constant action sequence heuristic. Further, results show that action sequence based approaches outperform the policy based approaches, though this is in the absence of forecast uncertainty.

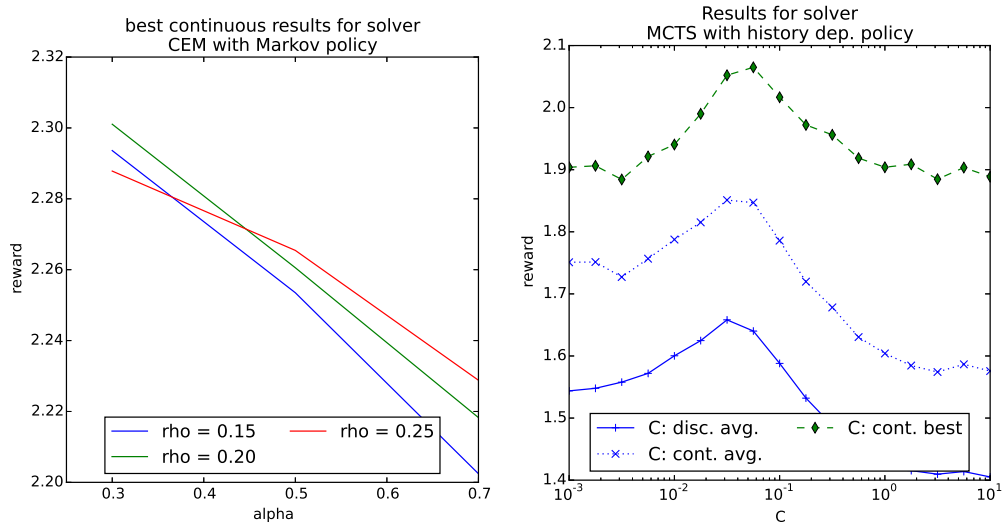


(a) information gathered for the benchmarked algorithms



(b) run times for the benchmarked algorithms

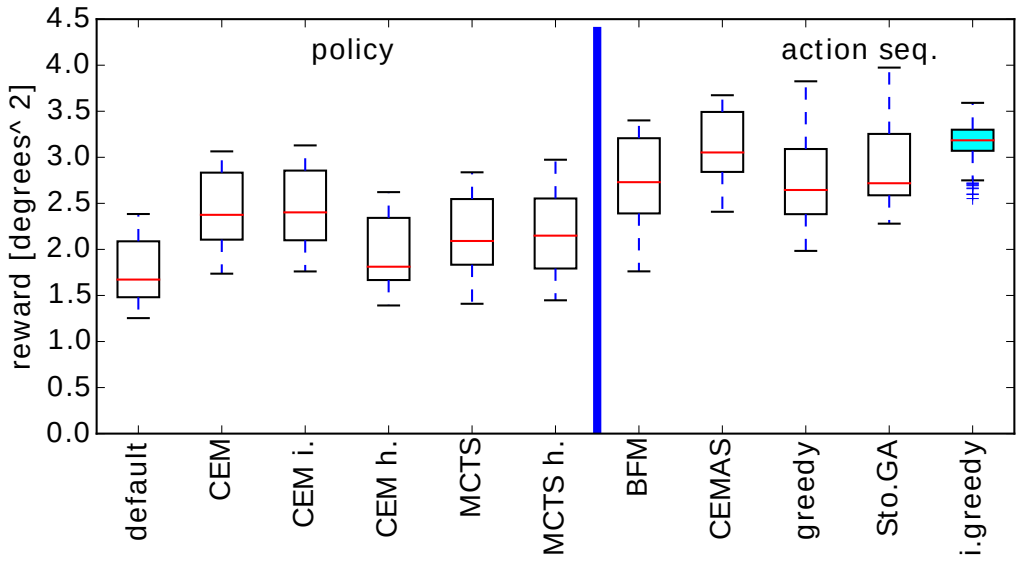
Figure 4.7: Validation Results for the benchmarked algorithms for Experiment 1.



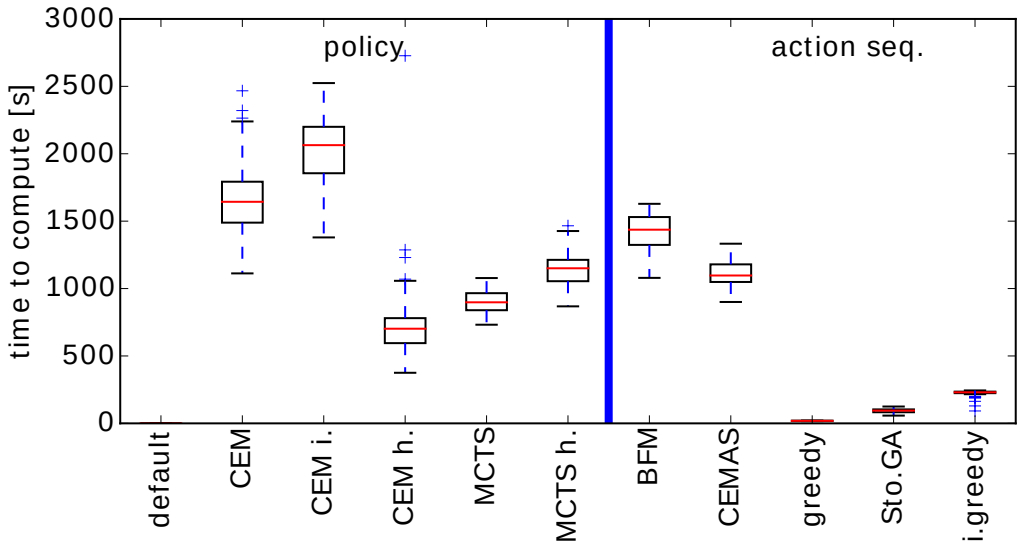
(a) Training results tuning parameters α, ρ for the Cross Entropy Method.

(b) Training results tuning parameter C for MCTS.

Figure 4.8: Training Results in Experiment 2 with the coverage reward. The performance of MCTS and CEM vary with respect to the choice of parameters and are tuned with training data to improve algorithm performance. Parameter selection for CEM and MCTS is domain specific, and no method for calculating the best parameters *a priori* is known. Despite CEM having more parameters, its performance is less sensitive to parameter selection than MCTS. Charts such as these are used to select parameters for validation runs and selected parameters are summarized in Table 4.3.

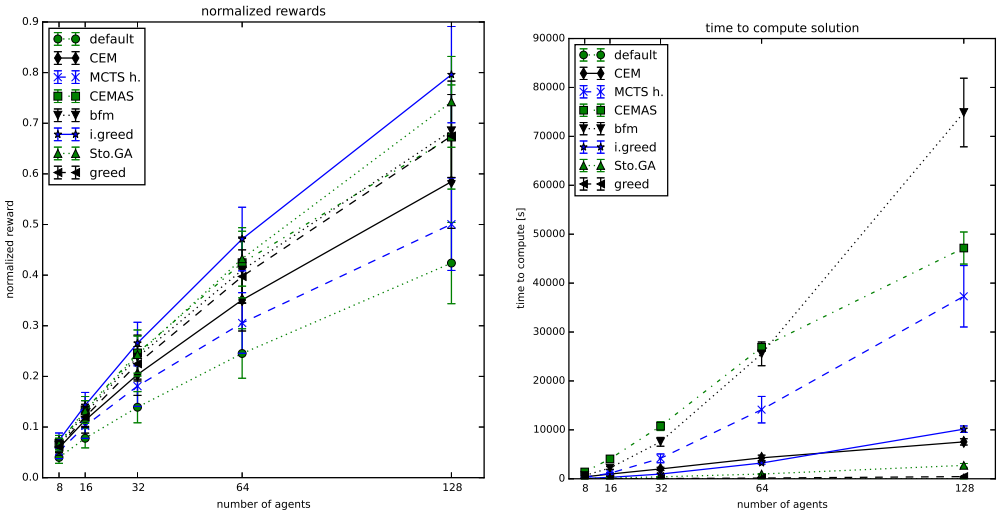


(a) information gathered for the benchmarked algorithms



(b) run times for the benchmarked algorithms

Figure 4.9: Validation Results for the benchmarked algorithms for Experiment 2.



(a) information gathered for the benchmarked algorithms

(b) run times for the benchmarked algorithms

Figure 4.10: Results for the benchmarked algorithms in Experiment 3.

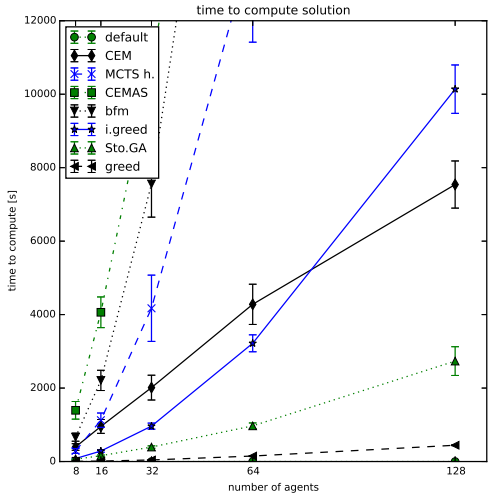


Figure 4.11: Run times for the benchmarked algorithms in Experiment 3, zoomed in.

Chapter 5: Persistent Monitoring with Teams of Autonomous Surface Vessels

Portions of this chapter are derived from work presented at the 2014 IEEE International Conference on Robotics and Automation [157].

5.1 Introduction

In this chapter, we investigate techniques for coordinating teams of autonomous surface vessels for persistent monitoring applications in a littoral environment. As an example application we consider the tracking of algae blooms by persistently monitoring areas in the ocean of a target temperature. One of the major challenges to increasing platform endurance is to limit propulsion capabilities and exploit the ocean currents. This however requires handling the difficulties of forecast uncertainty, which is an unresolved problem in the context of persistent monitoring tasks.

Generating routes or closed tours for vehicles during persistent monitoring tasks is similar to sentry patrolling tasks. For example, consider the use of an unmanned surface vehicle (USV) conducting harbor patrols to detect

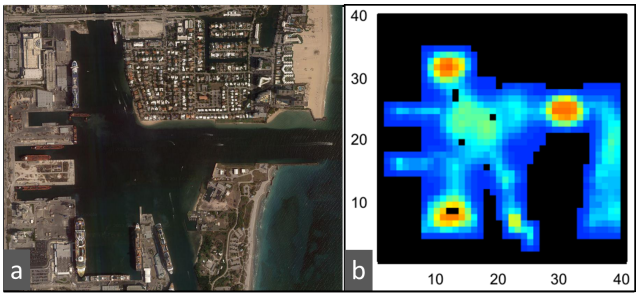


Figure 5.1: (a) An example of a harbor patrol environment with multiple entry points for intruders (A harbor in Hollywood, FL; source: Map data ©2013 Google). (b) Obstacle regions (black) and the "information value" map

.

intruders. It is reasonable to assume that possible intruders will enter the harbor (Fig. 5.1(a)) from certain locations such as harbor entrances and shipping channels. This suggests the use of an "information value map" (see Fig. 5.1(b)) that signifies how some regions are more dynamic or interesting and should be observed more often.

It is possible for static obstacles to exist in a littoral environment. Further, windy conditions or swift currents can contribute significant uncertainty to the USV's location and motion, compounding the problem. This suggests using a physics-aware planner that is capable of planning under motion uncertainty while avoiding obstacles in the environment. Many established techniques for coverage planning cannot account for either of these two challenges and must be accounted for in the given task. The goals of this topic are to:

- Investigate a Monte Carlo based technique for generating initial multi-

tours for teams of vehicles in the littoral environment.

- Extend previous work on Informative Coverage & Persistent Sensing for MDPs to be able to coordinate teams of vehicles for persistent monitoring applications.

- Develop a novel locally optimal greedy algorithm that has similar performance to other techniques but further reduces the expected number of agents lost during a deployment scenario.

- Benchmark the proposed algorithms against the state-of-the-art algorithms in littoral environments using high fidelity ocean forecast models.

We extend previous single-agent techniques to teams of agents, by combining locational optimization techniques (Voronoi Partitions) with a Markov Decision Process based formulation to account for motion uncertainty. We employ ocean forecast data from an Navy Coastal Ocean Model (NCOM) model to generate realistic motion models and benchmark several techniques in realistic deployment scenarios of teams of agents with limited propulsion capabilities. Results show that the novel solver reduces the expected number of agents lost and finds solutions with lower objective function cost for a modest increase in computational cost. The proposed approaches beat state-of-the-art persistent monitoring techniques that do not factor in the physics of the ocean environment.

We now describe the organization of the chapter. The problem formulation is in Section 5.2. Section 5.3 identifies already existing algorithms in use while Section 5.4 outlines our approach. Section 5.5 covers the simulated experiment setup. The results of the experiment are shared in Section 5.6 with a discussion of the findings in Section 5.7. Section 5.8 concludes the paper.

5.2 Problem Formulation

The problem formulation consists of a continuous space, the stochastic motion model (the Markov Decision Process), other environmental factors and the objective function. Let $c = \{(lat, lon)\} \in \mathcal{C} \subset \mathbb{R}^2$ be the continuous configuration space in which the environment is defined. Let $C_d = \{(lat_d, lon_d)\} \subset \mathcal{C}$ be the finite, discrete regular grid. The environment $\mathcal{E} = (\phi, \mathcal{O}, V_{lat}, V_{lon})$ where $\phi : X \to \mathbb{R}^+$ is the sensor function that assigns the density of information in each configuration. V_{lat} and V_{lon} are the ocean current data in the lateral and longitudinal directions respectively, defined on the grid C_d. $\mathcal{O}$ is an obstacle grid denoting locations of obstacles on C_d.

We now proceed to define the MDP or stochastic motion model of the system which uses the data in $\mathcal{E}$. This is the equivalent cost-based formulation of the rewards-based MDP formulation in Section 2.7, and the same stochastic motion model in Section 4.4.2. Consider that there are N agents, where agent i resides at the configuration $s^{(i)}$ in a finite configuration space

$\mathcal{S}^{(i)}$. Since all agents are homogeneous and collisions between agents can be safely ignored, all agents share a common motion model (state space, motion model, transition probabilities) with common state space $\mathcal{S}$. We note that each configuration $c_d \in C_d$ is assigned a unique state $S \in \mathcal{S}$ and we will use the notation of c_d and s interchangeably. Note that upper case S is a random variable, while lower case s is a realization. At each state s, the agent may select from a finite action set $a \in \mathcal{A}(s)$. The motion models of concern are Markovian systems, where the Markov property is satisfied:

$$\begin{aligned} &\Pr(S_{t+1} = s' | S_t = s_t, A_t = a_t) = \\ &\Pr(S_{t+1} = s' | S_t = s_t, \ldots, S_0 = s_0, A_t = a_t, \ldots, A_0 = a_0) \end{aligned} \tag{5.1}$$

The probability mass function $\mathcal{P}^a_{ss'} = \Pr(S_{t+1} = s' | S_t = s, A = a)$ defines the transition probabilities for the agent given the present state and action selected. Further, define the trap state s_T that captures all failure states (collisions) or leaving the boundaries of the environment. For any action, s_T transitions to itself with probability one.

The MDP's cost function $L(s_t, a, s_{t+1})$ defines the costs of executing various maneuvers on the vehicle. This is essentially the costs for fuel, amortized maintenance costs, and the value of any assets lost during a collision or leaving the mission boundaries. This is independent of any costs associated with (not) gathering information. Solving an MDP requires computing the value

function Φ which stores the cost to go from any discrete state to the goal. Φ satisfies the Bellman equation:

$$\Phi(s) = \min_{a \in \mathcal{A}} \mathbb{E}_{s_{t+1}}[L(s_t, a, s_{t+1}) + \Phi(s_{t+1})] \tag{5.2}$$

When computing value function Φ, the optimal actions are stored in policy $\pi : s \mapsto a$. Before proceeding to the objective function definition, we now define tours and multi-tours that define the persistent monitoring plan as a collection of waypoints, along with functions that define the next and previous waypoint in the respective agent's tour. We do not restrict whether or not the waypoints reside in the discrete space $\mathcal{S}^{(i)}$ or the continuous space.

A tour $\tau^{(i)} = (p_j)_{j=1}^{n_i}$ for $p_j \in C$ is a finite ordered sequence of n_i waypoints that generates a cycle or a closed path for agent i. Defining functions $next$ and $prev$ defines the connectedness of waypoints to each other. For a single agent, $next(p_j) = p_{j+1}$ if $j < n_i$ and $next(p_{n_i}) = p_1$ completes the tour. Similarly, $prev(p_j) = p_{j-1}$ if $j > 1$ with $prev(p_1) = p_{n_i}$ completing the tour in reverse.

A multi-tour $\mathcal{M}_{\tau,N} = (\tau^{(1)}, \tau^{(2)}, \ldots, \tau^{(N)})$ of N agents is a collection of N tours with one tour assigned to each agent. Agent i has n_i waypoints consisting of varying length. The waypoint sequences of each tour can be directly concatenated with a slight abuse in notation resulting in a single n-tuple of waypoints τ where $n = \sum_{i=1}^{N} n_i$. Care must be taken when defining $next$ and $prev$ for $\mathcal{M}_{\tau,N}$ to ensure that all tours are properly closed paths, however. If

agent i's tour consists of waypoints $\{p_j\}_{j=j_{i1}}^{j_{i2}}$ in $\mathcal{M}_{\tau,N}$, then $next(p_j) = p_{j+1}$ when $j_{i1} \leq j < j_{i2}$ and $next(p_{j_{i2}}) = p_{j_{i1}}$. Similarly $prev(p_j) = p_{j-1}$ when $j_{i1} < j \leq j_{i2}$ and $prev(p_{j_{i1}}) = p_{j_{i2}}$.

Let $\mathcal{H}$ be the single objective function defined in (Eq. 5.3):

$$\mathcal{H}(\mathcal{M}_{\tau,N}) = \sum_{j=1}^{n} \int_{V_j} \frac{W_s}{2} ||q - p_j||^2 \phi(q) dq + \sum_{j=1}^{n} \frac{W_g}{2} [(\Phi_{prev(p_j)}(p_j))^2 + (\Phi_{next(p_j)}(p_j))^2] \tag{5.3}$$

The goal is to find the multi-tour $\mathcal{M}_{\tau,N}$ that minimizes $\mathcal{H}$, (Eq. 5.4):

$$\mathcal{M}^*_{\tau,N} = \arg\min \mathcal{H}(\mathcal{M}_{\tau,N}) \tag{5.4}$$

Here, W_s and W_g reflect the trade off between the competing objectives of steering waypoints to informative regions vs. reducing path length. Further, V_j is the Voronoi partition of waypoint p_j. Voronoi partitions have been frequently used in similar locational optimization problems [56, 62, 65]. (Eq. 5.4) is similar to the Lyapunov-like function candidate found in previous work, excluding the adaptation parameter [66]. Also, note that values of $\mathcal{H}$ in (Eq. 5.3) are sensitive to the number of waypoints in the objective function.

5.2.1 Sensor Function

The sensor function ϕ is mission specific. Missions intending to sample locations based on temperature include tracking algae blooms, which often favor a specific range of temperature conditions [18]. Supposing there is some desired temperature T_0, we will use the same reward function in Section 4.3.1.2 that favors sampling locations nearest T_0, where T is the temperature at configuration c:

$$I(c) = \begin{cases} e^{-a_p(T-T_0)} & \text{if } T \geq T_0 \\ e^{-a_n(T_0-T)} & \text{if } T < T_0 \end{cases}$$

a_p is a scale factor for the positive case and a_n is a scale factor for the negative. Letting $a_p \neq a_n$ allows one to ensure that the region $\{c : I(c) \geq b\}$ for $b \in (0, 1)$ is a guaranteed fraction of the environment, η. Decreasing η increases the difficulty of the problem by having fewer locations of higher reward.

5.3 Existing Algorithms

We now outline several existing algorithms that are used in part to solve the problem outlines in Section 5.2. These include using Value Iteration to solve for value functions for MDPs, and algorithms based on Lloyd's algorithm for optimizing waypoint locations for objective function $\mathcal{H}$ defined in (Eq. 5.3).

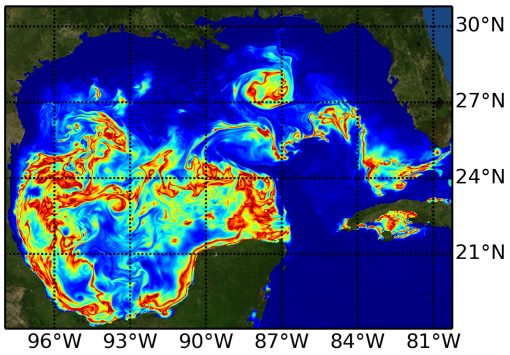


Figure 5.2: The example Gulf of Mexico Environment with the plotted temperature function $I(c)$ with $T_0 = 26.3°\text{C}$.

5.3.1 Solving the MDP

Due to intrinsic motion uncertainty, solving the MDP requires generating feedback policies. Computing policies is performed by employing dynamic programming principles by generating a (globally optimal) value function Φ satisfying (Eq. 5.2), which the policy π greedily descends. Define a goal state s_{goal} (or collection of goal states). A goal state is similar to the trap state, but no additional cost is incurred by traversing to the goal location. Further the value function is set to zero for all goal states and to arbitrarily high cost for other states.

To solve for the policy $\pi_{s_{goal}}$ and value function $\Phi_{s_{goal}}$, use Probabilistic Backwards Value Iteration (PBVI or VI) [67]. To speed up VI, we employ Gauss-Seidel backups [73]. Precomputing state transition probabilities and storing them in a look up table (LUT) further speeds up calculations by

eliminating repeated calculations and doing loop unrolling.

Noting the presence of swift currents, it is possible for states that do not contain obstacles to be states of imminent collision, where traversing through the state will result in collision with probability one resulting in prohibitively high cost in the value function. See Fig. 5.3 for a simplified example of imminent collision states. VI is particularly suited to discovering such areas, but this is goal dependent, especially when there are disjoint safe regions of the environment. The reachability of a state s_{goal} from another state s can be quickly determined by inspecting whether or not the value $\Phi_{s_{goal}}(s)$ is less than the cost of a collision or a lost asset. One major difficulty to the problem at hand is being able to reliably identify such regions of imminent collision. We denote a region or collection of (connected) states safe if the agent can eventually traverse between any 2 states in the set with high probability.

5.3.2 Techniques based on Lloyd's algorithm

Locational Optimization based techniques optimize locations of assets given a density map based on Voronoi Partitions. Such problems are computationally intractable to solve globally in practice, so gradient based techniques are used. One such technique is to use Lloyd's algorithm [64] to generate steering laws for teams of vehicles to adequately sense an environment with a nonuniform distribution of information [56].

Previous work [66] extends such techniques to that of persistent monitoring problems, and includes adaptive control laws to estimate the sensor

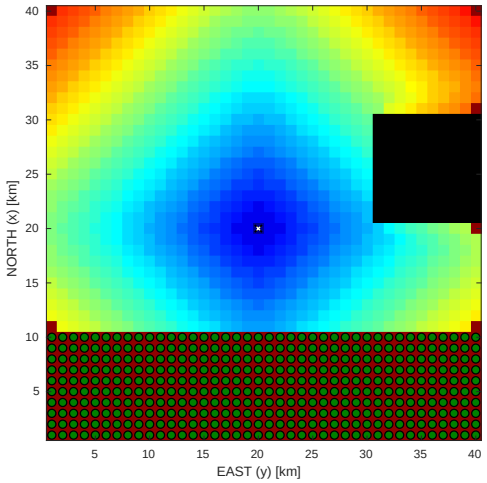


Figure 5.3: Example simplified environment illustrating the concept of imminent collision states. The agent can move in the cardinal directions of one cell (1km) per time step with no motion uncertainty. The currents are zero except in the bottom quarter of the space (denoted by green circles), where the currents move downward at two cells per time step, overpowering the agent's control authority. Black indicates physical obstacles. The colored background (color map jet) is the value function or the cost to go to the goal location at (20,20), with blue indicating low cost and maroon indicating arbitrarily high cost. While there is no physical obstacles present, the agent will be swept out of the environment with probability one if it enters the swift currents, generating imminent collision states.

function ϕ. We ignore the challenge of estimating ϕ in the problem at hand, since temperature maps are available from ocean forecasts. In the absence of environmental effects (ocean currents, obstacles, etc), the objective function (Eq. 5.3) is used with $\Phi_{p_k}(p_j) = ||p_j - p_k||$ being the arc length. Employing gradient descent on the objective function results in the following waypoint steering policy:

$$\dot{p}_j = u_j = \frac{K_j}{\beta_j}[W_s M_{V_j} e_j + W_g(\Phi_{prev(p_j)}(p_j))(prev(p_j) - p_j) \qquad (5.5)$$
$$+ W_g(\Phi_{next(p_j)}(p_j))(next(p_j) - p_j)]$$

Here, $M_{V_j} = \int_{V_j} W_s \phi(q) dq$ is the mass of the Voronoi partition V_j using the sensor function ϕ as the density function. $e_j = C_{V_j} - p_j$ is the vector difference of the Voronoi partition centroid and the waypoint p_j. $C_{V_j} = \frac{L_{V_j}}{M_{V_j}}$, where $L_{V_j} = \int_{V_j} W_s q \phi(q) dq$ is the first moment. The integrals defined over the Voronoi partitions (e.g., M_{V_j} and C_{V_j}) are well approximated by the Riemann sum of points on the regular grid C_d. K_j is a positive gain constant. Also note that $\beta_j = M_{V_j} + 2W_g$ is a normalization parameter.

Although the algorithm by Soltero does not factor in the effects of environment dynamics and static obstacles, a straightforward repair mechanism can be implemented. Any waypoint that is in a collision state can be moved to a neighboring location that isn't a collision state. While a line search can be conducted, we propose conducting a local search to move the waypoint to a neighbor with the lowest cost.

We now consider different algorithms that solve related tasks but are unsuitable for this task. Generating multi-tours of a large number of waypoints resides in a vast solution space. For example, the small problem of selecting 8 waypoints on an 8×8 grid for a single agent yields 10^{14} combinations! Consider the following techniques that are not suitable for this problem:

- Brute force enumeration techniques are infeasible due to solution space size, also making things similarly difficult for global solvers using search based techniques.

- A sequential allocation trick where individual waypoints are added to the multi-tour are added one at a time in the way that reduces objective costs the most would dramatically limit the solution space, and could potentially offer bounded performance losses. However, extending the formulation to handle the MDP motion model will require solving for the value function for each location being considered, which would be prohibitively expensive to compute.

- Solvers for the M-TSP problem do not offer a complete solution to the problem, but are good at optimally connecting pre-selected waypoints to generate tours. Rather, they will be incorporated with other techniques, namely Monte Carlo.

- Convex optimization techniques are not suitable when obstacles are introduced into the problem.

5.4 Approach

We now describe the approach we will take at solving the objective in (Eq. 5.4). We start off with two distinct means of developing initial plans. The first technique is for a single agent and uses Soltero's algorithm to generate an

initial tour ignoring obstacles. However, this technique becomes impractical when extending to more realistic scenarios for multiple agents. We use Monte Carlo to generate initial multi-tours. We then illustrate a means of generalizing the Lloyd's based algorithm to factor path traversal costs generated from a MDP model. We also introduce a novel greedy method and discuss how to compute the expected loss in agents when executing a multi-tour.

When executing the coverage plan τ on the vehicles, each vehicle will start at its initial waypoint $p_{j_{i1}}$, and using the feedback plan $\pi_{next(p_{j_{i1}})}$. Once the vehicle reaches the goal state then the planner follows the next waypoint feedback plan to move on to the next waypoint $\pi_{next^2(p_{j_{i1}})}$.

5.4.1 Initial Tour Selection

Before employing the proposed methods in the paper, an initial plan for the team of agents must be generated. Employing any of the locally optimizing techniques without first generating an initial tour results in a rubber-band like contraction of the waypoints wrapping around obstacles, with "attractor springs" pulling waypoints to the informative regions. Our initial approach to this was to first ignore obstacles, [157] simply ignoring obstacles can be insufficient for generating reasonable plans for the multiagent case. This is still an open research question in its own right, but we introduce a Monte Carlo based approach with several heuristics to find good initial policies to generate initial multi-tours. One sensible approach to generate the initial multi-tour is to randomly sample candidate waypoints, then assign them to

individual agents, which is equivalent to the Multiple Traveling Salesman Problem (M-TSP). A probability distribution of cells is generated proportional to the information content of each cell. Cells with swift currents or that contain obstacles have their sample probabilities set to zero. We use the Genetic Algorithm based solver MTSP_GA for MATLAB to approximately solve the M-TSP problem [189]. Traversal costs between waypoints for M-TSP are generated by solving for the value function of each waypoint. Many sets of candidates are sampled randomly, the M-TSP problem is solved and the solution with the lowest objective function cost is stored.

One limitation to the current approach is that it is still possible for sampled waypoints to be in cells that are in imminent collision. Including these imminent collision states significantly hampers the Monte Carlo approach in finding collision free multi-tours. Consider the example scenario in Fig. 5.4. To sidestep this, one can in principle determine which states are truly imminent collision states by looking at every state's value function. While entering an imminent collision state eventually results in a collision with probability one, the reachability graph of an imminent collision state can contain states that are "downstream". However, this requires computing the value function of every state in the environment which is impractical, so we approximate this with Monte Carlo. After 5 iterations of the Monte Carlo method with 15 waypoints, up to 75 unique value functions have been computed. We then suppose that if a state cannot reach more than a small fraction (μ = 10-33%) of the goal states, then the state is assumed to be an imminent

collision state and has its sample probability set to zero. While this tends to overestimate how many states are imminent collision states, this works well in practice by eliminating difficult to reach locations. Having nonzero μ will eliminate the inclusion of imminent collision states that are upstream of other waypoints that are also imminent collision states. Note that having multiple disconnected safe regions will exacerbate this problem.

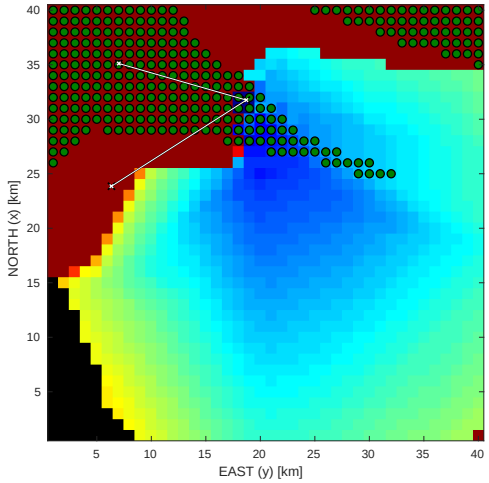


Figure 5.4: This is an example ocean environment denoting imminent states of collision for a given waypoint by plotting the value function. Note that the regions of swift currents (green) do not always correspond with states that result in collision with high probability (maroon). In this example, the goal waypoint at (32,19) (blue) is inside the swift current region yet is still reachable. However, reaching the neighboring waypoints from the goal waypoint is impossible, meaning that the candidate tour cannot be completed successfully (not shown).

5.4.2 ICPS-MDP: a Lloyd's algorithm based solver

One major limitation to such techniques that use the arc length between waypoints in the cost function (Section 5.3.2) do not account for the presence of obstacles, swift currents, or other deployment costs. In our initial study, we generalized the objective function to use path traversal costs generated by dynamic programming, as illustrated in (Eq. 5.3) [157]. However, in order to extend gradient based techniques, the gradient of the value function must be evaluated

$$\begin{aligned}\dot{p}_j = u_j = \frac{K_j}{\beta_j}[W_s M_{V_j} e_j + W_g(\Phi_{prev(p_j)}(p_j))\hat{h}_{prev(p_j)}(p_j) \\ + W_g(\Phi_{next(p_j)}(p_j))\hat{h}_{next(p_j)}(p_j)]\end{aligned} \tag{5.6}$$

When computing $\frac{\partial}{\partial p_j}(\Phi_{p_l}(p_j))^2$ in (5.6), it is assumed that the subscript p_l is fixed and hence $\frac{\partial}{\partial p_j}(\Phi_{p_l}(p_j))^2 = (2\Phi_{p_l}(p_j))\nabla\Phi_{p_l}(p_j)$. Also, we assume $\frac{\partial}{\partial p_l}(\Phi_{p_l}(p_j)) = 0$ which explains the use of two terms in (5.4). The gradient $\nabla\Phi_{p_l}$ is not well defined since the domain of Φ is C_d. Instead of using gradient descent, we will define a descent direction $\hat{h}$. It is reasonable to use either the expected value (Eq. 5.7) by averaging over all possible outcomes in $\mathcal{P}^a_{ss'}$ when executing the feedback plan. It is also reasonable to conduct a local search around s to minimize the value function (Eq. 5.8), ignoring the motion model since waypoint maneuvers do not need to reflect agent dynamics:

$$\hat{h}(s) := \hat{h}_{mean}(s) = \mathbb{E}[s_{t+1}|s_t, \pi(s)] \tag{5.7}$$

$$\hat{h}(s) := \hat{h}_{ls}(s) = (s' - s), s' = \underset{s' \in Neighbors(s)}{\arg\min} \Phi(s') \tag{5.8}$$

where $Neighbors(s)$ denotes the local neighborhood of state s, and the proper Φ for (Eq. 5.8) is determined by context (the Φ the respective $\hat{h}$ operator is adjacent to). We note that (Eq. 5.7) can be poorly defined in regions of imminent collision, often pushing waypoints deeper into region instead of pushing them out. Due to the existence of regions with prohibitively high cost (due to e.g. high probability and high cost of collision), and due to the discrete time implementation, it is possible for the gradient of the policy to become very large. With maximum desired control u_{max}, define normalized control $\upsilon_j = u_{max} \tanh(u_j/u_{max})$.

Algorithm 13 ICPS-MDP(τ, ϕ)

Require: An informative coverage path τ and a sensor function ϕ.
Ensure: An informative coverage path τ.

```
while ||u|| > ε do
    for all p_j ∈ τ do
        Compute Voronoi-like partition V_j using k-nearest-neighbor (KNN) algorithm for p_j.
        Run PBVI to generate (Φ_{p_j}, π_{p_j}).
        Integrate the system dynamics of ṗ_j = u_j = f(p_j) using (5.6) and p_{k+1,j} = p_{k,j} + u_{k,j} · Δt_k
    end for
    k ← k + 1
end while
return τ
```

5.4.3 Locally Optimal Greedy Algorithm

The hybrid approach described in Section 5.4.2 efficiently coordinates joint movements of all waypoints, but it requires an approximation of the gradient of the computed value functions, and must account for various difficulties in combining discrete and continuous approaches. One major limitation to the definition of the gradient is that in swift currents, the gradient will push waypoints into regions of imminent collision.

Conceptually, it is desirable to restrict waypoint movement along directions that are feasible and don't cause collisions, effectively restricting the gradient based solvers into a constrained optimization problem. Alas, defining these constraints during joint maneuvers of waypoints increases the computational burden of the algorithm. It is possible to instead design a discrete algorithm that moves one waypoint at a time greedily minimizing the objective function. Consider an algorithm that iterates through each waypoint, and perturbs the waypoint's location in multiple directions and selects the direction that minimizes the objective function the most. Considering only individual waypoint moves eliminates the combinatorial complexity of making joint decisions for waypoints.

To simplify the presentation of the Locally Optimal Greedy algorithm (L'OGRE: *l'ogre* is French for "the ogre"), Alg. 14 assumes that the waypoints that constitute the tour remain in C_d. This allows one to easily index computed value functions on the goal state and store them in a look up

table for fast look-up in subsequent iterations (not shown in Alg. 14). Assume existence of a function $Neighbors(s)$ that returns the local neighbors adjacent to state s. To simplify the algorithm's presentation assume that $s \in Neighbors(s)$. Letting $\tau = (s_j)_{j=1}^n$ enables the notational shorthand where updating the value of s_j updates the jth waypoint in τ.

Algorithm 14 L'OGRE: Locally optimal greedy algorithm

Require: initial configuration τ_0, $\mathcal{H}$, functions $next$,$prev$,$Neighbors$

$k \leftarrow 1$
$\tau \leftarrow \tau_0$
$\mathcal{H}_{min} \leftarrow \mathcal{H}(\tau)$
$\mathcal{H}_{prev} \leftarrow \infty$
while $\mathcal{H}_{prev} \neq \mathcal{H}_{min}$ **do**
  $k \leftarrow k + 1$
  $\mathcal{H}_{prev} \leftarrow \mathcal{H}_{min}$
  for $s_j \in \tau$ **do** {optimize individual waypoints in tour}
    $h \leftarrow \infty$
    $s \leftarrow s_j$
    for $n_l \in Neighbors(s_j)$ **do** {iterate over neighbors}
      $s_j \leftarrow n_l$
      $h \leftarrow \mathcal{H}(\tau)$ {evaluate cost after moving s_j to one of its neighbors}
      if $h < \mathcal{H}_{min}$ **then** {update best known solution}
        $s \leftarrow n_l$
        $\mathcal{H}_{min} \leftarrow h$
      end if
    end for
    $s_j \leftarrow s$ {update $s_j \in \tau$ greedily}
  end for
end while
return $(\tau_k, \mathcal{H}_{min})$

We now investigate the termination conditions of the algorithm. Define the set $T(\tau) = \{(\sigma_j)_{j=1}^n | k \in \{1, 2, \ldots, n\}, \sigma_j = s_j, j \neq k, \sigma_k \in Neighbors(s_k)\}$.

$T(\tau)$ is the set of all tours near τ where at most one waypoint is perturbed to one of its neighbors. A local minimizer $x^* \in \mathcal{X}$ of $f(x)$ in local neighborhood $S \subseteq \mathcal{X}$ is such that $f(x^*) \leq f(x), \forall x \in S$.

Theorem 4 *Alg 14 terminates. Further, it will terminate when it finds a solution $\tilde{\tau}$ that is a local minimizer of $T(\tilde{\tau})$*

Proof: This can be proven using contradiction. Because the problem resides in a finite space, one can in principle enumerate over all candidate solutions and sort them based on cost. Therefore there is a global minimizer, and at least one local minimizer. If the algorithm never terminates, there must exist an infinite sequence of $(\mathcal{H}_{min,k})$ for iteration k that are strictly decreasing. In addition, it must visit at least one τ infinitely often by means of the pigeonhole principle (placing infinite pigeons into a finite number of pigeonholes), with continually decreasing cost. This is impossible. Therefore the algorithm terminates.

Alg. 14 iterates over the elements in τ's local neighborhood $T(\tau)$ in Line 8 by construction. When the algorithm terminates at Line 5, then $\mathcal{H}_{min}$ did not change between iterations, therefore τ did not change between iterations. It searched $T(\tilde{\tau})$ in Line 8 during the last iteration concluding that $\tilde{\tau}$ was a local minimizer by exhaustively searching $T(\tilde{\tau})$.

□

It is noted that the neighborhood $T(\tau)$ is indeed small. One might prefer that the neighborhood include tours that jointly perturb all waypoints simul-

taneously, but this will dramatically increase the search space at each iteration, exacerbating the algorithm's speed and is perhaps unnecessary given the following reason. The form of $\mathcal{H}$ ensures that moving waypoints only introduces local changes in the objective function, since a waypoint's Voronoi Partition only changes if it or another waypoint that it shares an edge with changes. Reasoning over joint decisions of moving two waypoints that are far away from each other will have little impact on the algorithm's performance. We find that in practice the greedy method continues over many iterations in Line 5 so the effective basin of attraction of the local minimizer is much larger than the local neighborhood.

5.4.4 Calculating Expected Agent Loss

Calculating the expected number of agents lost when enacting a plan is non-trivial because of the motion uncertainty, as one must average over an entire distribution of possible agent trajectories resulting in success or failure. However, computing the success probability of each agent is quite feasible in the following manner. $P^{(i)}_{succ,(j)}$ is the probability of reaching $next(p_j)$ from p_j without collision and P_{succ} is the probability of successfully executing the entire tour $\tau^{(i)}$, i.e., $P^{(i)}_{succ} = \prod_{j=1}^{n_i} P^{(i)}_{succ,(j)}$. The expected number of agents lost due to collisions or model uncertainty is the sum of the probabilities of the individual agents being lost: $\mathrm{E}[N_{fail}] = \sum_{i=1}^{N}(1 - P^{(i)}_{succ})$.

Now the task is to compute the probability of success when a single agent is maneuvering from one waypoint to the next waypoint $P^{(i)}_{succ,(j)}$. The two

motion models that are involved in this calculation are the planning model $\tilde{\mathcal{P}}^a_{ss'}$ and the ground truth model $\mathcal{P}^a_{ss'}$. When the solvers employ VI, the solver computes optimal policy π mapping under model $\tilde{\mathcal{P}}^a_{ss'}$. A Markov Chain (MC) can be constructed by defining state transition matrix $T = [T_{i,j}]$ from $\mathcal{P}^a_{ss'}$ with $T_{i,j} = \Pr[S_{t+1} = j | S_t = i]$ under the policy π. We wish to find the stationary distribution Y of T satisfying $Y = YT$ with $Y^{(l)} = \Pr[S = l]$ as $t \to \infty$. This allows one to compute $\Pr[S = s_T]$, the probability of failing the objective of getting to the next waypoint. We solve for stationary distribution Y numerically by executing the fixed point method $Y_{k+1} = Y_k T$ until it converges (T is sparse). Note that Y exists but is not necessarily unique. However, we are most concerned with the limiting distribution Y when Y_0 is a delta distribution centered at the previous waypoint.

5.5 Experimental Setup

For this work, we concern ourselves with multi-agent problems using stochastic motion models defined using high fidelity ocean forecast models in a littoral environment. One important performance metric previously unspecified in the informative path planning literature is the probability of path execution success (conversely, expected number of agents lost during plan execution for the multi-agent case) while executing the feedback plans to traverse between waypoints. For each simulation, we compare the vehicle's probability of success while executing the specified coverage plan. Other per-

formance metrics of interest are the cost of the final solution and the time to compute the solution.

We extend our work to a more realistic setting for multiple agents. We use the Navy Coastal Ocean Model (NCOM) to model the ocean forecast for the Gulf of Mexico (GoM), contained in a bounding box spanning latitudes 18°N-30.8°N and longitudes 80.25°W - 98.0°W [186]. This is equivalent to the region defined in Fig. 5.2. The NCOM model describes ocean current velocity, temperature and salinity forecasts at 72 depths, at 1km resolution in latitude and longitude, and 3hr temporal resolution for 7 days. However, since this work considers ASVs, only the surface conditions are used in planning. Further, for initial results, we assume that the currents are static throughout the duration of the mission.

We benchmark the various algorithms in a wide range of operating conditions by extracting 32 regions that are 40km x40km from the GoM data set. Regions must adhere to the following criteria:

- the speed of the ocean currents in half the cells cannot exceed the average speed v_{max}
- 2%-20% of the environment must be occupied by obstacles
- the total information content must exceed 10 units

The mentioned criteria were selected to ensure that there was sufficient area for the vehicles to maneuver safely with a rich enough distribution of

obstacles and swift currents. Finally, the minimum information content requirement ensured that there was an interesting enough distribution of information to gather in the environment.

A total of 15 waypoints were selected for each scenario and a total of four tours must be selected for four distinct agents.

5.5.1 ASV Stochastic Motion Model

The maximum thrust of the agents $v_{max} = 0.44\frac{\text{m}}{\text{s}}$ is set to the average current speed in the entire Gulf of Mexico for the given data set. This implies that there are large contiguous regions of the ocean that contain swift currents that can overpower the agent's control authority. Define the set of thrust velocities $v_{thrust} \in \{(0,0),(v_{max},0),(-v_{max},0),(0,v_{max}),(0,-v_{max})\}$ in the cardinal directions. Thrust commands last for Δt =3 hours, which is sufficient time to move 1-2km in the absence of any ocean currents.

Given configuration $c_t \in C_d$ at time t with velocity v_{thrust}, define net velocity $v_t = v_{thrust} + v$ where v includes the lateral and longitudinal components of the ocean currents from (V_{lat}, V_{lon}) for a given configuration. In Universal Transverse Mercator (UTM, i.e. Euclidean) coordinates, the update equations are: $c_{t+1} = c_t + \Delta t \cdot v_t$. and can be readily extended to coordinates (lat, lon) on the earth. Note that for our experiment, we employ use of static current fields, though the MDP formulation extends to time varying current fields.

The stochastic motion model uses 1-km cells, with 5 particles to approx-

imate the transition probabilities given the state and action to be simulated. We note that the spatial resolution of our NCOM model is limited to 1-km / 3hr time duration, tracking particles does not offer a good approximation of the motion uncertainty. In the absence of ensemble based predictions, we instead introduce motion uncertainty into the planner's motion model by manually perturbing outputs $\tilde{c}_{t+1} = c_{t+1} + \delta_{err} \cdot \rho + d_{drift}$ by $\delta_{err} = 1$ and $\rho \in \{(0,0),(1,0),(-1,0),(0,1),(0,-1)\}$ km in the cardinal directions with $d_{drift} = (0,0)$. Each c_d cell on the discrete grid is assigned a discrete state S, any coordinate c_t that transitions to the cell is assigned the state S. The MDP motion model is computed in Python and stored in a look up table that is then queried by the solvers. This also significantly speeds up the code by precomputing state transition probabilities and doing loop unrolling.

To observe how robust the solvers are to model uncertainty, we wish to plan with the stochastic model $\tilde{\mathcal{P}}^a_{ss'}$ and observe system performance when the true system dynamics are sampled from $\mathcal{P}^a_{ss'}$. This is accomplished by adding noise to the simulator. For each state S, sample $\delta_{err} \sim Unif(2/3, 3/2)$ and drift $d_{drift} \sim \mathcal{N}(0, \sigma_{drift})$ with $\sigma_{drift} = 0.25\frac{\text{km}}{\text{h}}$. Combining these two random processes with the mentioned parameters into the motion model cause about 50% of all (state,action) outcome distributions to differ between the planning model to the ground truth model.

5.5.2 Algorithm Configurations

For the objective function, we set $W_g = 50$. The ICPS-MDP algorithm runs for 100 iterations. In terms of selecting the gradient, ICPS-MDP is configured to run with either $\hat{h}_{mean}(s)$ (ICPS-MDP-mean), or $\hat{h}_{ls}(s)$ (ICPS-MDP-ls). The set $Neighbors(s)$ for L'OGRE constitutes the 4-connected neighbors in the cardinal directions and s. On the other hand, $Neighbors(s)$ consists of all 8 connected neighbors and itself.

5.6 Results

Example results for four different environments are shown for the initial Monte Carlo solution (Fig. 5.6), for L'OGRE (Fig. 5.7) and ICPS-MDP-mean (Fig. 5.8). Fig. 5.9 shows the box plots of the results for time to compute, objective function values, and the expected number of agents lost. Median results and other metrics are summarized in Table 5.1.

Table 5.1: Results in the 32 environments. Except for the failure rates, median values amongst the 32 environments are posted. Box plots of the same data are available in Fig. 5.9.

algorithm	$\bar{\mathcal{H}}$	$t_{compute}[s]$	R_{fail}	$\mathrm{E}[N_{fail}]$
Initial MC	$5.1 \cdot 10^5$	–	–	0.20
Soltero	$4.9 \cdot 10^5$	4.9	0.375	0.11
ICPS-MDP-mean	$4.1 \cdot 10^5$	41.4	0.28	0.092
ICPS-MDP-ls	$5.5 \cdot 10^5$	17.9	0.4	0.093
L'OGRE	$3.6 \cdot 10^5$	45.3	0.031	0.065

An algorithm is said to fail a trial if it is unable to improve the initial way-

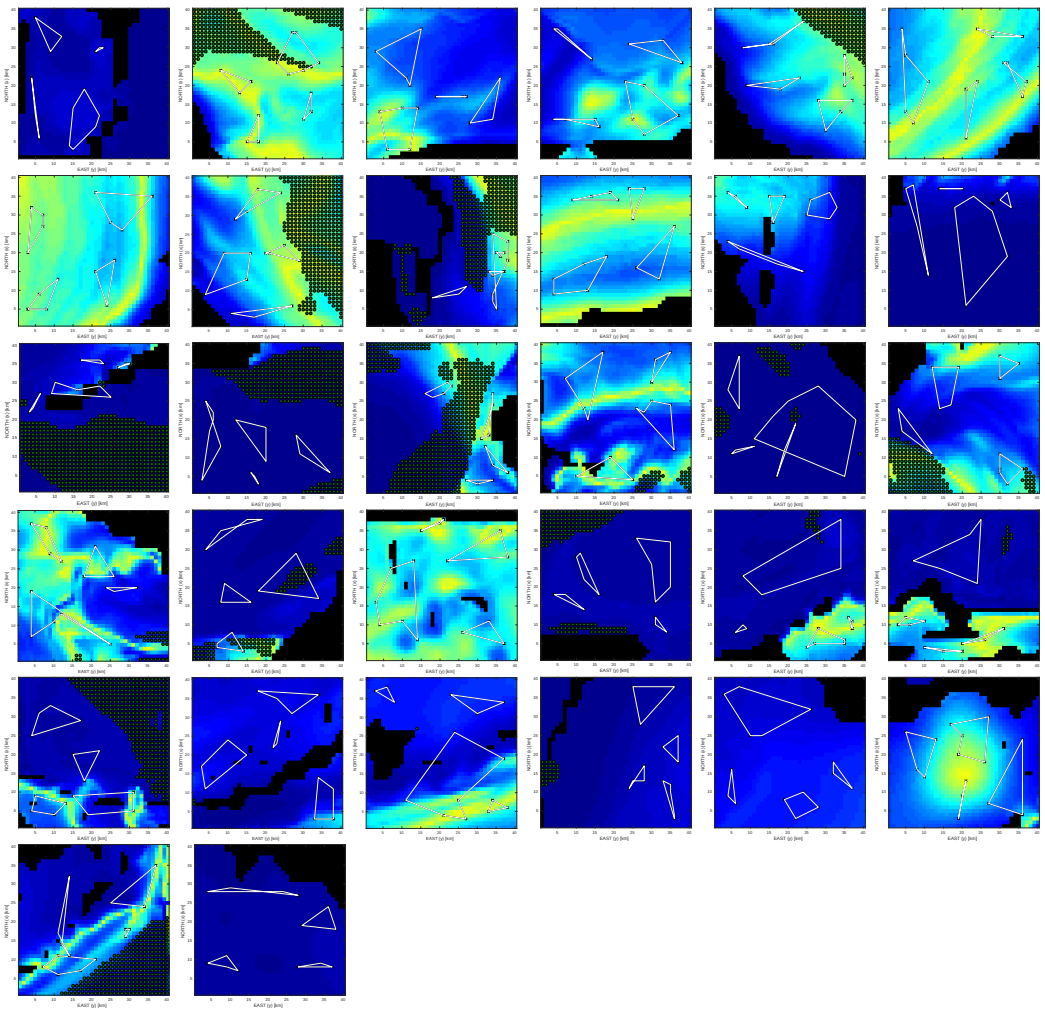


Figure 5.5: The 32 test environments. Initial Monte Carlo generated multi-tours are also shown.

point configuration generated by Monte Carlo. This is often the case when an algorithm moves a waypoint into an imminent collision state. We observe that the greedy method has a significantly lower failure rate, and behaves more consistently than ICPS-MDP. ICPS-MDP fails at a rate of 28% much more frequently than greedy with a failure rate of 3.1%, therefore impacting the distributions of objective function costs and the expected number of agents lost.

Between ICPS-MDP-mean and ICPS-MDP-ls, ICPS-MDP-mean finds better solutions with lower expected loss of agents and a lower failure rate. We find that taking more time to compute is inconsequential and will compare ICPS-MDP-mean to L'OGRE. In terms of median performance, the greedy algorithm takes 8.6% more time to compute than ICPS-MDP-mean, finds solutions with 12% lower costs, and has a 29% reduction in expected agent loss. ICPS-MDP is in fact capable of finding high quality solutions that can beat the greedy method 65% of the time, the high 28% failure rate drastically impairs the algorithm's performance in aggregate. We note that in 62.5% of the trials, greedy finds a safer solution than ICPS-MDP-mean.

The run times for generating the initial Monte Carlo were not recorded, but it is safe to assume that they were significantly longer than the run times for the other solvers. Improving the solver that can generate initial solutions is an open research question.

5.7 Discussion

We note that the task of generating the initial candidate plans is still an open research question. Note that in one of the example environments, one tour enters the convex hull of another tour, which is not penalized by the objective function. For more elaborate sensor models such behavior should be discouraged by the objective function.

In some regards, L'OGRE is conservative in how it moves waypoints as it checks the feasibility of moving the waypoint to each neighboring location before moving it, making it a statistically safer algorithm. L'OGRE can in some instances recover from poor initial solutions while the gradient based methods cannot. This exhaustiveness increases the search time a little but the algorithm is able to more efficiently store and look up previously computed value functions.

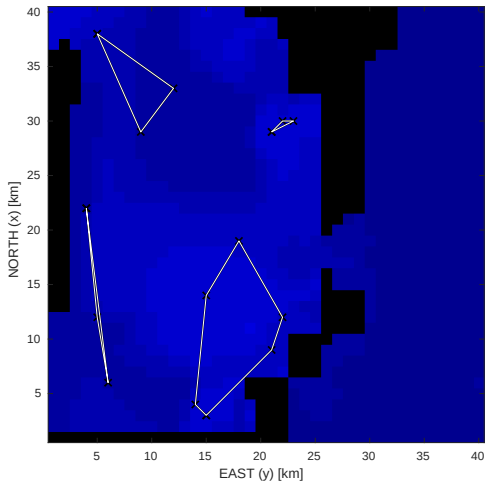
When ICPS-MDP fails, it fails because it pushes waypoints into imminent collision states and the poorly defined gradient information is unable to recover. We have tried multiple alternate definitions of the gradient of the value function, and all behave similarly in performance with significant failure rates. One could also argue that the greedy method's solution quality could be improved by intelligently considering joint actions to increase the size of $T(\tau)$. Care is needed in making computational tractability vs. solution quality tradeoffs.

Extending the Soltero algorithm with the repairing mechanism enables

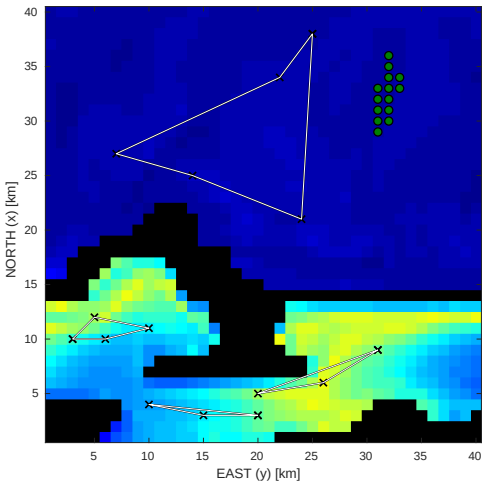
the approach to find safer multi-tours at the cost of losing solution quality, as the repair mechanism does not factor in the information content of the environment. To exclude the repairing mechanism would incur arbitrarily high cost in the objective function because the use of arc length cost does not factor the presence of obstacles or swift currents. The alternate approach to factor obstacle and swift current information is to employ ICPS-MDP.

5.8 Summary

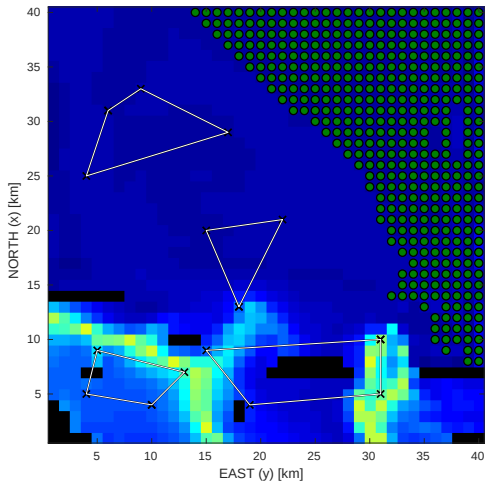
In this chapter, we introduce the problem of persistent monitoring tasks such as tracking algae blooms for teams of autonomous surface vehicles in a manner that can account for the ocean forecast data. We propose a greedy method and show that for a modest 8.6% increase in compute time, median performance results in a 12% reduction in objective function cost and a 29% reduction in expected loss of agents due to model uncertainty. These performance improvements are due to the fact that ICPS-MDP has a high 28% failure rate, though it can often beat the greedy method when it does find a solution. Both ICPS-MDP and L'OGRE outperform the state-of-the-art algorithm by Soltero.



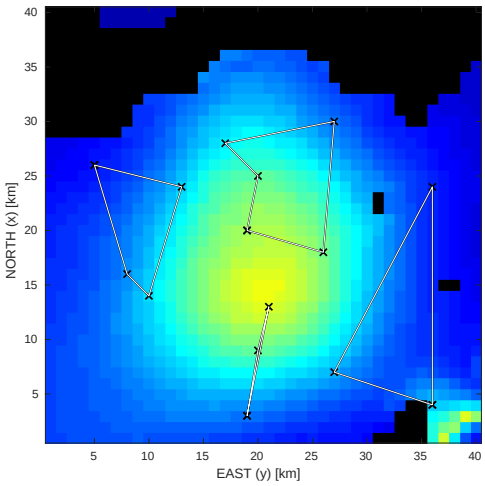
(a) Example env. with low information content.



(b) Example env. with disconnected regions.

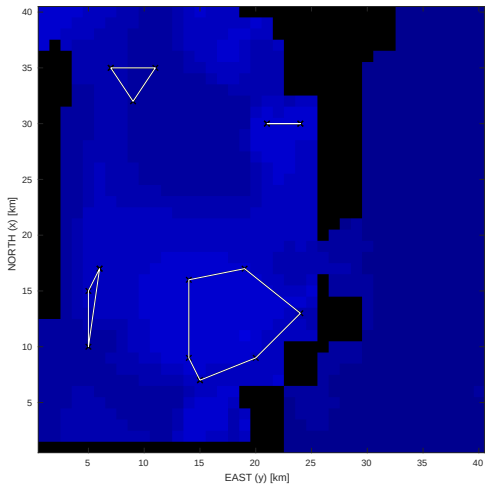


(c) Example env. with swift currents.

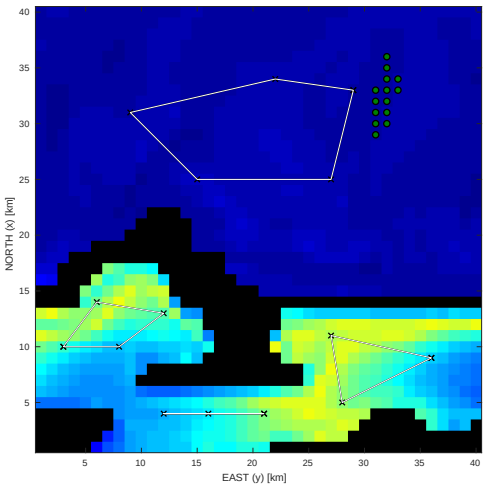


(d) Example env. with strong central dist. of info

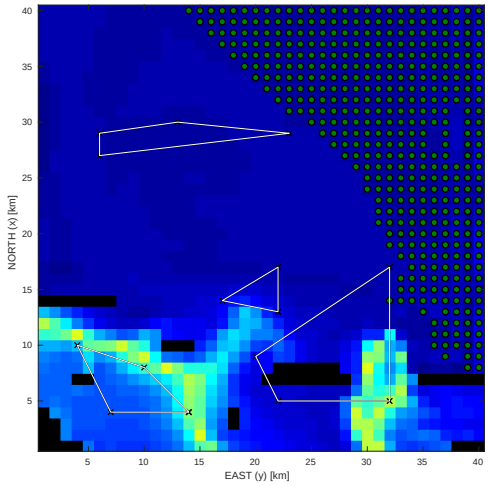
Figure 5.6: Example environments and initial waypoints



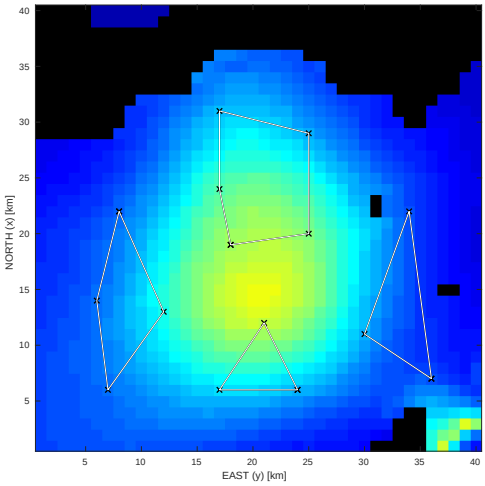
(a) Example env. with low information content.



(b) Example env. with disconnected regions.

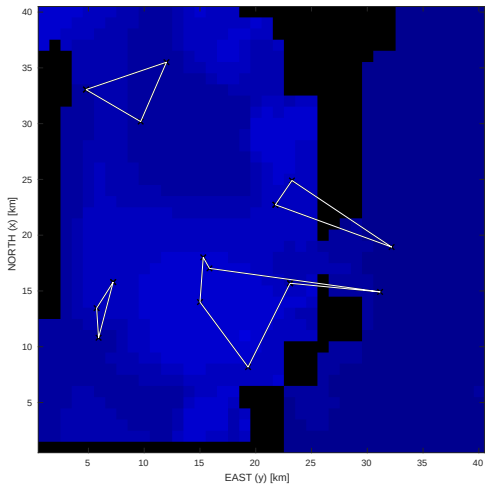


(c) Example env. with swift currents.

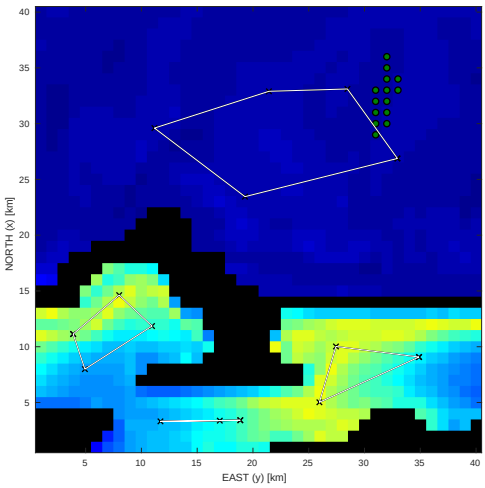


(d) Example env. with strong central dist. of info

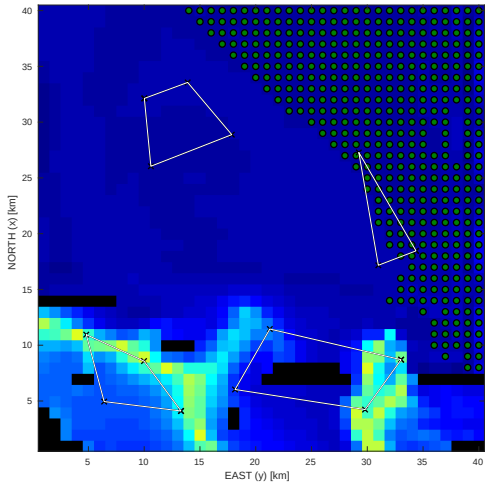
Figure 5.7: Example greedy solutions



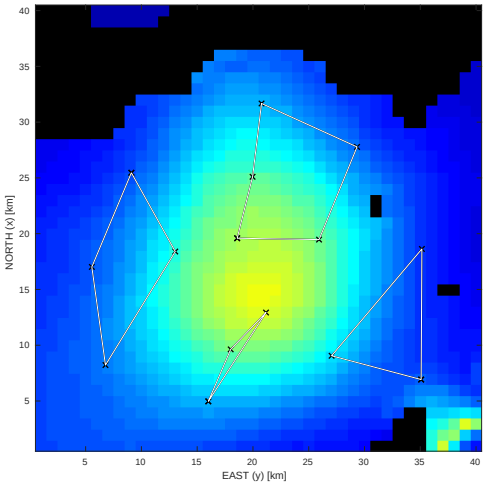
(a) Example env. with low information content.



(b) Example env. with disconnected regions.

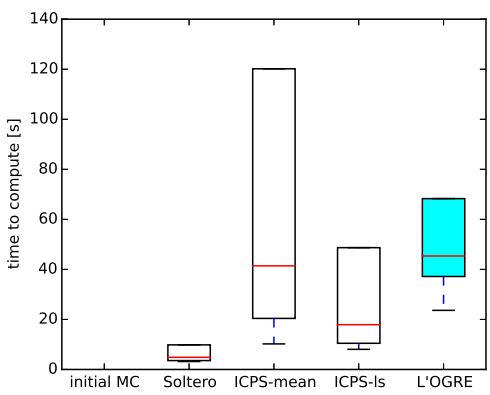


(c) Example env. with swift currents.

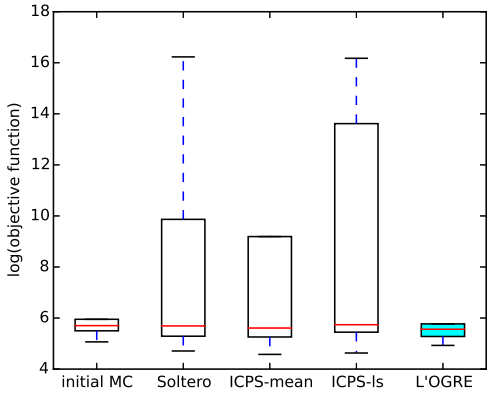


(d) Example env. with strong central dist. of info

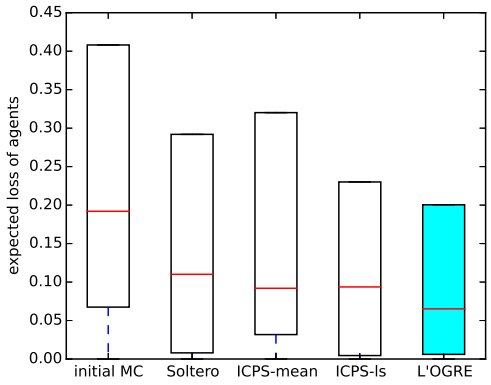
Figure 5.8: Example ICPS-MDP solutions



(a) The distribution of run times for the various algorithms. Run times for the Monte Carlo method finding initial solutions were not recorded.



(b) Box plots showing the distribution of objective function values for the various algorithms.



(c) Box plots showing the distribution of the expected agent loss in the presence of model uncertainty for the benchmarked algorithms

Figure 5.9: Box plots showing the results in the 32 environments. Outliers (whiskers) are not shown because in a small number of cases, values of high cost scenarios ($> 10^{10}$) or high expected loss of agents (>1) occur for the ICPS-MDP algorithm.

Chapter 6: Conclusions

6.1 Intellectual Contributions

While this dissertation investigated several mission scenarios for three distinct platforms, three different algorithms were developed that are of broad impact and extend to a variety of mission scenarios and vehicle types.

1. The development of ϵ-admissible heuristics. This enables branch and bound to discover high quality solutions for path-dependent rewards.

In Chapter 3, we investigated improving heuristic guided search algorithms by introducing $\epsilon-$admissible B&B, enabling the discovery of high quality solutions for information gathering tasks. The formulation and guarantees of $\epsilon-$admissible B&B were general and domain independent. $\epsilon-$admissible heuristics were introduced to aggressively prune nodes of similar reward, improving performance when applied to problems with one or more of the following attributes:

- Admissible heuristics that offer tight bounds which are difficult to construct or compute.
- Many candidate solutions are of similar heuristic reward, limiting the

pruning capabilities of, e.g. A* or B&B.

It is well known that relaxing formal guarantees of various admissible heuristics can offer tradeoffs of interest to the practitioner for search problems. However, designing heuristics that offer a tight bound on how much information is gathered while satisfying motion constraints is difficult in practice. Further, such heuristics are often computationally expensive to compute since information gathering tasks require reasoning over visiting many regions in the environment. In addition, many problems in information gathering tasks do not care about the order of observations, meaning that many trajectories have similar reward. Such similar rewards can be safely pruned by ϵ-admissible heuristics without affecting final solution quality.

On the other hand, $\epsilon-$admissible heuristics do not offer a means to reduce the branching factor and will not help in problems of high branching factor. While $\epsilon-$admissible B&B would work on a wide variety of problems, other search algorithms can be more suitable in some circumstances. For example, when a "nearness to goal" metric is available, techniques such as GBFS would be better able to exploit that information in the absence of good heuristics [23].

2. Development of the iterative greedy method for coordination of large teams for simulation-based information gathering tasks.

The solutions to information gathering tasks, especially for multi-agent problems, reside in solution spaces so large that it is difficult to find global solutions for large realistic problems (Chapter 4: 10^{301} joint action sequences.

Chapter 5: 10^{80} waypoint combinations). Finding high quality solutions requires finding a small enough coordination space that can be searched in a reasonable amount of time. In Chapter 4, we searched the space of sequentially coordinated constant action sequences because it offers an even smaller coordination space than SGA, and forms the basis of the iterative greedy method. This approach enables agents to disperse efficiently because it is able to efficiently reason over a large multitude of search directions by exploiting the ocean currents at a wide variety of depths. The iterative greedy method always reasoned about action sequences spanning the entire mission duration, and was able to outperform CEM and brute force method, techniques aimed at globally maximizing objective functions. Iterative greedy is well suited for problems with the following attributes:

- High branching factor.
- Coordinated dispersion of many agents.
- Simulation based objectives with path dependent reward structures.

Because of its dependence on SGA, iterative greedy is not suitable for problems in which SGA is not suitable. SGA and iterative greedy will have trouble in congested or maze-like environments [190]. Coordination in congested environments requires use of global optimizers. Also, the motion model must be defined such that constant actions lead to reasonable behaviors. An extreme undesirable example would be holding constant actions

for stabilizing a quadrotor, while selecting higher level actions maintaining a given bearing while executing collision avoidance routines would be suitable.

3. Development of L'OGRE to minimize costs in information gathering tasks while maintaining asset safety. In Chapter 5, we observed how L'OGRE was able to conservatively use local information to outperform various extensions to Lloyd's Algorithm, greedily refining approximate global solutions from MC. L'OGRE is guaranteed to find a local optimizer over the set of small single waypoint perturbations. L'OGRE moves waypoints into regions that were both safe to reach, and gathered more information on average, significantly improving upon state of the art Lloyd's based algorithms. L'OGRE is good for problems with the following attributes:

- High dimensional trajectory optimization problems.
- Simulation based motion models/objectives without well-defined gradients where maintaining asset safety is critical.
- Problems with a locality property ensure that joint selections need not be considered [41].

While Voronoi partitions were used in the objective function in Chapter 5, L'OGRE does not explicitly assume any structure in the objective function. L'OGRE is well suited to optimizing objectives with a locality property of information, and would work well with models such as a Gaussian Process. In Chapter 5, the dominant runtime costs of evaluating the objective function was in computing the value functions using Gauss-Seidel backups. Value

functions from previous evaluations were stored in a look-up table facilitating subsequent objective function evaluations, reducing algorithm run time.

L'OGRE would not be suitable for minimizing objectives that require reasoning over joint actions to find good solutions. For example L'OGRE would be unable to optimize an exclusive-OR (XOR) problem where two waypoint moves must be coordinated jointly. L'OGRE could be modified to consider joint actions, but algorithm runtime performance will suffer accordingly. In addition, L'OGRE would be inefficient to run if evaluating the objective cannot be facilitated by computing the cost of a neighboring trajectory, as was the case with reusing value functions from previous iterations in Chapter 5.

6.2 Anticipated Benefits

This dissertation discovered several measures to reduce mission costs when conducting various information gathering missions. In the three mentioned topics, we developed algorithms such as $\epsilon-$admissible B&B, iterative greedy, and L'OGRE that were able to gather additional information that state of the art competitors. Various utility models show that spending the extra time computing the better rewards either improves the effort of ground crews or enables one to complete missions with fewer assets, saving costs. For UAVs locating survivors (Chapter 3), we demonstrated that our proposed algorithm ϵ-admissible B&B reduced the effort required to locate survivors, enabling ground crews to be able to rescue more people in less time. When coordinat-

ing large teams of UUVs for geospatial tasks (Chapter 4), the proposed iterative greedy method reduces overall mission costs by requiring fewer vehicles to accomplish a given task (lowering deployment costs), drastically outweighing the additional computational costs required which are on the order of pennies. For ASVs in persistent monitoring tasks (Chapter 5), the proposed method L'OGRE significantly improves vehicle safety by moving waypoints to safer regions (outside regions of imminent collision) when conducting a mission while increasing the amount of information gathered. Qualitatively, we observe that exploiting the ocean currents can boost the area covered by increasing dispersion.

6.3 Potential Future Directions

The contributions of this dissertation discovered many avenues for future research. I now present two largely unexplored areas of interest that are of significant importance.

6.3.1 Information Gathering with Communications Uncertainty

One currently unresolved challenge is enabling teams of vehicles to coordinate during information gathering tasks factoring realistic constraints on communications such as limited communications range, bandwidth, or dropouts. This is important in underwater environments due to the limited bandwidth ($\approx$ 1kbit/s) and range ($\approx$ 2 km) of acoustic modems [191]. Previous work

has investigated how auction based techniques for task allocation perform as communication quality degrades [192]. The work assumes tasks are spatially distributed, with total costs computed by solving the associated m-TSP problem. However, when conducting information gathering tasks, agents will have to continually share information to improve the search. Previous work in data fusion often models communication limits as a function of distance between agents [156] or with a stationary probability of successful transmission [131]. We would like to extend such approaches to handle more realistic communication errors with correlated transmission failures. One naive solution is to transmit data structures that are robust to correlated dropped packets, increasing bandwidth on an already congested network. We propose investigating strategies that monitor the channel's error state and transmit the subsets of such data that impact decision making the most. For example, the value of information was used previously to prioritize sensor data in a bandwidth limited underwater sensor network [193], but the effects of communications dropouts on such algorithms is unknown.

6.3.2 Verification and Validation of Information Gathering Systems

One major unresolved challenge in Robotics is verifying the correctness of behavior of robotic systems. This is a distinct from test driven development [194], or other techniques to identify defects in source code [195]. For-

mal verification and can be considered a methodology in system design [196]. Alternatively, one can define the verification problem (is the task being completed successfully?) as a formal mathematical optimization problem, and using a mathematical solver with proven guarantees to generate a correct solution. However, formal method implementations are only correct if their solver implementations are free of software defects. For especially complex experiments such as multi-vehicle coordination, it can be difficult to generate test scenarios for validation. Use of adaptive sampling to identify critical transitions in systems results in fewer trials required [197]. We identify four forms of correctness that are important to assess for robotic systems:

1. correctness of the solver
2. correctness of the problem formulation
3. correctness of the objective
4. correctness of the data

Techniques such as B&B, or DP techniques are formal optimization techniques whose guarantees and solvers are well studied. Major challenges occur for designing good/correct heuristics, or in assessing how approximations degrade solution correctness. The quality of solutions generated by the iterative greedy method in Chapter 4 frustrates as it beats techniques with formal guarantees such as MCTS and CEM. However, these different solvers are capable of working in different solution spaces, given their computational

complexity. It therefore stands to reason that finding the "right" problem formulation or representation is critical in solving the problem, though this remains an open question. However, compromises need not be made between good performance and correctness. The proposed L'OGRE algorithm in Chapter 5 was correct by construction, and was found to have competitive performance.

Another important issue is the correctness of the objective. For example, it is well-known that learning strategies are good at exploiting simulation inaccuracies at maximizing objectives, which don't translate well to hardware [198]. Also, optimizing the wrong objective can have unintended consequences that the designer could not anticipate. The entire story-line of Asimov's *I, Robot* novel is the exploration of the Three Laws of Robotics and their unintended consequences when dictating robot decision making. Sometimes, solving an ancillary objective can be beneficial. For example, the use of DFS in Chapter 3 was able to generate solutions with quality only 3% less than B&B while taking 1% of the compute time. Other times, maximizing an ancillary objective is not guaranteed to maximize another objective. For example, there are complicated ties between information theory and search theory so trading objectives for computational reasons must be vetted with care [32]. Finding ancillary objectives that improve runtime performance while offering negligible losses in solution quality is an open challenge.

Appendix A: Public Release Statements

Portions of the work presented in this dissertation was performed in part at the Naval Research Laboratory, and was based on the following materials:

Chapter 3:

- "Maximizing mutual information for multipass target search in changing environments" by Kuhlman et. al. Publication approval 16-1231-3706
- "Multipass target search in natural environments" by Kuhlman et. al. Publication approval 17-1231-3546

Chapter 4:

- "Stochastic Optimization for Autonomous Vehicles with Limited Control Authority" by Jones et. al., Publication approval 18-1231-0883
- "Coordinating Underwater Vehicle Teams to Conduct Large-Scale Geospatial Tasks" by Kuhlman et. al., Publication approval 18-1231-1366

Chapter 5:

- "Physics-aware informative coverage planning for autonomous vehicles" by Kuhlman et. al., Publication approval 13-1231-4076,
- "Persistent Monitoring for Autonomous Surface Vessel Teams" by Kuhlman et. al., Publication approval 18-1231-1903

Bibliography

[1] D. Liberzon, *Calculus of Variations and Optimal Control Theory: a Concise Introduction*. Princeton, NJ, USA: Princeton University Press, 2012.

[2] M. J. Kuhlman, M. W. Otte, D. A. Sofge, and S. K. Gupta, "Multipass target search in natural environments," *Sensors*, vol. 17, no. 11, p. 2514, Nov. 2017.

[3] R. R. Murphy, S. Tadokoro, D. Nardi, A. Jacoff, P. Fiorini, H. Choset, and A. M. Erkmen, "Search and rescue robotics," in *Springer Handbook of Robotics*, B. Siciliano and O. Khatib, Eds. Heidelberg, Germany: Springer-Verlag, 2008, pp. 1151–1174.

[4] J. Qi, D. Song, H. Shang, N. Wang, C. Hua, C. Wu, X. Qi, and J. Han, "Search and rescue rotary-wing UAV and its application to the Lushan ms 7.0 earthquake," *J. of Field Robot.*, vol. 33, no. 3, pp. 290–321, 2016.

[5] P. Dames, M. Schwager, D. Rus, and V. Kumar, "Active magnetic anomaly detection using multiple micro aerial vehicles," *IEEE Robot. Autom. Lett.*, vol. 1, no. 1, pp. 153–160, 2016.

[6] M. Schwager, P. Dames, D. Rus, and V. Kumar, *A Multi-robot Control Policy for Information Gathering in the Presence of Unknown Hazards*. Cham, Switzerland: Springer, 2017, pp. 455–472.

[7] J. L. Williams, J. W. Fisher III, and A. S. Willsky, "Performance guarantees for information theoretic active inference," in *Proc. of the Eleventh Int. Conf. on Artificial Intell. and Statistics*, Mar. 2007, pp. 620–627.

[8] A. Krause and C. Guestrin, "Near-optimal nonmyopic value of information in graphical models," in *Proc. of the Twenty-First Annual Conf. on Uncertainty in Artificial Intell.*, Jul. 2005, pp. 324–331.

[9] A. Ryan and J. K. Hedrick, "Particle filter based information-theoretic active sensing," *Robot. and Auton. Syst.*, vol. 58, no. 5, pp. 574–584, 2010.

[10] J. Binney and G. S. Sukhatme, "Branch and bound for informative path planning," in *Proc. of the 2012 IEEE Int. Conf. on Robot. and Autom.*, pp. 2147–2154.

[11] G. A. Hollinger and G. S. Sukhatme, "Sampling-based robotic information gathering algorithms," *Int. J. of Robot. Res.*, pp. 1271–1287, 2014.

[12] M. J. Kuhlman, M. W. Otte, D. A. Sofge, and S. K. Gupta, "Maximizing mutual information for multipass target search in changing environments," in *Proc. of the 2017 IEEE Int. Conf. on Robot. and Autom.*, pp. 4383–4390.

[13] J. C. Kinsey, D. R. Yoerger, M. V. Jakuba, R. Camilli, C. R. Fisher, and C. R. German, "Assessing the Deepwater Horizon oil spill with the sentry autonomous underwater vehicle," in *Proc. of the 2011 IEEE/RSJ Int. Conf. on Intell. Robot. and Syst.*, pp. 261–267.

[14] E. Saltzman, R. Kiss, R. Pittenger, F. Chavez, M. Edwards, N. Fine, R. Rabalais, J. Swift, W. Wilcock, and D. Yoerger, *Science at Sea: Meeting Future Oceanographic Goals with a Robust Academic Research Fleet*. Washington, DC: The National Academies Press, 2009. [Online]. Available: https://escholarship.org/uc/item/5rz727b2

[15] M. Troesch, S. Chien, Y. Chao, and J. Farrara, "Planning and control of marine floats in the presence of dynamic, uncertain currents," in *Proc. of the Twenty-Sixth Int. Conf. on Automated Planning and Scheduling*, June 2016, pp. 431–439.

[16] M. T. Wolf, L. Blackmore, Y. Kuwata, N. Fathpour, A. Elfes, and C. Newman, "Probabilistic motion planning of balloons in strong, uncertain wind fields," in *Proc. of the 2010 IEEE Int. Conf. on Robot. and Autom.*, pp. 1123–1129.

[17] N. Fathpour, L. Blackmore, Y. Kuwata, C. Assad, M. T. Wolf, C. Newman, A. Elfes, and K. Reh, "Feasibility studies on guidance and global path planning for wind-assisted montgolfière in Titan," *IEEE Syst. J.*, vol. 8, no. 4, pp. 1112–1125, Oct. 2014.

[18] J. Das, J. Harvey, F. Py, H. Vathsangam, R. Graham, K. Rajan, and G. S. Sukhatme, "Hierarchical probabilistic regression for AUV-based adaptive sampling of marine phenomena," in *Proc. of the 2013 IEEE Int. Conf. on Robot. and Autom.*, pp. 5571–5578.

[19] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Reading, MA, USA: Addison-Wesley Pub. Co., Inc., 1984.

[20] S. M. LaValle, *Planning Algorithms*. New York, NY, USA: Cambridge University Press, 2006.

[21] G. Wagner and H. Choset, "M*: A complete multirobot path planning algorithm with performance bounds," in *Proc. of the 2011 IEEE/RSJ Int. Conf. on Intell. Robot. and Syst.*, pp. 3260–3267.

[22] M. Likhachev, G. J. Gordon, and S. Thrun, "ARA*: Anytime a* with provable bounds on sub-optimality," in *Advances in neural information processing systems*, Dec 2003, pp. 767–774.

[23] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning and Acting*. New York, NY, USA: Cambridge University Press, 2016.

[24] A. M. Kabir, J. D. Langsfeld, S. Shriyam, V. S. Rachakonda, C. Zhuang, K. N. Kaipa, J. Marvel, and S. K. Gupta, "Planning algorithms for multi-setup multi-pass robotic cleaning with oscillatory moving tools," in *Proc. of the 2011 Int. Conf. on Automation Sci. and Eng.*, pp. 751–757.

[25] T. Stewart, "Search for a moving target when searcher motion is restricted," *Computers & Operations Res.*, vol. 6, no. 3, pp. 129–140, 1979.

[26] H. Sato and J. O. Royset, "Path optimization for the resource-constrained searcher," *Naval Res. Logistics*, vol. 57, no. 5, pp. 422–440, 2010.

[27] P. M. Narendra and K. Fukunaga, "A branch and bound algorithm for feature subset selection," *IEEE Trans. Comput.*, vol. 26, no. 9, pp. 917–922, 1977.

[28] M. Held and R. M. Karp, "The traveling-salesman problem and minimum spanning trees," *Operations Res.*, vol. 18, no. 6, pp. 1138–1162, 1970.

[29] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proc. of the American Math. Society*, vol. 7, no. 1, pp. 48–50, 1956.

[30] J. Erikson, *Algorithms*, 2015, ch. Minimum Spanning Trees. [Online]. Available: http://jeffe.cs.illinois.edu/teaching/algorithms/notes/20-mst.pdf

[31] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. Hoboken, NJ, USA: John Wiley & Sons, 2012.

[32] J. G. Pierce, "A new look at the relation between information theory and search theory," in *The Maximum Entropy Formalism*. Cambridge, MA, USA: MIT Press, 1978, pp. 339–402.

[33] E.-M. Wong, F. Bourgault, and T. Furukawa, "Multi-vehicle Bayesian search for multiple lost targets," in *Proc. of the 2005 IEEE Int. Conf. on Robot. and Autom.*, pp. 3169–3174.

[34] J. Tisdale, Z. Kim, and J. K. Hedrick, "Autonomous UAV path planning and estimation," *IEEE Robot. Autom. Mag.*, vol. 16, no. 2, pp. 35–42, 2009.

[35] G. A. Hollinger, S. Yerramalli, S. Singh, U. Mitra, and G. S. Sukhatme, "Distributed data fusion for multirobot search," *IEEE Trans. Robot.*, vol. 31, no. 1, pp. 55–66, 2015.

[36] G. L. Nemhauser and L. A. Wolsey, "Maximizing submodular set functions: formulations and analysis of algorithms," in *North-Holland Mathematics Studies*. Amsterdam, Netherlands: Elsevier, 1981, vol. 59, pp. 279–301.

[37] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, "An analysis of approximations for maximizing submodular set functionsi," *Math. Programming*, vol. 14, no. 1, pp. 265–294, 1978.

[38] C. Chekuri and M. Pal, "A recursive greedy algorithm for walks in directed graphs," in *Proc. of the 46th Annual IEEE Symp. on Foundations of Computer Science*, Oct 2005, pp. 245–253.

[39] G. Hollinger, S. Singh, J. Djugash, and A. Kehagias, "Efficient multi-robot search for a moving target," *Int. J. of Robot. Res.*, vol. 28, no. 2, pp. 201–219, Feb. 2009.

[40] M. Corah and N. Michael, "Efficient online multi-robot exploration via distributed sequential greedy assignment," in *Robotics: Science and Systems*, Jul. 2017.

[41] A. Krause, C. Guestrin, A. Gupta, and J. Kleinberg, "Near-optimal sensor placements: Maximizing information while minimizing communication cost," in *Proc. of the 5th Int. Conf. on Information Processing in Sensor Networks*, Apr. 2006, pp. 2–10.

[42] A. Singh, A. Krause, C. Guestrin, W. J. Kaiser, and M. A. Batalin, "Efficient planning of informative paths for multiple robots," in *Proc. of the of the 20th Int. Joint Conf. on Artificial Intell.*, pp. 2204–2211.

[43] D. Levine, B. Luders, and J. P. How, "Information-rich path planning with general constraints using rapidly-exploring random trees," in *Proc. of the AIAA Infotech@Aerospace*, Apr. 2010.

[44] G. Hollinger and S. Singh, "Multi-robot coordination with periodic connectivity," in *Proc. of the 2010 IEEE Int. Conf. on Robot. and Autom.*, pp. 4457–4462.

[45] E. Galceran and M. Carreras, "A survey on coverage path planning for robotics," *Robot. and Auton. Syst.*, vol. 61, no. 12, pp. 1258–1276, 2013.

[46] A. Zelinsky, R. Jarvis, J. C. Byrne, and S. Yuta, "Planning paths of complete coverage of an unstructured environment by a mobile robot," in *Proc. of the 1993 Int. Conf. on Adv. Robot.*, May 1993, pp. 533–538.

[47] H. Choset, "Coverage of known spaces: The boustrophedon cellular decomposition," *Auton. Robot.*, vol. 9, no. 3, pp. 247–253, 2000.

[48] M. Meghjani, S. Manjanna, and G. Dudek, "Multi-target rendezvous search," in *Proc. of the 2016 IEEE/RSJ Int. Conf. on Intell. Robot. and Syst.*, Oct 2016, pp. 2596–2603.

[49] V. Shivashankar, R. Jain, U. Kuter, and D. S. Nau, "Real-time planning for covering an initially-unknown spatial environment." in *Proceedings of the Twenty-Fourth Int. Florida Artificial Intell. Res. Soc. Conf.*, May 2011, pp. 63–68.

[50] C. T. Cunningham and R. S. Roberts, "An adaptive path planning algorithm for cooperating unmanned air vehicles," in *Proc. of the 2001 IEEE Int. Conf. on Robot. and Autom.*, pp. 3981–3986.

[51] D. Sofge, A. Schultz, and K. DeJong, "Evolutionary computational approaches to solving the multiple traveling salesman problem using a neighborhood attractor schema," in *Proc. of the Appl. of Evolutionary Computing on EvoWorkshops 2002: EvoCOP, EvoIASP, EvoSTIM/EvoPLAN*, pp. 153–162.

[52] M.-K. Kwan, "Graphic programming using odd or even points," *Chinese Math*, vol. 1, pp. 273–277, 1962.

[53] K. P. Dahl, D. R. Thompson, D. McLaren, Y. Chao, and S. Chien, "Current-sensitive path planning for an underactuated free-floating ocean sensorweb," in *Proc. of the 2011 IEEE/RSJ Int. Conf. on Intell. Robot. and Syst.*, pp. 3140–3146.

[54] J. McMahon, H. Yetkin, A. Wolek, Z. J. Waters, and D. J. Stilwell, "Towards real-time search planning in subsea environments," in *Proc. of the 2017 IEEE/RSJ Int. Conf. on Intell. Robot. and Syst.*, pp. 87–94.

[55] Z. Drezner, *Facility Location: a Survey of Applications and Methods*. New York, NY, USA: Springer-Verlag, 1995.

[56] J. Cortés, S. Martinez, T. Karatas, and F. Bullo, "Coverage control for mobile sensing networks," *IEEE Trans. Robot. Autom.*, vol. 20, no. 2, pp. 243–255, 2004.

[57] M. Candeloro, A. M. Lekkas, and A. J. Sørensen, "A Voronoi-diagram-based dynamic path-planning system for underactuated marine vessels," *Control Eng. Practice*, vol. 61, pp. 41 – 54, 2017.

[58] A. Kwok and S. Martínez, "A coverage algorithm for drifters in a river environment," in *Proc. of the 2010 American Control Conf.*.

[59] L. Zuo, W. Yan, R. Cui, and J. Gao, "A coverage algorithm for multiple autonomous surface vehicles in flowing environments," *Int. J. of Control, Automation and Systems*, vol. 14, no. 2, pp. 540–548, 2016.

[60] A. Marino and G. Antonelli, "Experimental results of coordinated sampling/patrolling by autonomous underwater vehicles," in *Proc. of the 2013 IEEE Int. Conf. on Robot. and Autom.*, pp. 4141–4146.

[61] S. Kemna, J. G. Rogers, C. Nieto-Granda, S. Young, and G. S. Sukhatme, "Multi-robot coordination through dynamic Voronoi partitioning for informative adaptive sampling in communication-constrained environments," in *Proc. of the 2017 IEEE Int. Conf. on Robot. and Autom.*, May 2017, pp. 2124–2130.

[62] L. C. Pimenta, M. Schwager, Q. Lindsey, V. Kumar, D. Rus, R. C. Mesquita, and G. A. Pereira, "Simultaneous coverage and tracking (SCAT) of moving targets with robot networks," in *Algorithmic Foundation of Robotics VIII*. Berlin, Germany: Springer, 2009, pp. 85–99.

[63] A. Okabe and A. Suzuki, "Locational optimization problems solved through voronoi diagrams," *European J. of Oper. Res.*, vol. 98, no. 3, pp. 445 – 456, 1997.

[64] S. Lloyd, "Least squares quantization in PCM," *IEEE Trans. Inf. Theory*, vol. 28, no. 2, pp. 129–137, March 1982.

[65] S. Salapaka, A. Khalak, and M. Dahleh, "Constraints on locational optimization problems," in *Proc. of the 2003 Conf. on Decision and Contr.*, pp. 1741–1746.

[66] D. E. Soltero, M. Schwager, and D. Rus, "Generating informative paths for persistent sensing in unknown environments," in *Proc. of the 2012 IEEE/RSJ Int. Conf. on Intell. Robot. and Syst.*, pp. 2172–2179.

[67] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Hoboken, NJ, USA: John Wiley & Sons, 1994.

[68] A. Thakur, P. Švec, and S. K. Gupta, "GPU based generation of state transition models using simulations for unmanned surface vehicle trajectory planning," *Robot. and Auton. Syst.*, vol. 60, no. 12, pp. 1457–1471, Dec. 2012.

[69] P. Švec, B. C. Shah, I. R. Bertaska, J. Alvarez, A. J. Sinisterra, K. v. Ellenrieder, M. Dhanak, and S. K. Gupta, "Dynamics-aware target following for an autonomous surface vehicle operating under COLREGs in civilian traffic," in *Proc. of the 2013 IEEE/RSJ Int. Conf. on Intell. Robot. and Syst.*, pp. 3871–3878.

[70] K.-C. Ma, L. Liu, and G. S. Sukhatme, "An information-driven and disturbance-aware planning method for long-term ocean monitoring," in *Proc. of the 2016 IEEE Int. Conf. on Robot. and Autom.*, pp. 2102–2108.

[71] C. Boutilier, "Sequential optimality and coordination in multiagent systems," in *Proc. of the 1999 Int. Joint Conf. on Artificial Intell.*, pp. 478–485.

[72] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.

[73] Mausam and A. Kolobov, *Planning with Markov Decision Processes: an AI Perspective*. San Rafael, CA, USA: Morgan & Claypool Publishers, 2012.

[74] D. Bertsekas, "Distributed dynamic programming," *IEEE Trans. Autom. Control*, vol. 27, no. 3, pp. 610–616, 1982.

[75] A. W. Moore and C. G. Atkeson, "Prioritized sweeping: reinforcement learning with less data and less time," *Machine Learning*, vol. 13, no. 1, pp. 103–130, 1993.

[76] D. Wingate and K. D. Seppi, "Efficient value iteration using partitioned models," in *Proc. of the Int. Conf. on Machine Learning and Appl.*, June 2003, pp. 53–59.

[77] ——, "P3VI: A partitioned, prioritized, parallel value iterator," in *Proc. of the Twenty-First Int. Conf. on Machine Learning*, July 2004, pp. 109–117.

[78] G. Chaslot, "Monte-Carlo Tree Search," Ph.D. dissertation, Maastricht: Universiteit Maastricht, Maastricht, Netherlands, 2010.

[79] J. A. Caley and G. A. Hollinger, "Data-driven comparison of spatio-temporal monitoring techniques," in *Proc. of OCEANS 2015 - MTS/IEEE Washington*, pp. 1–7.

[80] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *Proc. of the 17th European Conf. on Machine Learning*, vol. 6, Sept. 2006, pp. 282–293.

[81] A. Fern and P. Lewis, "Ensemble Monte-Carlo planning: an empirical study," in *Proc. of the Twenty-First Int. Conf. on Automated Planning and Scheduling*, June 2011, pp. 58–65.

[82] B. Bonet and H. Geffner, "Action selection for MDPs: Anytime AO* versus UCT," in *Proc. of the Twenty-Sixth AAAI Conf. on Artificial Intell.*, July 2012, pp. 1749–1755.

[83] R. Y. Rubinstein and D. P. Kroese, *The Cross-Entropy Method: A Unified Approach to Monte Carlo Simulation, Randomized Optimization and Machine Learning.* New York, NY, USA: Springer-Verlag, 2004.

[84] P.-T. De Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein, "A tutorial on the cross-entropy method," *Annals of Operations Res.*, vol. 134, pp. 19–67, Feb 2005.

[85] S. Mannor, R. Y. Rubinstein, and Y. Gat, "The cross entropy method for fast policy search," in *Proc. of the 20th Int. Conf. on Machine Learning*, Aug. 2003, pp. 512–519.

[86] T. Y. Teck and M. Chitre, "Single beacon cooperative path planning using cross-entropy method," in *Proc. of OCEANS 2011- MTS/IEEE Kona*, Sept 2011, pp. 1–6.

[87] Y. T. Tan, R. Gao, and M. Chitre, "Cooperative path planning for range-only localization using a single moving beacon," *IEEE J. Ocean. Eng.*, vol. 39, no. 2, pp. 371–385, 2014.

[88] F. Stulp and O. Sigaud, "Path integral policy improvement with covariance matrix adaptation," arXiv:1206.4621(cs.LG), 2012.

[89] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou, "Information theoretic MPC for model-based reinforcement learning," in *Proc. of the 2017 IEEE Int. Conf. on Robot. and Autom.*, pp. 1714–1721.

[90] A. Costa, O. D. Jones, and D. Kroese, "Convergence properties of the cross-entropy method for discrete optimization," *Operations Res. Letters*, vol. 35, no. 5, pp. 573–580, 2007.

[91] F. Dambreville, "Cross-entropy method: convergence issues for extended implementation," arXiv:math/0609461, Sept. 2006.

[92] K. H. Low, J. M. Dolan, and P. Khosla, "Active markov information-theoretic path planning for robotic environmental sensing," in *Proc. of the 10th Int. Conf. on Autonomous Agents and Multiagent Systems*, May 2011, pp. 753–760.

[93] M. Roberts, S. Shah, D. Dey, A. Truong, S. Sinha, A. Kapoor, P. Hanrahan, and N. Joshi, "Submodular trajectory optimization for aerial 3D scanning," in *Proc. of the 2017 IEEE Int. Conf. on Computer Vision*, pp. 5334–5343.

[94] H. Schellhorn, "An algorithm for optimal stopping with path-dependent rewards based on regression and malliavin calculus," in *AIP Conf. Proc.*, vol. 936, no. 1, 2007, pp. 503–506.

[95] G. A. Hollinger, A. A. Pereira, and G. S. Sukhatme, "Learning uncertainty models for reliable operation of autonomous underwater vehicles," in *Proc. of the 2013 IEEE Int. Conf. on Robot. and Autom.*, pp. 5593–5599.

[96] J. Quiñonero-Candela and C. E. Rasmussen, "A unifying view of sparse approximate Gaussian process regression," *J. of Machine Learning Res.*, vol. 6, pp. 1939–1959, Dec 2005.

[97] R. Furrer, M. G. Genton, and D. Nychka, "Covariance tapering for interpolation of large spatial datasets," *J. of Computational and Graphical Statistics*, vol. 15, no. 3, pp. 502–523, 2006.

[98] A. Datta, S. Banerjee, A. O. Finley, and A. E. Gelfand, "Hierarchical nearest-neighbor gaussian process models for large geostatistical datasets," *J. of the American Statistical Association*, vol. 111, no. 514, pp. 800–812, Aug. 2016.

[99] K. Erkorkmaz and Y. Altintas, "High speed CNC system design. Part I: jerk limited trajectory generation and quintic spline interpolation," *Int. J. of Machine Tools and Manufacture*, vol. 41, no. 9, pp. 1323–1345, 2001.

[100] R. W. Brockett, *Finite Dimensional Linear Systems*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics (SIAM), 2015.

[101] M. De Pinho, M. Ferreira, and F. Fontes, "An Euler–Lagrange inclusion for optimal control problems with state constraints," *J. of Dynamical and Control Systems*, vol. 8, no. 1, pp. 23–45, Jan. 2002.

[102] S. Karaman and E. Frazzoli, "Optimal kinodynamic motion planning using incremental sampling-based methods," in *Proc. of the 49th Conf. on Decision and Contr.*, Dec. 2010, pp. 7681–7687.

[103] D. J. Webb and J. van den Berg, "Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics," in *Proc. of the 2013 IEEE Int. Conf. on Robot. and Autom.*, pp. 5054–5061.

[104] R. Fierro and F. L. Lewis, "Control of a nonholomic mobile robot: Backstepping kinematics into dynamics," *J. of Robotic Systems*, vol. 14, no. 3, pp. 149–163, 1997.

[105] V. N. M. J. and M. R. M., "Real-time trajectory generation for differentially flat systems," *Int. J. of Robust and Nonlinear Control*, vol. 8, no. 11, pp. 995–1020.

[106] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *Proc. of the 2011 IEEE Int. Conf. on Robot. and Autom.*, pp. 2520–2525.

[107] M. W. Achtelik, S. Lynen, M. Chli, and R. Siegwart, "Inversion based direct position control and trajectory following for micro aerial vehicles," in *Proc. of the 2013 IEEE/RSJ Int. Conf. on Intell. Robot. and Syst.*, pp. 2933–2939.

[108] C. Richter, A. Bry, and N. Roy, *Polynomial Trajectory Planning for Aggressive Quadrotor Flight in Dense Indoor Environments*. New York, NY, USA: Springer, 2016, pp. 649–666.

[109] R. Choe, J. Puig, V. Cichella, E. Xargay, and N. Hovakimyan, "Trajectory generation using spatial pythagorean hodograph Bézier curves," in *Proc. of the AIAA Guidance, Navigation, and Control Conf.*, Jun. 2015.

[110] E. Frazzoli, M. A. Dahleh, and E. Feron, "Maneuver-based motion planning for nonlinear systems with symmetries," *IEEE Transactions on Robotics*, vol. 21, no. 6, pp. 1077–1091, Dec 2005.

[111] E. Frazzoli, M. Dahleh, E. Feron *et al.*, "Real-time motion planning for agile autonomous vehicles," in *Proc. of the 2001 American Control Conf.*, Jun. 2001, pp. 43–49.

[112] T. Tomić, M. Maier, and S. Haddadin, "Learning quadrotor maneuvers from optimal control and generalizing in real-time," in *Proc. of the 2014 IEEE Int. Conf. on Robot. and Autom.*, pp. 1747–1754.

[113] M. Likhachev and D. Ferguson, "Planning long dynamically feasible maneuvers for autonomous vehicles," *Int. J. of Robot. Res.*, vol. 28, no. 8, pp. 933–945, 2009.

[114] B. C. Shah, P. Švec, I. R. Bertaska, A. J. Sinisterra, W. Klinger, K. von Ellenrieder, M. Dhanak, and S. K. Gupta, "Resolution-adaptive risk-aware trajectory planning for surface vehicles operating in congested civilian traffic," *Auton. Robot.*, vol. 40, no. 7, pp. 1139–1163, Oct 2016.

[115] G. Wagner, H. Choset, and A. Siravuru, "Multirobot sequential composition," in *Proc. of the 2016 IEEE/RSJ Int. Conf. on Intell. Robot. and Syst.*, pp. 2081–2088.

[116] B. Koopman, "The theory of search. II. target detection," *Operations Res.*, vol. 4, no. 5, pp. 503–531, 1956.

[117] S. Waharte and N. Trigoni, "Supporting search and rescue operations with UAVs," in *Int. Conf. on Emerging Security Technologies*, Sept. 2010, pp. 142–147.

[118] T. H. Chung, G. A. Hollinger, and V. Isler, "Search and pursuit-evasion in mobile robotics," *Auton. Robots*, vol. 31, no. 4, pp. 299–316, 2011.

[119] C. Robin and S. Lacroix, "Multi-robot target detection and tracking: taxonomy and survey," *Auton. Robots*, vol. 40, no. 4, pp. 729–760, 2016.

[120] P. Dames and V. Kumar, "Autonomous localization of an unknown number of targets without data association using teams of mobile sensors," *IEEE Trans. Autom. Sci. Eng.*, vol. 12, no. 3, pp. 850–864, 2015.

[121] T. Furukawa, F. Bourgault, B. Lavis, and H. F. Durrant-Whyte, "Recursive Bayesian search-and-tracking using coordinated UAVs for lost targets," in *Proc. of the 2006 IEEE Int. Conf. on Robot. and Autom.*, pp. 2521–2526.

[122] R. Mahler, "statistics 102 for multisource-multitarget detection and tracking," *IEEE J. Sel. Topics Signal Process.*, vol. 7, no. 3, pp. 376–389, 2013.

[123] L. F. Bertuccelli and J. P. How, "Robust UAV search for environments with imprecise probability maps," in *Proc. of the 2005 IEEE Conf. on Decision and Control and the European Control Conf.*, Dec 2005, pp. 5680–5685.

[124] A. Shende, M. J. Bays, and D. J. Stilwell, "Toward a mission value for subsea search with bottom-type variability," in *Proc. of the 2012 Oceans - Yeosu*, pp. 1–5.

[125] H. Yetkin, C. Lutz, and D. Stilwell, "Utility-based adaptive path planning for subsea search," in *Proc. of MTS/IEEE OCEANS'15*, Oct. 2015, pp. 1–6.

[126] G. Hollinger, S. Singh, J. Djugash, and A. Kehagias, "Efficient multi-robot search for a moving target," *Int. J. of Robot. Res.*, vol. 28, no. 2, pp. 201–219, 2009.

[127] S. Kemna, M. J. Hamilton, D. T. Hughes, and K. D. LePage, "Adaptive autonomous underwater vehicles for littoral surveillance," *Intelligent service robotics*, vol. 4, no. 4, p. 245, 2011.

[128] S. Coraluppi and C. Carthel, "Multi-hypothesis sonar tracking," in *Proc. of the Seventh Int. Conf. on Information Fusion*, Jun. 2004, pp. 33–40.

[129] A. Khan, E. Yanmaz, and B. Rinner, "Information exchange and decision making in micro aerial vehicle networks for cooperative search," *IEEE Trans. Control Netw. Syst.*, vol. 2, no. 4, pp. 335–347, 2015.

[130] F. Bourgault, T. Furukawa, and H. E. Durrant-Whyte, "Coordinated decentralized search for a lost target in a Bayesian world," in *Proc. of the 2003 IEEE/RSJ Int. Conf. on Intell. Robot. and Syst.*, pp. 48–53.

[131] G. A. Hollinger, S. Yerramalli, S. Singh, U. Mitra, and G. S. Sukhatme, "Distributed coordination and data fusion for underwater search," in *Proc. of the 2011 IEEE Int. Conf. on Robot. and Autom.*, pp. 349–355.

[132] M. Otte, M. J. Kuhlman, and D. Sofge, "Competitive two team target search game with communication symmetry and asymmetry," in *Int. Workshop on the Algorithmic Foundations of Robotics Papers*, Dec. 2016.

[133] P. Lanillos, E. Besada-Portas, J. A. Lopez-Orozco, and J. M. de la Cruz, "Minimum time search in uncertain dynamic domains with complex sensorial platforms," *Sensors*, vol. 14, no. 8, pp. 14 131–14 179, 2014.

[134] S. Perez-Carabaza, E. Besada-Portas, J. A. Lopez-Orozco, and J. M. de la Cruz, "A real world multi-UAV evolutionary planner for minimum time target detection," in *Proc. of the Genetic and Evolutionary Computation Conf.*, 2016, pp. 981–988.

[135] T. H. Chung and S. Carpin, "Multiscale search using probabilistic quadtrees," in *Proc. of the 2011 IEEE Int. Conf. on Robot. and Autom.*, pp. 2546–2553.

[136] S. Carpin, D. Burch, and T. H. Chung, "Searching for multiple targets using probabilistic quadtrees," in *Proc. of the 2011 IEEE/RSJ Int. Conf. on Intell. Robot. and Syst.*, pp. 4536–4543.

[137] H. Lau, S. Huang, and G. Dissanayake, "Probabilistic search for a moving target in an indoor environment," in *Proc. of the 2006 IEEE/RSJ Int. Conf. on Intell. Robot. and Syst.*, pp. 3393–3398.

[138] P. Lermusiaux, D. Subramani, J. Lin, C. Kulkarni, A. Gupta, A. Dutt, T. Lolla, P. Haley, W. Ali, C. Mirabito, and S. Jana, "A future for intelligent autonomous ocean observing systems," *J. of Marine Res.*, vol. 75, no. 6, pp. 765–813, 2017.

[139] N. E. Leonard, D. A. Paley, R. E. Davis, D. M. Fratantoni, F. Lekien, and F. Zhang, "Coordinated control of an underwater glider fleet in an adaptive ocean sampling field experiment in Monterey Bay," *J. of Field Robot.*, vol. 27, no. 6, pp. 718–740, Sept. 2010.

[140] A. Marino and G. Antonelli, "Experiments on sampling/patrolling with two autonomous underwater vehicles," *Robot. and Auton. Syst.*, vol. 67, pp. 61–71, May 2015.

[141] J. Das, F. Py, T. Maughan, T. OReilly, M. Messi, J. Ryan, G. S. Sukhatme, and K. Rajan, "Coordinated sampling of dynamic oceanographic features with underwater vehicles and drifters," *Int. J. of Robot. Res.*, vol. 31, no. 5, pp. 626–646, 2012.

[142] N. E. Leonard, D. A. Paley, F. Lekien, R. Sepulchre, D. M. Fratantoni, and R. E. Davis, "Collective motion, sensor networks, and ocean sampling," *Proc. IEEE*, vol. 95, no. 1, pp. 48–74, Jan 2007.

[143] A. Garcia-Olaya, F. Py, J. Das, and K. Rajan, "An online utility-based approach for sampling dynamic ocean fields," *IEEE J. Ocean. Eng.*, vol. 37, no. 2, pp. 185–203, April 2012.

[144] N. K. Yilmaz, C. Evangelinos, P. F. J. Lermusiaux, and N. M. Patrikalakis, "Path planning of autonomous underwater vehicles for

adaptive sampling using mixed integer linear programming," *IEEE J. Ocean. Eng.*, vol. 33, no. 4, pp. 522–537, Oct 2008.

[145] D. Kularatne, S. Bhattacharya, and M. A. Hsieh, "Time and energy optimal path planning in general flows," in *Proc. of Robotics: Science and Systems*, June 2016.

[146] M. Wei, G. Jacobs, C. Rowley, C. N. Barron, P. Hogan, P. Spence, O. M. Smedstad, P. Martin, P. Muscarella, and E. Coelho, "The performance of the US Navy's RELO ensemble, NCOM, HYCOM during the period of GLAD at-sea experiment in the Gulf of Mexico," *Deep Sea Res. Part II: Topical Studies in Oceanography*, vol. 129, pp. 374–393, 2016.

[147] R. N. Smith, Y. Chao, P. P. Li, D. A. Caron, B. H. Jones, and G. S. Sukhatme, "Planning and implementing trajectories for autonomous underwater vehicles to track evolving ocean processes based on predictions from a regional ocean model," *Int. J. of Robot. Res.*, vol. 29, no. 12, pp. 1475–1497, 2010.

[148] M. Leutbecher and T. N. Palmer, "Ensemble forecasting," *J. of Computational Physics*, vol. 227, no. 7, pp. 3515–3539, 2008.

[149] T. Wang, O. P. Le Maître, I. Hoteit, and O. M. Knio, "Path planning in uncertain flow fields using ensemble method," *Ocean Dynamics*, vol. 66, no. 10, pp. 1231–1251, 2016.

[150] P. Švec, M. Schwartz, A. Thakur, and S. K. Gupta, "Trajectory planning with look-ahead for unmanned sea surface vehicles to handle environmental disturbances," in *Proc. of the 2011 IEEE/RSJ Int. Conf. on Intell. Robot. and Syst.*, pp. 1154–1159.

[151] S. Smith, M. Schwager, and D. Rus, "Persistent robotic tasks: Monitoring and sweeping in changing environments," *IEEE Trans. Robot. Autom.*, vol. 28, no. 2, pp. 410–426, 2012.

[152] J. O'rourke, *Art Gallery Theorems and Algorithms*. Oxford, United Kingdom: Oxford University Press, 1987.

[153] T. Alam, G. M. Reis, L. Bobadilla, and R. N. Smith, "A data-driven deployment approach for persistent monitoring in aquatic environments,"

in *Proc. of 2018 Second IEEE Int. Conf. on Robotic Computing*, pp. 147–154.

[154] S. Shriyam, B. C. Shah, and S. K. Gupta, "Decomposition of collaborative surveillance tasks for execution in marine environments by a team of unmanned surface vehicles," *J. of Mechanisms and Robot.*, vol. 10, no. 2, p. 025007, 2018.

[155] R. Graham and J. Cortés, "Adaptive information collection by robotic sensor networks for spatial estimation," *IEEE Trans. Autom. Control*, vol. 57, no. 6, pp. 1404–1419, 2012.

[156] B. J. Julian, M. Angermann, M. Schwager, and D. Rus, "Distributed robotic sensor networks: An information theoretic approach," *Int. J. of Robot. Res.*, vol. 31, no. 10, pp. 1134–1154, 2012.

[157] M. J. Kuhlman, P. Švec, K. N. Kaipa, D. Sofge, and S. K. Gupta, "Physics-aware informative coverage planning for autonomous vehicles," in *Proc. of the 2014 IEEE Int. Conf. on Robot. and Autom.*, pp. 4741–4746.

[158] E. Raboin, P. Švec, D. S. Nau, and S. K. Gupta, "Model-predictive asset guarding by team of autonomous surface vehicles in environment with civilian boats," *Autonomous Robots*, vol. 38, no. 3, pp. 261–282, Mar 2015.

[159] M. Kwong, "Nepal earthquake: Drones used by Canadian relief team," CBC News World, April 27, 2015. [Online]. Available: http://www.cbc.ca/news/world/nepal-earthquake-drones-used-by-canadian-relief-team-1.3051106

[160] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, MA, USA: MIT Press, 2005.

[161] L. R. Harris, "The heuristic search under conditions of error," *Artificial Intell.*, vol. 5, no. 3, pp. 217–234, 1974.

[162] M. Lauri and R. Ritala, "Optimal sensing via multi-armed bandit relaxations in mixed observability domains," in *Proc. of the 2015 IEEE Int. Conf. on Robot. and Autom.*, pp. 4807–4812.

[163] J. C. Gittins, "Bandit processes and dynamic allocation indices," *J. of the Royal Statistical Society. Series B (Methodological)*, vol. 41, no. 2, pp. 148–177, 1979.

[164] S. Edelkamp and S. Schroedl, *Heuristic Search: Theory and Applications*. Waltham, MA, USA: Elsevier, 2011.

[165] S. Karaman and E. Frazzoli, "High-speed flight in an ergodic forest," in *Proc. of the 2012 IEEE Int. Conf. on Robot. and Autom.*, pp. 2899–2906.

[166] L. D. Stone, *Theory of Optimal Search*. New York, NY, USA: Academic Press, Inc., 1976.

[167] W. Viscusi, "The value of life," Harvard Law School John M. Olin Center for Law, Economics and Business Discussion Paper Series, Tech. Rep. Paper 517., 2005.

[168] S. Van Der Walt, S. C. Colbert, and G. Varoquaux, "The numpy array: a structure for efficient numerical computation," *Computing in Science & Eng.*, vol. 13, no. 2, pp. 22–30, 2011.

[169] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using NetworkX," in *Proc. of the 7th Python in Science Conf. (SciPy2008)*, Aug 2008, pp. 11–15.

[170] K. Perlin, *Real-time Shading*. ACM SIGGRAPH Course Notes, 2001, ch. 9: Noise Hardware. [Online]. Available: https://www.csee.umbc.edu/~olano/s2001c24/

[171] A. Lagae, S. Lefebvre, R. Cook, T. DeRose, G. Drettakis, D. S. Ebert, J. P. Lewis, K. Perlin, and M. Zwicker, "A survey of procedural noise functions," *Computer Graphics Forum*, vol. 29, no. 8, pp. 2579–2600, 2010.

[172] K. Spencer. *OpenSimplexNoise.java*. [Online]. Available: https://gist.github.com/KdotJPG/b1270127455a94ac5d19. Accessed on: March 2, 2016.

[173] A. Svensson. *opensimplex 0.2*. [Online]. Available: https://pypi.python.org/pypi/opensimplex/. Accessed on: March 2, 2016.

[174] B. C. Shah and S. K. Gupta, "Speeding up A* search on visibility graphs defined over quadtrees to enable long distance path planning for unmanned surface vehicles," in *Proc. of the Twenty-Sixth Int. Conf. on Automated Planning and Scheduling*, June 2016, pp. 527–535.

[175] Edmond Optics, "Imaging resource guide," [Online]. Available: https://www.edmundoptics.com/resources/industry-expertise/imaging-optics/imaging-resource-guide/, Accessed on: April 13, 2017.

[176] R. Zhou and E. A. Hansen, "Beam-stack search: Integrating backtracking with beam search." in *Proc. of The Int. Conf. on Automated Planning and Scheduling*, Jun. 2005, pp. 90–98.

[177] D. Jones, M. J. Kuhlman, D. A. Sofge, G. A. Hollinger, and S. K. Gupta, "Stochastic optimization for autonomous vehicles with limited control authority," in *Proc. of the 2018 IEEE/RSJ Int. Conf. on Intell. Robot. and Syst. (accepted for publication).*

[178] D. B. Chelton, R. A. Deszoeke, M. G. Schlax, K. El Naggar, and N. Siwertz, "Geographical variability of the first baroclinic Rossby radius of deformation," *J. of Physical Oceanography*, vol. 28, no. 3, pp. 433–460, Mar. 1998.

[179] L.-Y. Oey, T. Ezer, and H.-C. Lee, "Loop current, rings and related circulation in the Gulf of Mexico: A review of numerical models and future challenges," *Circulation in the Gulf of Mexico: Observations and models*, pp. 31–56, Mar. 2005.

[180] J. Das, K. Rajany, S. Frolovy, F. Pyy, J. Ryany, D. A. Caronz, and G. S. Sukhatme, "Towards marine bloom trajectory prediction for AUV mission planning," in *Proc. of the 2010 IEEE Int. Conf. on Robot. and Autom.*, pp. 4784–4790.

[181] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic trajectory optimization for motion planning," in *Proc. of the 2011 IEEE Int. Conf. on Robot. and Autom.*, May 2011, pp. 4569–4574.

[182] D. Jones and G. A. Hollinger, "Planning energy-efficient trajectories in strong disturbances," *IEEE Robot. Autom. Lett.*, vol. 2, no. 4, pp. 2080–2087, 2017.

[183] L. Panait and S. Luke, "Ant foraging revisited," in *Proc. of the Ninth Int. Conf. on the Simulation and Synthesis of Living Systems*, Sept. 2004.

[184] D. E. Gierl, L. Bobadilla, O. Sanchez, and S. M. LaValle, "Stochastic modeling, control, and verification of wild bodies," in *Proc. of the 2014 IEEE Int. Conf. on Robot. and Autom.*, May 2014, pp. 549–556.

[185] S. Bhattacharya, "Approximate structure construction using large statistical swarms," arXiv:1706.03842(cs:RO), 2017.

[186] G. A. Jacobs, H. S. Huntley, A. Kirwan, B. L. Lipphardt, T. Campbell, T. Smith, K. Edwards, and B. Bartels, "Ocean processes underlying surface clustering," *J. of Geophysical Res.: Oceans*, vol. 121, no. 1, pp. 180–197, 2016.

[187] *GEOS - Geometry Engine, Open Source*. [Online]. Available: https://trac.osgeo.org/geos/. Accessed on: February 10, 2018.

[188] S. T. Leutenegger, M. A. Lopez, and J. Edgington, "STR: a simple and efficient algorithm for R-tree packing," in *Proc. 13th Int. Conf. on Data Eng.*, Apr 1997, pp. 497–506.

[189] J. Kirk. *Multiple Traveling Salesmen Problem - Genetic Algorithm*. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/19049. Accessed on: April 10, 2018.

[190] D. R. Robinson, R. T. Mar, K. Estabridis, and G. Hewer, "An efficient algorithm for optimal trajectory generation for heterogeneous multi-agent systems in non-convex environments," *IEEE Robot. Autom. Lett.*, vol. 3, no. 2, pp. 1215–1222, April 2018.

[191] M. Tommaso, K. Hovannes, K. LiChung, and D. Emrecan, "Advances in underwater acoustic networking," in *Mobile Ad Hoc Networking*. Piscataway, NJ, USA: IEEE Press, 2013, ch. 23, pp. 804–852.

[192] M. Otte, M. J. Kuhlman, and D. Sofge, "Multi-robot task allocation with auctions in harsh communication environments," in *Proc. of the Int. Symp. on Multi-Robot and Multi-Agent Systems*, Dec. 2017, pp. 32–39.

[193] L. Bölöni, D. Turgut, S. Basagni, and C. Petrioli, "Scheduling data transmissions of underwater sensor nodes for maximizing value of information," in *Proc. of the 2013 IEEE Global Communications Conf. (GLOBECOM)*, pp. 438–443.

[194] D. Janzen and H. Saiedian, "Test-driven development concepts, taxonomy, and future direction," *Computer*, vol. 38, no. 9, pp. 43–50, Sept 2005.

[195] N. E. Fenton and M. Neil, "A critique of software defect prediction models," *IEEE Trans. Softw. Eng.*, vol. 25, no. 5, pp. 675–689, 1999.

[196] D. Walter, H. Täubig, and C. Lüth, "Experiences in applying formal verification in robotics," in *Proc. of the Int. Conf. on Computer Safety, Reliability, and Security*, 2010, pp. 347–360.

[197] G. E. Mullins, P. G. Stankiewicz, R. C. Hawthorne, and S. K. Gupta, "Adaptive generation of challenging scenarios for testing and evaluation of autonomous vehicles," *J. of Systems and Software*, vol. 137, pp. 197 – 215, 2018.

[198] A. Farchy, S. Barrett, P. MacAlpine, and P. Stone, "Humanoid robots learning to walk faster: From the real world to simulation and back," in *Proc. of the 2013 Int. Conf. on Autonomous Agents and Multi-agent Systems*, pp. 39–46.