Decision Uncertainty Minimization and Autonomous Information Gathering

by

Lawrence A. M. Bush

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

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2013

Author ....
Department of Aeronautics and Astronautics
August 22, 2013

Certified by ....
Brian Williams
Professor of Aeronautics and Astronautics
Massachusetts Institute of Technology
Thesis Supervisor

Certified by ....
Sridhar Mahadevan
Professor in Computer Science
University of Massachusetts, Amherst

Certified by ....
Pierre Lermusiaux
Doherty Associate Professor in Ocean Utilization
Massachusetts Institute of Technology

Accepted by...... ....
Eytan H. Modiano
Associate Professor of Aeronautics and Astronautics
Chair, Committee on Graduate Students

Decision Uncertainty Minimization

and

Autonomous Information Gathering

by

Lawrence A. M. Bush

Submitted to the Department of Aeronautics and Astronautics
on August 22, 2013, in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

Abstract

Over the past several decades, technologies for remote sensing and exploration have become increasingly powerful but continue to face limitations in the areas of information gathering and analysis. These limitations affect technologies that use autonomous agents, which are devices that can make routine decisions independent of operator instructions. Bandwidth and other communications limitation require that autonomous differentiate between relevant and irrelevant information in a computationally efficient manner.

This thesis presents a novel approach to this problem by framing it as an adaptive sensing problem. Adaptive sensing allows agents to modify their information collection strategies in response to the information gathered in real time. We developed and tested optimization algorithms that apply *information guides* to Monte Carlo planners. Information guides provide a mechanism by which the algorithms may blend online (realtime) and offline (previously simulated) planning in order to incorporate uncertainty into the decision-making process. This greatly reduces computational operations as well as decisional and communications overhead.

We begin by introducing a 3-level hierarchy that visualizes adaptive sensing at synoptic (global), mesocale (intermediate) and microscale (close-up) levels (a spatial hierarchy). We then introduce new algorithms for decision uncertainty minimization (DUM) and representational uncertainty minimization (RUM). Finally, we demonstrate the utility of this approach to real-world sensing problems, including bathymetric mapping and disaster relief. We also examine its potential in space exploration tasks by describing its use in a hypothetical aerial exploration of Mars. Our ultimate goal is to facilitate future large-scale missions to extraterrestrial objects for the purposes of scientific advancement and human exploration.

Brian Williams
Title: Professor of Aeronautics and Astronautics
Massachusetts Institute of Technology
Thesis Supervisor

Acknowledgments

Graduate research never occurs in a vacuum. I would like to express my deep gratitude to members of both my professional and personal community who supported me during the completion of this thesis. My foremost supporter in the professional realm was my advisor Brian Williams who has offered encouragement, guidance and many other forms of support. Brian facilitated my collaborations with researchers at MBARI and the NASA Ames Research Center and has played an active role in my career development, especially in the areas of grant writing and funding acquisition. The amount of time Brian contributed to developing the concepts in this thesis, improving my writing skills and telling me funny stories is a debt that I hope I will have the opportunity to repay in future collaborative efforts. My minor advisor Patrick Winston has always supported and inspired me, especially during coursework with him, in which I learned oral and written presentation skills. Committee members Sridhar Mahadavan and Pierre Lermusiaux provided encouragement and helpful advice through all stages of my research.

I am also grateful to Richard Lippmann and Sai Ravela for their service as thesis readers. Early in my career, through his course Rich provided me with an unbelievably thorough background in pattern recognition. I was also inspired by his enthusiasm and the generosity with which he gave his attention to my research efforts. Sai Ravela taught me about uncertainty quantification and provided invaluable discussions on the subject as a reader. Sai's good nature and intelligence has also offered me a great example of collegiality at its best.

I have also enjoyed collegial relations with my fellow students including Alborz Geramifard, Tony Jimenez, Andrew Wang, Eric Timmons, James Patterson, James McGrew and Lars Blackmore. My peers Pedro Santana, Shawn Kraut and Ben Landon have contributed by reading and offering feedback on this thesis. Julie Shah provided perspective, moral support and strategy throughout my graduate career. I am grateful to Robert Morris for hosting me at the NASA-Ames Research Institute and to Nico Mealeau for his mentorship there at Moffett Field. Our interactions inspired my interest in active sensing, space applications and reinforcement learning. Kanna Rajan collaborated with and hosted me at

MBARI (the Monterey Bay Aquarium Research Institute) where I also had the privilege of working with David Caress in field testing my algorithm on their autonomous underwater vehicle. I also wish to acknowledge my former boss, Joe Chapa for always recognizing the forward looking and innovative aspects of my work. I was also inspired by his unwavering willingness to stand up for his employees. I also fondly recall Douglas Marquis, another former boss, who encouraged me to "go put a Dr. in front of [my] name."

I am deeply grateful to the Lincoln Scholars program and Ken Estebrooke for financially supporting the greater part of my Ph.D. career. [1] I will always appreciate Bob Shin and Lou Bellaire for continuing to have faith in me and support me to the end of my PhD. Karen Wilcox and Brian Williams also helped me identify sources of financial support. Bernadette Johnson and Andy Vidan who supported me in numerous ways, especially during some uphill sections of my journey. Sara Peterson, our group's administrative assistant has been generous and diligent in helping me put out fires and find my way around the Institute. My graduate studies have also provided me with a number of enriching opportunities, not directly related to uncertainty minimization. Over the past 3 years I have mentored 6 RSI (Research Science Institute) students: Changlin Ke, Lukas Missik, Albert Chu, Maxwell Wang, Jon Xia and Fredric Moezinia. These exceptional 12th graders each came to MIT for 6-weeks of a summer during which I had the privilege of sharing my research with them. They were always elated by the opportunity, and fanatical in their efforts. Their zeal carried me through some of the less inspiring parts of my research.

My brother Chris has happily answered many statistics questions for me and supported me like only a brother can. Finally I am most grateful to my loving and supportive wife, Nancy, and our three wonderful children who are a deep inspiration to me, and are the real secret of my success.

[1] This work is sponsored by the Department of Air Force under Air Force Contract #FA8721-05-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the author and are not necessarily endorsed by the United States Government

1 Forward

This thesis addresses the integration of data collection and processing needs of orbital and airborne missions. The motivating purpose is to develop a consistent strategy for exploration in support of long-term human missions to terrestrial planets. The thesis specifically addresses the informational requirements for large-scale adaptive sensing missions on Earth and Mars. This research addresses several remote adaptive sensing concepts through ocean sensing, disaster response and Mars exploration applications. Using these tasks as a framework, the thesis lays out a quantitative model for how to make decisions to gather information intelligently and dynamically. The primary objective of this thesis is to introduce algorithms that improve the quality of automated decisions made in real time and to incorporate uncertainty terms into informational value estimates used in remote sensing. The thesis makes a critical contribution by developing algorithms that address information and decision uncertainty, and using them to define an information-gathering architecture. The central application question is: how to autonomously navigate and characterize an unknown space such as a planetary surface?

1.1 Motivation: Greater Automation in Sensing and Exploration

The need to gather information more intelligently and navigate complex state spaces becomes more pronounced as humans create more powerful remote sensing devices and extend the reach of their exploration activities. Adaptive sensing is a form of remote sensing that collects information selectively, according to a policy that can incorporates information as the task progresses (see Sec. below). Adaptive sensing can modify its information collection strategy as more information comes in, so as to moderate overall informational possibilities that unfold with devices such as drones, smart phones, GPS, and robots. Identifying the valuable information in a large state space means sorting through possible actions and sensing tasks that include relatively large uncertainties. This kind of strategic information gathering and processing can benefit larger scale exploration tasks by automating environmental characterization. Exploration also requires a method of using current information to sort through alternative actions and identify the most valuable action according to mission objectives. This thesis addresses those needs in anticipation of expanded remote sensing and exploration activities in the coming decades. This research is also applicable to many Earth-based applications, but planetary exploration offers some of the greatest rewards, including potentially habitable environments and even evidence of life outside of Earth. As pointed out in a quote that's often attributed (with some uncertainty) to Carl Sagan,"somewhere, something incredible is waiting to be known."

The successes celebrated by ESA, NASA over the last decade, and more recently by the Chinese National Space Agency (CNSA), and commercial space travel firms have demonstrated new possibilities in planetary exploration, and a global competitive interest in the endeavor. It is reasonable to assume that some combination of energy-related economic incentives, scientific impetus and perhaps even a geopolitical catalyst (akin to the Cold War space race) will induce humans to continue and expand their exploration activities solar system. The technological challenges of this eventuality require us to begin developing the technology that automates this process wherever possible, making it cheaper and faster, and circumventing current exploration bottlenecks. Adaptive sensing offers extensive reconnaissance of promising objects (both planets and moons) and their environs. Europa, Titan and Mars are three objects that hold particular promise as potentially habitable environments.

Jupiter's moon Europa (Figure 1-1) was discovered by Galileo in 1610 and is slightly smaller than Earth's moon. Investigation by the *Pioneer*, *Voyager*, *Galileo* (NASA) and *New Horizons* (ESA) space probes indicate that Europa includes a metallic core and a rocky mantle that are enveloped by a deep, salty, ice-covered surface ocean. These inferences have rendered Europa the subject of many discussions concerning future mission and potential habitability. Numerous studies and proposed missions seek to explore the subterranean oceans of Europa, using submersibles equipped with active sensing algorithms.

Titan (Figure 2-5) is Saturn's largest moon and includes a dense atmosphere with stable bodies of liquid hydrocarbons on its surface. While Titan may be habitable for microbial life, its surface liquids also suggest potential economic incentives for human space exploration. Titan may serve as an important energy reserve in our solar system. Ligeia Mare, a liquid body in Titan's north polar region, is larger than Lake Superior, (Figure 2-5) and consists of a liquid mixture of methane and ethane according to data returned in 2006 by NASA's *Cassini* mission.

Figure 1-1: Europa, one of the four Galilean moons of Jupiter, has been the subject of extensive satellite-based research. The first photos of Europa were taken by Pioneer 10 and 11 in the early 1970s; the most recent data were collected by the Galileo probe, which crashed into Jupiter in 2003. NASA made the decision to direct it into the Jovian atmosphere to avoid the risk of it carrying terrestrial bacteria into the seas of Europa.

Mars (Figure 1-3), the fourth planet from the sun, was named for its red color which evokes the Roman god of war. The planet's landscape is characterized by volcanic features, impact craters and a giant rift likely caused nascent extensional forces, the Valles Marineris. Recent orbital and landed missions to Mars have demonstrated that liquid water once flowed on Mars and had a similar chemistry to the liquid water found on Earth. Simple organic molecules have also been detected on Mars. The Mars Curiosity Rover is currently exploring the Gale crater, with the broad directive of looking for signs and conditions for life.

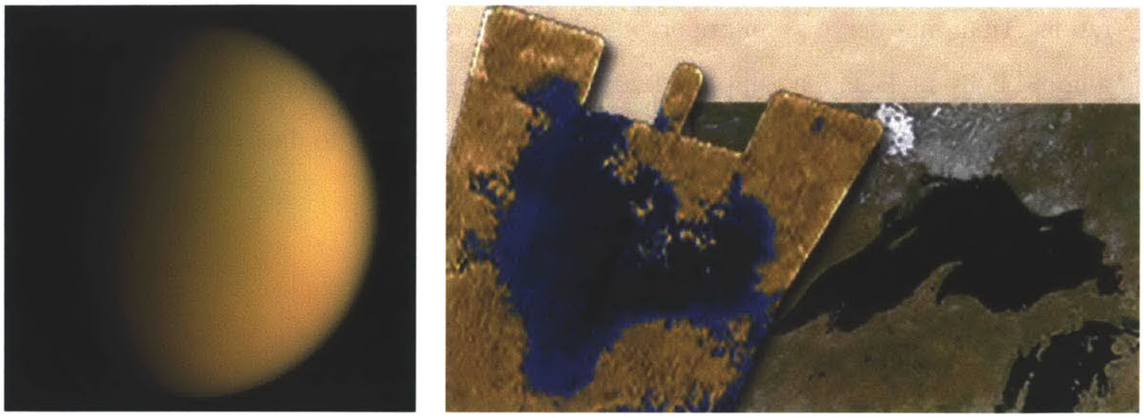


Figure 1-2: As seen in the image on the right, Titan's Ligeia Mare is larger than Lake Superior, and filled with liquid hydrocarbons (methane and ethane). Interestingly, there is some dispute over the age of Titan due to its low crater count. A low crater count could indicate that the planet's surface was refreshed by a natural process, or that it is simply young.

Figure 1-3: Nineteenth-century astronomical observations, made with low-powered telescopes, led to widespread speculation that the Red Planet was not only habitable, but inhabited. More recent data, including samples taken by the Mars Curiosity Rover, have found evidence that liquid water once flowed across the planet's surface, and its polar caps are composed largely of water ice.

1.2 Current Technological Challenges

In the current age of planetary exploration, landed and orbital missions provide data for Earth-based analysis. Few decisions are automated in this framework. Agency planning indicates a transition to sample return missions and eventual human exploration. Human exploration will require more detailed planetary surveys using orbital, airborne, submersible and landed missions. One of the novel challenges that has arisen during the on-going MSL mission is that the shear volume of data sent by *Curiosity* is overwhelming the current scientific team assigned to processing and interpreting it. Future missions will thus require a more integrated and automated exploration model that differs from the current system in its framing of mission objectives, operational architecture and its use of algorithms.

The task of automated exploration specifically calls for an architecture and algorithms that operate from different vantage points and at different scales to achieve informational goals. The architecture and algorithms can both simulate alternatives and adapt to a changing informational landscape. The planetary survey is framed as a large and complex control space addressed through both online and offline analysis. [1] The survey control space behaves in a stochastic manner due to the evolving informational landscape. We do not know what information will be collected, nor do we know its value in terms of the mission's informational goals. Uncertainties however can be quantified and minimized by algorithms. The architectures thus guides the algorithms through an evolving landscape of stochastic rewards, which is typically the value of information relative to informational goals.

1.2.1 Technical Problem

How do we improve automated decision-making given real-time constraints? This thesis focuses on decision-making for remote sensing surveys, the problem we address can be classified as an active sensing task. Active sensing addresses the problem of collecting the right information in an environment of infinite informational possibilities. This is an application of adaptive sampling techniques, applied to real-world problems of remote-sensing

[1] Online refers to real-time simulations and action in the real-world environment whereas offline refers simulated action conducted prior to entering the environment.

data collection similar to that described in Krause and Guestrin [2009]. It mathematically defines the optimal solution of the active sensing problem. The findings of this thesis are therefore relevant to anyone who would like to optimize according to an informational criteria.

Exploration tasks applied to large-scale environments also face an infinite number of possible actions to take. How do we decide among these alternatives using information gathered from the environment and simulated through learning models? This thesis describes an architecture for the active sensing problem, and goes on to design and test several appropriate optimization algorithms using terrestrial information-gathering applications and a hypothetical aerial survey of Mars. Here, we apply an algorithmic mechanism we call *information guides* to a Monte Carlo search algorithm. Decisions tend to focus on the known, and assume that the optimal path of action is evident from the value function. Information guides explicitly incorporate the unknown into decision-making and offer alternatives when the optimal path is not evident. Information guides use uncertainty concerning an observation derived from a posterior distribution (or current reality as we know it) to guide the search through a set of actionable alternatives in a given situation. This method offers a novel way of applying empirical observations to reoptimize the value function concerning a particular decision (see below). Previous research concerning decision making in uncertain environments using value functions and simulated models is summarized in Sec. .

Monte Carlo (MC) information guides

Adaptive sampling is the mathematical approach to selecting the right samples (see Guestrin *et al.* [2005] and Bush *et al.* [2008a]). In this thesis we develop and test appropriate optimization algorithms for solving the adaptive sampling problem. The thesis describes a novel algorithmic mechanism we call information guides. Information guides are applied to a Monte Carlo planning algorithm and help integrate online and offline computer simulation models. The mechanism and applications described here demonstrate the key innovation of approximate planning in real time for remote sensing operations.

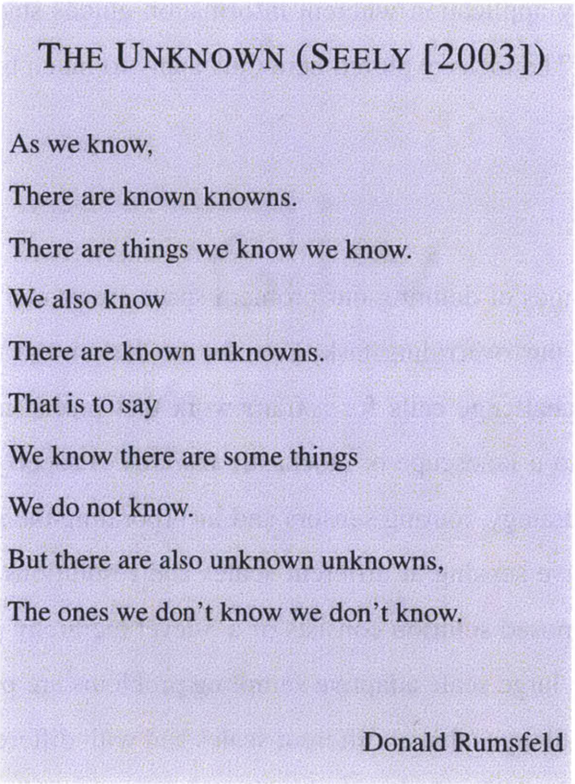
Information guides specifically quantify uncertainty implicit in a given current decision and compare it to the uncertainty of alternatives. We assume that the 'correct' or most desirable decision offers greater rewards than an alternative decision. Information guides additionally may reveal alternatives that hold promising outcomes. How do we avoid subjective resolution of these different outcomes? Given the right inputs and software, a computer can quantify the alternatives assigning them values that can be directly compared. Online search is an everyday application wherein information guides steer search algorithms to appropriate content. The ideas explored here thus apply to much broader information and planning applications.

Architecture

The technical challenges of defining our problem space are primarily ones of architecture and framing. Given our overriding task, how do we design and frame a complex planetary survey? This challenge calls for a framework that meets informational needs and dynamically adapts to a landscape of stochastic rewards. Properly framing this problem means defining our strategy, routing sensors and incorporating the information they gather into our model. Active sensing at different scales and resolutions also calls for different approaches. Our proposed solution consists of a 3-level hierarchy presented in Chapter 2, which describes how large scale adaptive sampling problems are parsed and framed so as to address unique challenges that at different scales and with different types of sensors.

1.3 Unifying Theme: Incorporating uncertainty reduction into decision making

The unifying theme of this thesis is uncertainty quantification and minimization. Exploration seeks to reduce or constrain what we do not know. As we explore, our knowledge about previously unknown areas and phenomena increases, while our uncertainty decreases. In a more specific context, a planetary survey must focus on mission objectives such as finding habitable conditions on another planet. The principle of guidance by mission objective is also applicable to other levels of decision-making. Exploration requires a clear and quantitative (wherever possible) understanding of what we know and its associated uncertainty. Only then can we proceed upon a course of action for improving our understanding. Likewise, as we explore the unknown, we gain a better awareness of the existence of things we do not know. This idea is captured in the following statement made by U.S. Secretary of Defense Donald Rumsfeld during a press briefing in February of 2002:



THE UNKNOWN (SEELY [2003])

As we know,
There are known knowns.
There are things we know we know.
We also know
There are known unknowns.
That is to say
We know there are some things
We do not know.
But there are also unknown unknowns,
The ones we don't know we don't know.

Donald Rumsfeld

These remarks were controversial at the time given public concern about the quality of planning that went into the invasion and occupation of Iraq. The actual statement itself was a concise and accurate description of the situation (if not a little too honest for a press briefing) and incisively conveys the nature informational uncertainty and how we conceptualize parameters from prior distributions. There is uncertainty in our knowledge with respect to the current situation, i.e., the state (the known unknowns). The existence of this uncertainty and our acknowledgment that the state is not fully represented in our current understanding indicates additional uncertainties (the unknown unknowns). Accurate representation of uncertainty in our situational model or state enhances mission decision-making. We can thus use knowledge more effectively when we fully understand the nature of its uncertainty.

We can also use properly characterized uncertainty to determine where to focus future data collection efforts.

Uncertainty reduction can also be used to enhance specifications of mission objectives. This functionality arises from how we assign a value to minimum success criteria. We can quantify for direct comparison, the outcomes from a strategy or path of action as well as those of an alternative strategy, including uncertainty for both. We then use that awareness to simulate through viable alternatives, potentially enhancing our understanding of these alternatives, and identifying the most rewarding path of action. In Chapter 3, we introduce Algorithm (Sec. 3.8) and specify how it performs these operations. The process of reducing the uncertainty associated with our optimal decision is guided by decision uncertainty information.

Kurt Gödel, the parent of modern mathematical logic, searched for certainty but could only prove uncertainty, thus proving that there are some things we can never know for sure. Given its persistence, I prefer the approach of embracing uncertainty at every level in problem solving. In doing so, we make better decisions with respect to some of humankind's greatest challenges and endeavors.

1.4 Tasks: Enhancing exploration strategies and addressing uncertainty

This thesis contains the following elements and discussions intended to enhance exploration strategies and address uncertainty:

- Formulates active sensing at each level as a Markov Decision Process.
- Develops a new Monte Carlo planning algorithm using information guides
- Describes a 3-level hierarchy architecture for active sensing problems
- Demonstrates the functionality of the architecture by applying it to marine exploration and disaster relief operations

- Outlines rationale for a compact representation and develops a discovery method that includes uncertainty representation

1.4.1 3-level hierarchical architecture for active sensing problems

The 3-level Hierarchy

Real world active sensing problems include planetary exploration and Earth-based endeavors such as submarine exploration and disaster relief. These cover large geographic areas, incorporate many different types of information and thus require multiple sensor types. The complexities of the real world require a sophisticated, multi-scale architecture.

Figure 1-4 shows a 3-level hierarchy designed for exploration of an object like Europa. The 3-level hierarchy includes synoptic, mesoscale and microscale aerial designations. The synoptic level covers the maximum area addressed in the survey, which could be planetary in scale, or the scale of an ocean basin or the combined area of several states affected by a disaster. This scale would range in practice from about 10^3 to 10^6 km (in the case of Europa). The mesoscale level covers an intermediate area, and the microscale level covers the smallest area that can be addressed by the algorithms described here, in a non-trivial manner. A microscale sensor would cover an approximate area of a square kilometer. The mesoscale sensor would thus cover areas ranging from 10 - 10^3 km. In the applications described in Chapter 2, sensor coverage at the synoptic level is provided by a satellite. Coverage at the mesoscale and microscale levels is provided by an autonomous aircraft or submersible. The system works as follows: The satellite coarsely surveys a large swath of Europa's surface. That information is processed and relayed to a group of vehicles which then deploy to the areas that are most likely to meet mission objectives. The vehicles survey the areas and return information for the next iteration of discovery.

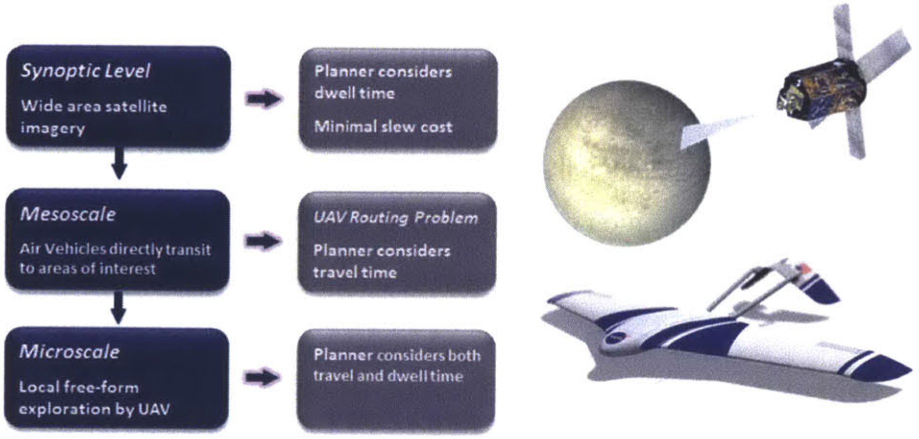

Synoptic Level
Wide area satellite imagery
Planner considers dwell time
Minimal slew cost
Mesoscale
Air Vehicles directly transit to areas of interest
UAV Routing Problem
Planner considers travel time
Microscale
Local free-form exploration by UAV
Planner considers both travel and dwell time

Figure 1-4: The 3-level hierarchy shown above includes a synoptic level (covering a wide area), a mesoscale level (covering an intermediate scale area) and a microscale level (covering a small local area). The synoptic sensing (satellite) optimization algorithm concentrates on dwell time and the value of the collected information. The mesoscale system is optimized with respect to travel time (e.g., for autonomous unmanned aircraft or other sensors). Travel and dwell time are simultaneously adjusted at the microscale to optimize free-form exploration by sensing devices.

1.4.2 Mapping each level of the architecture as a Markov Decision Process

This thesis describes two detailed Earth-based applications of the 3-level architecture, an algae bloom ocean sensing project and disaster relief. The first application involves autonomous underwater vehicles (AUVs) collectively mapping the biological phenomena of algae blooms. The second application involves air vehicles surveying a disaster area in order to reporting road conditions and transportation related information to a ground crew assigned to locate and rescue survivors. In both applications, the mapping response takes place at two levels of abstraction. At one level, the vehicles are assigned and routed to the mapping locations. The second level is a sensing platform that collates the data and takes steps to maximize the amount of useful information collected. Operations from both levels can be mapped into a special type of Markov decision process, referred to as an information state Markov decision process (info-MDP).

The info-MDP formulation is especially suitable for information gathering problems because these situations are inherently a problem of imperfect information. The core objective in other words, is to improve the information state. This process can then be reduced to a state of perfect information that includes statistics measuring the available observations. Given this formulation, paths of action can be chosen according to the available observations. Similar info-MDP applications can be found in Liu [2013] and Bush *et al.* [2008a]). Section 2.3 offers a full definition of an info-MDP. Chapter 4 describes how an info-MDP can be applied to an active sensing problem.

1.4.3 Approximate inference driven by decision uncertainty minimization

With each level of the problem framed as an info-MDP, we can then use dynamic programming, and specifically *approximate dynamic programming* (ADP) to solve the problem. Solving the problem in this case means defining a consistent policy that will control the data collection system. Chapter 2 presents a series of algorithms developed for this purpose, and describes a framework for applying these algorithms. We initially use ADP to develop a policy for how a sensor navigates through space and searches for information, assuming basic informational goals (i.e., accurate navigation, finding evidence of water on Mars, etc.). The second step entails modifying that policy online, during operational phases of the mission, using forward search. The online search sets initial conditions according to initial observations of the mission and searches forward through possible outcomes. The search space can be narrow or broad, shallow or deep (i.e., set according so scale level and mission objectives). The process uses the uncertainty about the expected value of each search branch to decide which actions are worthy of simulating. These information guides point the search toward promising path of action or avenues that have a high probability of yielding the best outcome. This process finally leads to use of Algorithm (see Sec. 4.6), *decision uncertainty minimization via time limited branching*. This algorithm applies state action value iteration with variable depth information guides and a Gaussian Process based uncertainty representation.

Figure 1-5 illustrates how decision uncertainty minimization via time limited branching (Algorithm) works. The search is set according to scale level and mission objectives,

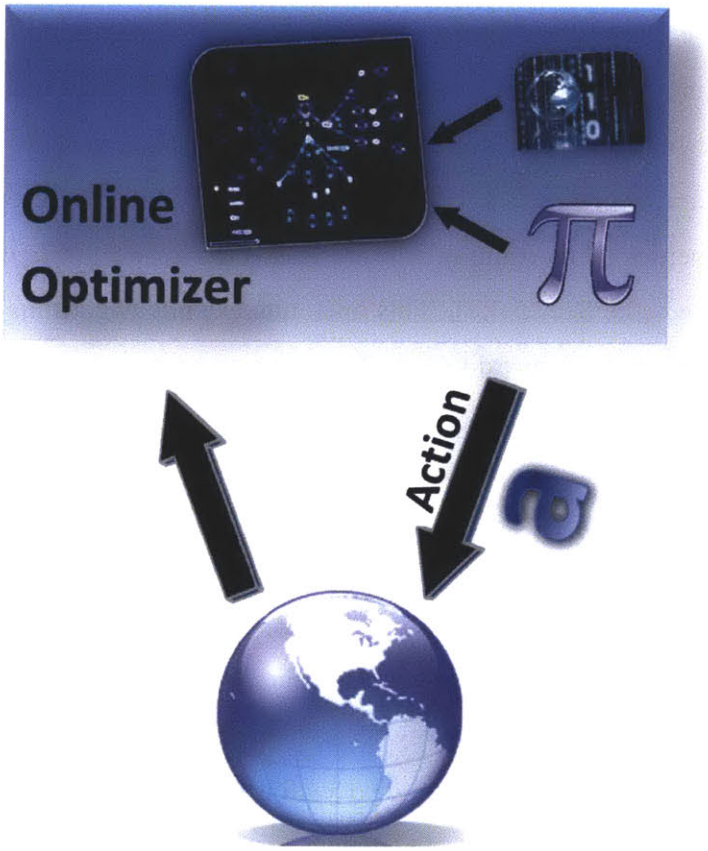


Figure 1-5: Online reoptimization algorithm : The online optimizer starts with policy π, which was produced offline. Combined with the world simulation, we can reoptimize for the current situation. For the given situation, policy π can convey its confidence in choosing one action over another, as well as which action value is most in question. Using this information as a guide, the online optimizer will simulate that action followed by a series of subsequent actions (chosen in the same manner) in order to re-estimate the value of that trajectory.

across a search space that can be narrow or broad, shallow or deep. This process uses the uncertainty about the expected value of each search branch to decide which actions are worthy of online simulation. Information guides point the search towards promising paths of action or avenues that have a high probability of yielding the best outcome.

Chapter 4 also describes two precursors algorithms related to Algorithm . Algorithm is referred to as the *closed form least squares value iteration* algorithm (see Sec. 5.1.5). This operation exploits the certainty equivalence property of the root mean squared error uncertainty metric in a way that accelerates the policy search algorithm. The root mean squared error uncertainty metric is a consistent alternative to Shannon entropy but has better mathematical properties. Algorithm is referred to as the *closed form least squares value iteration with rollout* (see Sec. 5.1.7). This operation extends Algorithm by employing a deep limited breadth online forward search. These two algorithms were natural extensions of Algorithm .

1.4.4 Demonstrating the framework in marine exploration and disaster relief applications

Highly automated, intelligent devices such as drones, submersibles and landers like *Curiosity* have tremendous potential to survey and explore. Exploring unknown areas involves sensing the environment with measuring devices for the purposes of scientific discovery, reconnaissance and logistical support. In Chapter 4, we frame the active sensing problem as a 3-level hierarchy (see Sec. above) and formulate it as an information state Markov decision process. The chapter goes on to demonstrate this framework in a scientific research campaign and in mission support operations. The algorithms described in Chapters 3 and 4 are specifically applied and tested to a data collection effort addressing a marine algal bloom and a disaster relief scenario. Both of these applications pertain to planetary exploration technology, which may include exploring the oceans of Europa and liquid bodies of natural gas on Titan.

1.4.5 A Compact representation and a discovery method that incorporates uncertainty

Decision uncertainty minimization depends on representation that is both accurate and conveys that accuracy as value prediction uncertainty. This in turn requires an approach to value function approximation with low uncertainty representation that also captures and expresses the uncertainty that remains (the unknown unknowns). This latter form of uncertainty is an important component of the overall certainty of our prediction. Identifying a solution or representation that minimizes uncertainty depends on accurate uncertainty estimates.

The utility of a value function distribution strongly depends on its having a manageable degree of variance. Excessive variance renders the 'guidance' steps ineffective and leads to a device response that is essentially random. Luckily the inaccuracy of a value function is a causal phenomenon: it stems from the underlying representation and can be modeled. While state-action value functions do not typically include expressions of representational uncertainty, our method constructs functions that can be assessed as 'more sure' in some places and 'less sure' in others. This method provides a representation that a) accurately represents the value function and therefore has low uncertainty and b) given whatever uncertainty is present, captures that uncertainty about the posterior value estimate. In this way, we use the uncertainty instead of ignoring it.

Chapter 3 describes how to find a value function representation with low posterior uncertainty relative to the value estimate. An online search algorithm that is guided by the uncertainty in the state-action value function requires a function that represents the uncertainty in its estimate. We also need a way to minimize the uncertainty function. To these ends, Chapter 3 outlines the design of such a value function. Specifically, we use non-negative matrix factorization to find good basis functions that capture the structure of the underlying data. We then use a Gaussian process to map the distribution over different value ranges. We integrate these complementary methods with the overall objective of representational uncertainty minimization.

1.5 Prior Research and Thesis Overview

1.5.1 Prior Research

Previous research in the field of information guides, active sensing and info-MDPs has provided a strong foundation for this project and demonstrated some additional areas of development addressed by this thesis. A number of the approaches such as approximate dynamic programming (ADP) and info-MDPs have been described and applied elsewhere (Bertsekas and Tsitsiklis [1996b] and Littman [1996]). However, the research presented in this thesis makes the novel contribution of combining several of these techniques and implementing them in navigational or exploration applications. In other areas such as decision uncertainty, the research described here presents new algorithms and approaches that have not been previously reported. Our initial efforts in combing offline simulation with online operations to complete adaptive sensing tasks were presented in Bush *et al.* [2009]. Previous researchers introduced the same approach of using ADP results to support online decision-making. Gelly and Silver [2007] considered several ways in which offline and online operations could be combined to improve value function learning. Boyan and Moore [2001] demonstrated iterative re-optimization of an MDP policy that improved search performance. The ADP methods used here to construct offline policies as well as an exhaustive list of re-optimization strategies are surveyed by Bertsekas [2005a].

Dearden *et al.* [1998] also performed traditional online reinforcement learning, using information guides (uncertainty about the value function) to improve policy learning. Their model conflates environmental uncertainty and uncertainty due to approximate inference. This approach differentiates between certainty and habitual actions in decision-making, while creating an inverse model of the environment online (see Dearden *et al.* [1998]). In contrast, our model-based approach separates these functions, using a previously learned model, and a previously learned policy, and then tailors the policy online to the particular situation. Kocsis and Szepesvari [2006] also demonstrated the use a crude form of state-action value uncertainty in guiding an online Monte Carlo sampling process. These workers (Kocsis and Szepesvari [2006]) presented an algorithm that retains information from previous states sampled to estimate uncertainty and formulate decisions. The entirely online method and simple state space representation outlined in Kocsis and Szepesvari [2006] differs from the methods described in Chapters 2 and 3. MacKay [1992b] uses a Bayesian active learning approach related to information guides to select training samples for a neural network, and thus provided our research with ideas and approaches concerning sample selection. MacKay [1992b] is an application agnostic sample selection idea for the purpose of providing data to a function. Other methods in the reinforcement learning community have developed action selection methods for the purpose of *in situ* learning. These methods attempt to optimize an exploration-exploitation tradeoff (Brafman and Tennenholtz [2003],Meuleau and Bourgine [1999],Auer [2003]). These works have informed our work, as we are making a related tradeoff. However our work differs from these methods namely because our method is model-based and we learn our policy offline.

Because active sensing has so many real-world applications, numerous studies have sought to automate and develop models for adaptive sensing tasks. The works most relevant to this thesis have focused on MDPs, state-space representations and how to measure the real world environment. Murphy [1999] describes how sensor information can be used to create and update a map of a given area, and then fed into a decision-making framework concerning a given area. Thrun [2003] explored the use of occupancy grid in adaptive sensing and introduced methods similar to those outlined in Chapter 2 of this thesis. The literature concerning the use of info-MDPs in active sensing is limited and relatively recent. Mihaylova *et al.* [2002] used partially observable MDPs (POMDPs) to optimize active sensing algorithms and points out computational difficulties that may arise from applying MDPs to large state spaces. Ahmad and Yu [2013], Liu [2013] and Bush *et al.* [2008a] also use POMDPs in active sensing.

1.5.2 Thesis Overview

The remaining body of this dissertation is organized as follows:

Chapter 2 defines active sensing and applies the aforementioned algorithms into the context of two active sensing tasks. The chapter describes a hierarchical mission architecture based according to different scales at which exploration tasks are carried out. We define information state Markov Decision Processes (info-MDPs) and frame active sensing tasks as MDPs. Tasks at each hierarchal level are specifically formulated as info-MDPs. Chapter 2 also demonstrates the performance improvement of the three level approach over a single level, micro scale approach. A marine bathymetric mapping task and a hypothetical Mars exploration mission are framed as info-MDPs using this 3-level hierarchy.

Chapter 3 introduces the new algorithm of decision uncertainty minimization via time limited branching. Several precursor algorithms are discussed as background to the primary algorithm. Chapter 3 defines and discusses the prior art of exact value iteration with value functions and with Q-values, approximate Q-value iteration, Monte Carlo planning in an MDP, open-loop feedback control, certainty equivalence planning and rollout. Chapter 3 also defines decision uncertainty minimization and puts it into the context of online and offline approximate dynamic programming. The chapter then discusses quantifying the uncertainty of a policy, and the dependence of its accuracy on the uncertainty representation. It then discusses the new idea of using representational uncertainty to direct the blending of online and offline computation.

Chapter 4 defines representational uncertainty and outlines the rationale for a low uncertainty representation that also captures the uncertainty that remains following observation. The use of value function uncertainty is discussed in the context of blending online and offline decision methods. The techniques discussed include basis function discovery, linear architectures, transition matrix factoring, and uncertainty directed transition matrix factoring.

Chapter 5 describes active sensing applications using the 3-level hierarchy in greater detail. The chapter describes an active sensing task that occupies the synoptic level of the hierarchy and a task that occupies the microscale level of the hierarchy. The results demonstrate the utility of the presented methods. [Additional copy needed]

Nomenclature

Markov decision process

$\langle \bullet, \cdots, \bullet \rangle$	=	tuple brackets : ordered heterogeneous non-removable set
$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$	=	Markov decision process model
$\mathcal{S}$	=	state set
$\mathcal{A}$	=	action set
$\mathcal{P}$	=	state transition probability distribution
$\mathcal{R}$	=	reward function
γ	=	discount factor
$\mathcal{P}^{a}_{ss'}$	=	probability of transitioning to state s' from state s when executing action a
$\mathcal{R}^{a}_{ss'}$	=	reward associated with transitioning from state s to state s' given action a

MDP Solutions

π^*	=	optimal policy, which maximizes the value (expected future discounted
	=	reward) of every state
V	=	refers to the ethereal & theoretical true value (expected discounted future reward) over the entire state space S
$V_{\pi}(s)$	=	average future rewards obtained when policy π is followed, starting in state s
$V^*(s), V_{\pi^*}(s)$	=	optimal value of state s
$\bar{V}(s)$	=	value lookup table *where* $s \in \bar{S} \subseteq S$
$\bar{V}$	=	refers to the subset of points $v \in \bar{V} \subseteq V$ pertaining to states $s \in \bar{S} \subseteq S$
$\widehat{V}$	=	value function approximation architecture
$\widehat{V}(s, a)$	=	approximated value
$V_{\pi}(s, a)$	=	true value under policy π
$V^*(s), V_{\pi^*}(s)$	=	optimal value
$\tilde{v} = \tilde{V}(s) \in \tilde{V}$	=	the improved value approximation obtained via a Bellman backup
$\bar{Q}(s, a)$	=	state-action value lookup table *where* $s \in \bar{S} \subseteq S$
$\widehat{Q}$	=	state-action value function approximation architecture
$\widehat{Q}(s, a)$	=	approximated state action value

$Q_{\pi}(s,a)$	=	true state-action value under policy π
$Q^{*}(s,a), Q_{\pi^{*}}(s,a)$	=	optimal state action value
$\tilde{Q}(s,a)$	=	the improved state-action value approximation obtained via a Bellman backup

Abbreviations

info-MDP	Information State Markov Decision Process
RUM	Representational Uncertainty Minimization
DUM	Decision Uncertainty Minimization
NMF	Non-negative Matrix Factorization
$\mathcal{GP}$	Gaussian Process

Gaussian Process

$cov(f_{*})$	Gaussian process posterior covariance

Function Approximation

$\sim$	=	distributed according to ; example $q \sim \mathcal{N}(\mu, \sigma)$
$\overset{obs}{\leftarrow}$	=	realization or observation of; example $b \overset{obs}{\leftarrow} B$, where b is a realization of random variable B

Glossary

3-level hierarchy A way of structuring and characterizing sensing problems at the global (synoptic), mesoscale (intermediate), and microscale (close-up) levels.

Action space The set of all possible actions $\mathcal{A} = a_1, a_2, ..., a_n$, e.g. in our problems, an action entails sensing at a particular grid location.

Active sensing The application of an adaptive sample design to a sensing task.

Adaptive sampling The mathematical optimization of information gathering.

Approximate dynamic programming (ADP) A method of developing control policies by approximating value functions, where those functions cannot be determined reliably.

Approximate value iteration An approach to learning the value function of a Markov decision process by combining function approximation and dynamic programming. This is one form of approximate dynamic programming.

Autonomy The ability of a robotic or software agent to choose its own actions, without the intervention of a human operator.

Base policy The default approach of a function or heuristic to a given problem.

Basis function An element of a basis for a function space, similar to a feature when independency is relaxed.

Bathymetric Relating to the study of underwater topography.

Belief map The probability that each space of an occupancy grid is inhabited. More generally, a representation of our belief about all of a map's parameters.

Belief or mental state A way of categorizing an autonomous agent's awareness of its uncertainty regarding a map of its environment, via a distribution over all physically possible world states.

Bellman optimality equation A necessary condition for optimality in dynamic programming applications. It describes a one-step look-ahead with respect to the value function.

Branching factor The number of successors of a particular node, in tree-based data structures.

Certainty equivalence approximation A method of reducing analytical complexity by assuming that the utility of the expected outcome is equal to the average utility over all possible outcomes.

Decision uncertainty minimization (DUM) A method of blending offline and online computation to improve realtime planning, by incorporating uncertainty into the online decisions regarding reevaluation of an offline policy.

Dynamic programming (DP) An approach to solving complex problems by breaking them into multiple subproblems, which are easier to solve.

Expected value In a Markov decision process, the expected value refers to the average discounted sum of rewards obtained by following a given policy from state (s).

Gaussian process A stochastic process in which random variables are associated with each point in a finite range, so that each variable has a normal distribution.

Information-state Markov decision process (info-MDP) An infinite state Markov decision process that makes decisions directly based on its belief about the world

Information-state partially observable Markov decision process (info-POMDP) A partially observable Markov decision process in which the reward is a function of the state estimate uncertainty.

Kalman filter A recursive algorithm that produce statistically optimal estimates of a system's state from observed sequential measurements, assuming a Gaussian linear model.

Linear architecture An architecture that approximates $V(s)$ by first mapping the state s to feature vector $\Phi(s) \in \Re^k$ and by computing a linear combination of those features $\Phi(s)\beta$, where β is a function parameter vector.

Markov decision process A decision making model where outcomes are ultimately stochastic yet an agent makes certain choices in an effort to maximize the *expected* discounted sum over future rewards.

Monte Carlo planning A method of comparing different policies by combining Monte Carlo analysis with forward search. Monte Carlo planning considers all actions at each state, but limits how thoroughly the action consequences are simulated.

Occupancy grid A mapping problem representation, where each cell in a grid conveys the likelihood that the phenomena of interest is present.

Offline planning The policies an agent learns before a mission begins, which are based upon simulated actions.

Online planning Policies generated in the field, based upon realtime simulations and actions in the real-world environment.

Open-loop feedback control An approach that generates an open loop (static non-reactive) plan for the mission duration, limiting the analysis by ignoring future contingent choices.

Partially observable Markov decision process (POMDP) A Markov decision process in which the underlying state is only partially observable, so that decisions must be made using the inferred probabilistic state estimate.

Policy iteration An algorithm that calculates a sequentially improving series of policies, through evaluation, and improvement.

Q-value The value (expected discounted future reward) of taking a particular action when in a given state of a Markov decision process.

Representational uncertainty minimization (RUM) A method using Gaussian processes to capture uncertainty and factored state transition matrix based features, with the goal of finding a state-action value function representation with low posterior uncertainty for use in decision uncertainty minimization planning.

Rollout A method of reevaluating mission planning options by simulating ahead from each possibility. Standard rollout evaluates only the first level in the policy tree, while nested rollout allows branching at multiple levels.

Sensing task A means of surveying an area and quantifying the relevant phenomena by measuring the environment.

State The current, future, or past condition of a given system.

State space The set of possible states in a dynamic system.

System dynamics A representation of the probability of transitioning from one state to another when taking an action. For example, in a sensing mission, where the map is a main part of your state, executing a sensing action will change your map and therefor your state, in some way.

Unmanned aerial vehicle (UAV) An autonomous or semi-autonomous vehicle for aerial survey and exploration.

Unmanned ground vehicle (UGV) An autonomous or semi-autonomous vehicle for ground-based survey and exploration.

Value function approximation A method of estimating a value function, where the true function cannot be accurately determined or represented.

Value function A function providing the expected discounted future reward to be gained by following a given policy.

Value The *expected* discounted future reward to be gained by following a given policy, from a given state. The term *value* in the context of an MDP solution always refers to an *expectation.* It is used here in the same way that value and valuation are used in finance, where they wish to determine the present value of an asset's expected future cash flows. The future cash flows may be stochastic, but the present value is not.

Vehicle routing problem with time windows (VRP-TW) A planning problem in which a number of vehicles must visit specific locations within a defined period of time.

2 Active Sensing

Recent technological advances have revolutionized robotics, data storage and communication in ways that call for new developments in automation. Semi-autonomous devices such as robotic landers, drones and even smart phones present us with extraordinary remote sensing and data collection capabilities. We can use these technologies to conduct environmental surveys, or any task wherein we wish to constrain large-scale state spaces. One of the foremost challenges of this technological revolution is processing data and achieving informational goals in an efficient manner. Current technology simply offers too many data collection opportunities and too much data to review. Adaptive sensing and automation can greatly enhance the utility of remote sensing technology by optimizing its use in situations where valuable information is scarce relative to the overall information content of the environment. Planetary exploration offers a good example of an exploration task wherein valuable information, such as habitable conditions or resource availability, is scarce within areas that we can access and survey. This thesis seeks to utilize advances in remote sensing and manage their proliferating data collection capabilities to efficiently attain informational objectives.

Chapter Outline

Adaptive sensing and other automation strategies are relatively untested as methods to solve these problems, partly due to the novelty of advanced sensing technology. This chapter describes how to apply adaptive algorithms within a planetary exploration architecture. An autonomous bathymetric mapping application provided a foundation for remote sensing tasks. This application informed a larger application task of exploring the surface of Mars.

Section 2.1 begins by defining terms used in the chapter as they relate to the marine exploration and planetary survey applications. For the intermediate and small-scale state spaces, we categorized adaptive sensing into vehicle routing and exploration tasks. These two categories are described in Section 2.1, and later in Sections 2.4.2 and 2.4.3. Section 2.1.1 goes on to describe how adaptive sensing can be framed as an information state Markov Decision Process (info-MDP) and how info-MDPs can be optimized to perform exploration tasks. Adaptive sensing presents different challenges as the spatial scale changes. Section 2.2 reviews info-MDPs in detail, and formulates the active sensing task for surveying Mars as an info-MDP. Section 2.3 describes the assumptions and practice of adaptive planning for a marine bathymetric mapping task and a hypothetical survey of Mars. Section 2.3.1 describes how algorithms can implement low level, real time adjustment to information gathering plans on a short time scale and according to several variables in order to enhance marine bathymetric mapping. Section 2.3.2 describes the function and effectiveness of adaptive planning strategies designed for a hypothetical Mars survey. In this application, we assumed coordinated use of current or proposed satellite and airborne technologies and discusses an active sensing platform that determines and optimal strategy in an evolving environment. Both the vehicle routing and information collection tasks described in Sections 2.1 can be framed as info-MDPs, in which a policy can be learned offline, using a simulated model of the state space. These tasks can be modeled as info-MDPs because the chosen control actions are necessarily conditioned on previously collected observations(see Littman [1996], Bertsekas and Shreve [1978] and Bertsekas [2005b]). Combined with adaptive sensing algorithms, our previously described 3-level hierarchy addresses issues of how to expand the focus of sensing tasks to planetary scales. Section 5.1.1 describes how the 3-level hierarchy integrates multiple sensors at different spatial scales, which can then be addressed as info-MPDs.

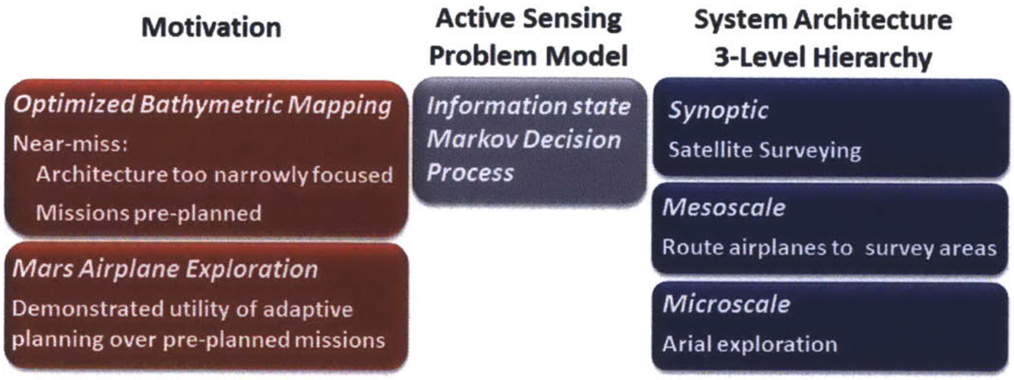

Motivation
Active Sensing Problem Model
System Architecture 3-Level Hierarchy
Optimized Bathymetric Mapping
Near-miss:
Architecture too narrowly focused
Missions pre-planned
Mars Airplane Exploration
Demonstrated utility of adaptive planning over pre-planned missions
Information state Markov Decision Process
Synoptic
Satellite Surveying
Mesoscale
Route airplanes to survey areas
Microscale
Arial exploration

Figure 2-1: Outline of the six applications discussed in this chapter.

2.1 Background

This Section describes the use of active sensing and sample design in information collection problems. In traditional remote sensing tasks, the sampling design is fixed prior to beginning the task, and follows an open loop based on a priori guidance (Aeronautics and Center, for Advanced Computer Science, Bresina *et al.* [2005]). An adaptive sample design changes based on observations collected during the current survey or sensing task.

ACTIVE SENSING

Active sensing is the application of an *adaptive sample strategy* to a *sensing task*. A *sample strategy* is the procedure used to select a sample. A *sensing task* is an operation in which each sample measures the environment (i.e., for survey and mission support).

See Fedorov [1972a], MacKay [1992a], MacKay [1992b], Thompson *et al.* [1996], Thompson and Collins [2002], MacKay [2003], Krause and Guestrin [2007], and Powell and Ryzhov [2012].

Sensing tasks allow us to survey an area and quantify the phenomena most relevant to mission planning and execution. A sensing task, such as a survey, typically observes relevant phenomena that can be characterized by one or more spatial variables. With adaptive sampling, the sampling procedure depends on the values of the observed variables relative to mission objectives. The sampling plan adapts to observed patterns in order to optimally survey the phenomena. Sensing tasks differ in terms of how much the decision and planning processes are affected by incoming information. In the case of current terrestrial surveys, including the marine exploration example described in the following section, information collection is primarily determined by *a priori* survey goals. The theoretical Mars exploration application described in Section 2.3.2 calls for an additional sensing task whose guidance system responds to information in real time as it conducts a survey.

Given the different tasks implicit in larger scale exploration activities, we categorize remote sensing activities as **vehicle routing** problems (See Soloman [1987], Lolla *et al.* [2012], Desrochers *et al.* [1988], Kilby *et al.* [2000], Kolen *et al.* [1987], Larsen [1999], Ralphs *et al.* [2003], and Fisher and Jörnsten [1997]) or adaptive sensing problems (See

Lermusiaux [2007], Mihaylova *et al.* [2003], Mihaylova *et al.* [2002], Ahmad and Yu [2013], Murphy [1999] and Krause and Guestrin [2009]) according to informational objectives, sometimes referred to as **freeform exploration**. Vehicle routing refers to navigational guidance provided to a set of vehicles in order to perform the optimal survey of an area. Vehicle routing treats parameters such as travel time, number of vehicles and area to be surveyed in a different manner than adaptive sensing according to information objectives. This latter activity receives guidance according to real-time information collection and synthesis, and less according to fuel constraints and the current state rewards that determine vehicle routing procedures. These two categories comprise the first order challenges for adaptive sensing at the meso- and micro-scale, and are explained in greater detail in Sections 2.4.2 and 2.4.3.

One of the advantages to using fully autonomous agents is their ability to perform **adaptive planning** and execution. These plans evolve with information gathered from the environment and in so doing, perform better than static planners when changes in utility are modeled at all levels of change in the environment. In order to perform adaptive planning, we have used a hierarchical method that combines a meso scale planner for adaptive sampling, with a low level, kino-dynamic path planner that implements the sampling planning(Léauté and Williams [2005a]) The architecture offers three critical features that benefit large scale information gathering tasks. First, the upper level is performing adaptive sampling, which compensates for new information about the environment. The two lower levels divide the planning into two problems with smaller state spaces, and fewer constraints. Third, online replanning at both levels enables the system to compensate for approximation errors introduced by he two level hierarchy, and because of model. Section 5.1.1 describes the development and deployment of the architecture.

2.1.1 Technical Problem

Vehicle routing (See Lolla *et al.* [2012]) and adaptive sensing are both computationally intensive. The decision uncertainty minimization algorithm described in Chapter 3 addresses this problem for all levels of planning. This chapter focuses on the task of characterizing an environment using adaptive sensing (Lermusiaux [2007]). Two real-world applications of adaptive sensing, satellite-based ocean sensing and unmanned aerial vehicle (UAV)-based mapping for search and rescue mission (detailed in Sections 5.1 and 5.4) demonstrate how adaptive sensing translated into large applications.

In the case of planetary surveys, the sensing platform ultimately attempts to build the most accurate map with respect to a mission-related reward metric, which in this case could be described as map uncertainty. In order to minimize uncertainty, the system state must include an expression for map uncertainty. More generally, the state must contain a mental state or projection of the optimizing model. This projection is distributed over all physically possible world states and is referred to as an information state because it reflects the totality of all information collected thus far or some sufficient statistic thereof. Given that the projection is represented as a sufficient statistic *with respect to optimal behavior*, we can define the information state as an information state Markov decision process (MDP). An info-MDP is an infinite state MDP that makes decisions directly based on its own information state, which allows it to optimize according to aspects of the real world map that positively affect mission objectives.

2.2 Modeling active sensing as an information state Markov decision process

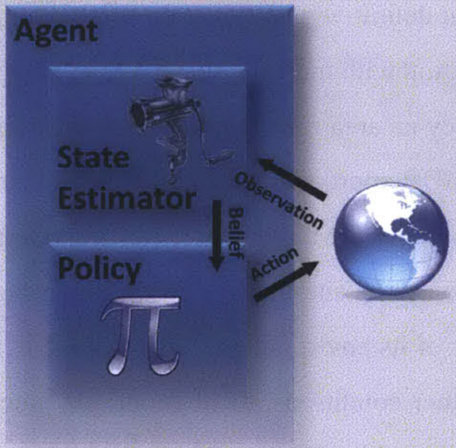
Thus far this chapter has operations in which agents plan automatically and online in order to gather information. Up to this point we have also assumed that exploration targets have arbitrarily pre-determined informational value. An actual survey of Mars would involve much greater uncertainty with respect to the informational value of potential exploration sites. An appropriate plan for larger scale survey would seek to provide the most 'useful' information. To formalize this problem, we specify 'useful' in terms of an objective function over 'information states'. The planner then generates a plan that maximizes this objective function. In this section we argue that this planning problem should be formulated as an information-state Markov decision process.

In actual Mars exploration efforts, information value often corresponds to several associated features that offer integrated evidence of a given phenomena of interest. In the case of current Mars survey missions for example, objects or features of high information value include geomorphologic patterns that suggest the presence of liquid water, mineral classes such as sulfates that form in association with liquid water, and specific minerals, such as montmorollinite clay, that are unique to habitable aqueous conditions. Each of these features can be detected at the synoptic, meso- and micro- levels depending on their scale and concentration, but definitive identification often requires reconnaissance of associated features and confirmation with multiple sensors (e.g., imaging, geochemical analysis, etc.). How then do we survey an area for high value information in an efficient manner?

We selected an MDP approach because of its efficiency at planning a trajectory that maximizes value. To apply a Markov Decision framework to this problem, we begin by assigning each location to be surveyed an information state, which is a probability distribution over the states of its environment. States in the example of the Mars survey might include variable weather conditions, local geography, such as, northern plains and southern highlands, or geomorphic significance, such as younger versus older craters, layered rock formations and dunes. The platform can then plan and conduct measurements that maximize information rewards and reduce uncertainty. The model uses initial, synoptic or mesoscale observations to assign information states and to encode past actions and observations as probabilities of informational rewards (with uncertainties) over states of the environment. These partially observable Markov decision processes (POMDPs), when an agent encodes its history of actions and observations as a probability distribution over states of the environment, can then be converted to an information state Markov decision process (info-MDP), or a fully observable problem. This section first describes the info-MDP approach (Sec. 2.2) and goes on to specify the variables and operations used to construct the model (Sec. 2.2).

INFORMATION STATE MARKOV DECISION PROCESS

An info-MDP is a type of MDP in which the uncertainty regarding the real world state is represented as an MDP or a component of an MDP state. Decisions are made based on what is known and *how well* it is known. An agent consists of a state estimator and a policy. The state estimator incorporates the latest observation into its belief, and the policy is a function of that new belief Littman [1996].



The info-MDP differs from other methods which create a policy for every possible world state then essentially create a compromise policy based on the probability that we are in each of those states (See Sec. 3.6 for a comparison of planning methods). In an info-MDP there is a direct mapping from belief to actions. An info-MDP policy is a function of the info-state rather than the world state.

See Littman [1996], Bertsekas and Shreve [1978], Bertsekas [2005b], and Bertsekas and Castanon [1999].

Experimental Approach

Information collection problems incorporate information from past iterations, referred to as memory in this context. Consideration of past observations is preferable to treating the current observation as a true state, due to uncertainties in observations and the state's evolving nature. In a true MDP, a subsequent state is conditionally independent of past states, given the current states. For this to hold true in a problem involving observations, we could consider all past states as part of the 'memory' of the current state. Policies that do not assimilate previously collected information perform poorly on information collection tasks, relative to those that have a memory component (Littman [1996]; Section 6.2.2). Computing some statistic or belief from past observations, which approximate the required conditional independence offers a better approach.

The value function of an info-MDP captures the value of the optimal experiment design, given the known or represented facts. Optimal experimental design is the process of selecting a set of samples or experiments so as to maximize the knowledge gained from them. A value function is the discounted future reward to be gained by following a given sequence of actions, which may include contingencies. In an info-MDP, a reward metric captures the information contribution produced by each action, to the situational model (our projected belief). The cumulative future reward for subsequent experiments such as samples, data or measurements, is thus a value function. In the Mars survey example, our sensing platform is trying to map an area with respect to a mission-related reward metric, which we model as uncertainty minimization. The agent's belief regarding the current map uncertainty must be included in the system state in order to reward actions that reduce uncertainty. This belief represents a distribution over all physically possible world states. The optimizing agents belief is an information state because it incorporates the sum total of all the information it has collected thus far. By representing the agents belief with a sufficient statistic with respect to optimal behavior, we can define an info-MDP (Littman [1996]), which includes the agents awareness of its own information state, given the uncertain world state. This framework rewards actions that leads to a better information state. See MacKay [1992a] for more information.

Information-state Markov Decision Processes: Variables and operations

Uncertainty minimization is a key objective of the information gathering problem described here. An MDP assumes that the state of the world is fully observable, but information-gathering must often estimate the world's state. How do we construct a model that captures world state uncertainty? The most general formulation of planning with uncertainty is the partially observable Markov decision process (POMDP). A POMDP is defined as a tuple

$(\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'}, \mathcal{Z}^a_{s'o}, \gamma)$, where $\mathcal{S}, \mathcal{A}$, and, $\mathcal{O}$ are the set of states, control actions, and observations. $\mathcal{P}^a_{ss'}$ is the probability of getting to state s' when taking action a in state s and $\mathcal{R}^a_{ss'}$ is the corresponding scalar reward. Since states are not observable, $\mathcal{Z}^a_{s'o}$ provides the probability of receiving observation o when taking action a and reaching state s'. The γ term is used as the discount factor to bound the total accumulated reward. Since the agent is unable to observe the state, it maintains a probability distribution over states called the *belief*. Hence a policy (π) maps such beliefs to actions. If the reward function depends on the belief, which is often the case for the exploration tasks the state *itself* must reflect the belief, since $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$. On the other hand, if the observation model is known *a priori*, the belief can be updated on each time step, given the observation. These facts allow us to operate in a fully observable domain, where the new state is the belief itself and the new transition model incorporates the observation model.

This formulation specifies the problem as an info-MDP, where the MDP is defined as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'}, \gamma)$. Any situation with imperfect state information can be reduced to a problem where the state is fully observable (Bertsekas [2005b] p. 218). The new problem makes a decision based on the information state, which is fully observable. Thus, the problem is equivalent to a problem where the state is fully observable. Bertsekas and Shreve [1978] first introduced the info-MDP and Cassandra *et al.* [1994] and Littman [1996] offer detailed descriptions of it.

2.2.1 Active sensing as an information-state MDP

The overall objective of active sensing objective is to construct an accurate map upon which exploration decisions are based. The objective function must therefore include some measure of map accuracy given an unknown true world state, and the overall objective and immediate reward must be information related. In framing this problem as an info-MDP, our task entails measuring real-world variables in order to maximize the value of informational in a given area.

Suppose we wish to measure and map the presence of some phenomena of interest over a geographic area. We can think of this problem as creating an accurate occupancy map and model the area as a set of n cells arranged as a rectangular grid.

- **State Space:** $\mathcal{S} = [0,1]^n$ where n is the total number of grids and s_i indicates the probability of grid cell i occupancy.
- **Action Space:** $\mathcal{A}(s) \subset \mathcal{A} = \{1, 2, \ldots, n\}$, where an action a is to sense a particular grid location a and the surrounding cells. $\mathcal{A}(s) \subset \mathcal{A}$ is the set of currently available sensing actions, those immediately reachable from the current location.
- **System Dynamics:** The system dynamics represents the probability of transitioning from information state s to s' when taking sensing action a, where s' is our new information state, given sensor return w (following Equation 2.2). Therefore, the transition probability is equal to the probability of getting sensor return w given action a in state s.

$$\mathcal{P}^a_{ss'} = P(w|s,a) \tag{2.1}$$

such that

$$s' = P(x|w) = \alpha P(w|x,a)s. \tag{2.2}$$

Note that in the continuous case, s is a probability density function over continuous variable x and in the general case the Bayesian update (Equation 2.2) may require integrating over x. When we can compute the expected costs of state estimation error, we may select sensing actions specifically to reduce those costs. For example, suppose an autonomous Mars airplane team is tasked with finding interesting sites to investigate. Locations are optimized according to signal strength. Furthermore, identifying safe traversable paths for a ground rover to get to these locations is equally important. On the other hand, when we cannot compute the expected cost of state estimation errors, we use an information theoretic approach of maximizing reduction in uncertainty. One of the most common measures of reduction in uncertainty is reduction in entropy:

$$\mathcal{R}^{a}_{ss'} = H\left(s\right) - H\left(s'\right), \tag{2.3}$$

where

$$H(s) = -\sum_{i} \big(s_i \log_2 s_i + (1 - s_i) \log_2 (1 - s_i)\big). \tag{2.4}$$

and i indexes each cell.

The literature concerning the design of these experiments refers to this as a "D-Optimal" objective function (Fedorov [1972a]). Another possible choice of an information theoretic objective is the "E-Optimal" reduction in expected root mean squared error (RMSE). For a single variable binomial distribution with probability p such as each grid cell in the occupancy map, the RMSE is:

$$\begin{aligned} RMSE(p) &= \sqrt{(1-p)^2 p + (0-p)^2 (1-p)} \\ &= \sqrt{p(1-p)} \\ \mathcal{R}^{a}_{ss'} &= \sum_{i=0}^{n} \big(RMSE\left(s_i\right) - RMSE\left(s'_i\right)\big) \end{aligned}$$

Figure 2-2 shows that RMSE is similar to entropy. RMSE may be a more appropriate reward because it is directly related to expected map error, computed as the true occupancy (1 or 0) minus our estimate p squared, multiplied by the probability with respect to occupancy. In addition to being computationally more tractable than entropy when computing the expected outcome of a sensing action Bush *et al.* [2008a].

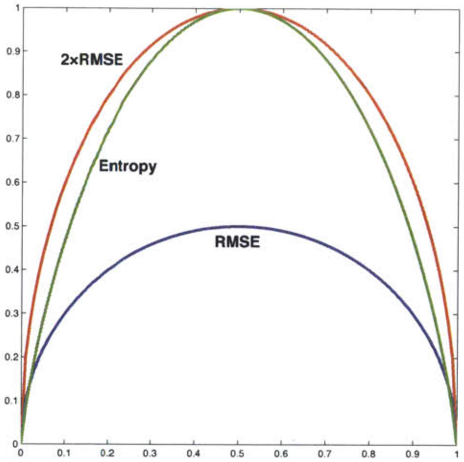


Figure 2-2: RMSE versus Entropy.

2.3 Case Studies

2.3.1 Case Study 1: Planning and Control for Autonomous Bathymetric Mapping

Before framing information gathering tasks as an info-MDP, we describe lessons learned from a project initially developed in support of information gathering missions. We equipped an autonomous underwater vehicle (AUV) operated by the Monterey Bay Aquarium Research Institute (MBARI) with a novel autonomous sensing control algorithm. The AUV mapped difficult to access region of Monterey Bay some three miles off shore (Monterey Bay Aquarium Research Institute [2006]). This project is described in detail in Appendix A. Briefly, the AUV produces a bathymetric map by descending to bathyal depths and by tracking its position above the ocean floor via multiple navigation sensors. As long as the sensors allow the AUV to maintain ground- lock, which occurs at about 50 meters above the ocean floor, the AUV can accurately track its position. When it loses ground-lock, the AUV must default to its inertial navigation system, which compromises accuracy. We implemented a spliced linear programming algorithm that plans vehicle maneuvers based on more accurate information concerning the vehicle's depth and surroundings. The modified strategy enhanced the AUV's ability to maintain ground-lock in variable terrain and thus improved its mapping capabilities.

This initial foray into planning for a sensing task fell short in two respects: a) it was a non-adaptive survey strategy, and b) the strategy was narrowly focused on only one environmental variable (depth).

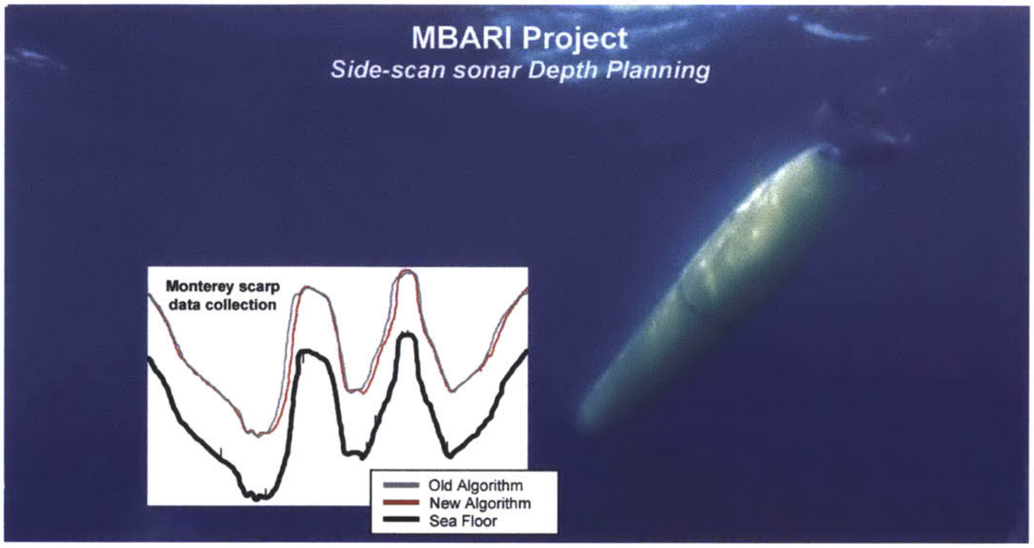


Figure 2-3: Bathymetric mapping optimization project, motivated by the desire to explore the oceans of Europa: The inset graph shows that the MBARI Dorado class AUV tracked the target depth better when controlled by our new algorithm. The black line shows the ocean floor. The red line shows the actual path followed when controlled by our new algorithm. The grey line shows the actual altitude under the old algorithm. Our AUV altitude planner uses a linearized dynamics model and controller, linear programming and a novel problem decomposition method in order to optimize mission data collection quality.

If the AUV sensed an unmapped structural fault or unknown circulation current, for example, we would have liked the algorithm to have responded by inspecting the potentially valuable patterns associated with these phenomena. The existing AUV survey was narrowly focused in terms of the limited information collected and the sensor use. These limitations raise questions of how the planning can be scaled to much larger information gathering tasks that, for example, included multiple sensors. Both of these limitations also relate to the mission's *a priori* planning. Broadly focused missions with adaptive sensing can enjoy and benefit from dynamic planning.

2.3.2 Case Study 2: Adaptive Mission Planning for Mars Exploration

As a second example, we consider NASAs concept of UAV on the surface of Mars. In current Mars surface missions, plans sent from Earth are essentially static given communication lag. Adaptive planning can help circumvent this challenge and provide a coordinated response to changing conditions. This section describes how active sensing algorithms and approaches can perform in more adaptive modes for a hypothetical autonomous aerial survey of Mars assuming use of current and concept aircraft. The model and architecture presented here differ from previous work in that they includes continuous high-level planning, allowing agents to incorporate newly acquired information on an ongoing basis. The following sections describe some of the assumptions necessary for adaptive planning and its benefits over static planning, and how adaptive planning performs in simulated environments.

2.3.3 Adaptive Mission Planning for Aerial Exploration of Mars: Assumptions

Adaptive mission planning on planetary scales requires some assumptions regarding planning strategies. First we assume access to current satellite technology, such as the Mars Reconnaissance Orbiter (MRO) (See NASA and JPL [2013] and Smrekar [2007]), as well as a concept aircraft, the Aerial Regional-scale Environmental Survey of Mars (ARES) (See NASA [2011a], Levine *et al.* [2003] and Braun *et al.* [2006]). MRO has mapped the Martian surface with its high-resolution cameras since 2006. The instrument has played an important role in choosing landing sites, providing navigation data and serving as a telecommunication relay for landed missions. In addition to its imaging capabilities, MRO also monitors the Martian climate and catalogs geomorphologic features of the Martian surface. ARES is a concept mission developed by Dr. Joel Levine, intended to study crustal magnetism, atmospheric chemistry and search for hydrologic clues on the Martian surface. Adaptive sensing with these instruments then requires us to specify decision making strategies according to our appraisal of the overall environment in which the survey takes place. How do we maximize information rewards in environments that are not only changing, but are likely to have variable rates of change?

To answer this question, we compared three mission planning strategies: static, greedy and non-myopic adaptive, over a range of simulated stochastic environments (see Appendix B). The results show that an adaptive non-myopic method performed better than both a static (open-loop) and greedy approach. The adaptive approach behaved in a similar manner to that of a greedy approach in a rapidly changing environment and in a similar manner to that of an open-loop approach in a slowly evolving environment. The adaptive non-myopic approach behaved better than both of the other approaches in an environment that is changing at a moderate rate. Our results showed that the adaptive planner outperformed the static finite-horizon and greedy planners in situations involving moderate rates of change, and performed at least as well as the other two planners under more variable conditions.

2.3.4 Adaptive Planning for Aerial Exploration of Mars: Demonstration of Principle

We constructed a set of simulations that explored the feasibility and benefits of adaptive planning for Mars exploration. These simulations combined high- and low-level planning route UAVs among Mars localities of varying scientific value, subject to fuel and time constraints. The high-level planner performed site selection according to scientific objectives in tandem with a low-level kino-dynamic path planner that transmitted waypoints for UAV routing. The full experiment is described in Appendix B, and summarized below. These simulations showed that the adaptive planning approach identified solutions that optimized the value of information within the constraints of the simulation.

Briefly, this experiment assumes a set of location of unknown information value. The adaptive mission planner calculates an exploration strategy that maximizes utility within fuel constraints. The plan is then transferred to the kino-dynamic path planner, which creates waypoints to guide UAVs to the locations in the most efficient manner. Way- points are designed based on a one-second time scale, so as to discretize each path unit on a relatively fine scale. We interfaced the adaptive mission planner with the Mars Plane simulator. The adaptive mission planner constructs a plan and calculates the fuel usage. Excessive fuel

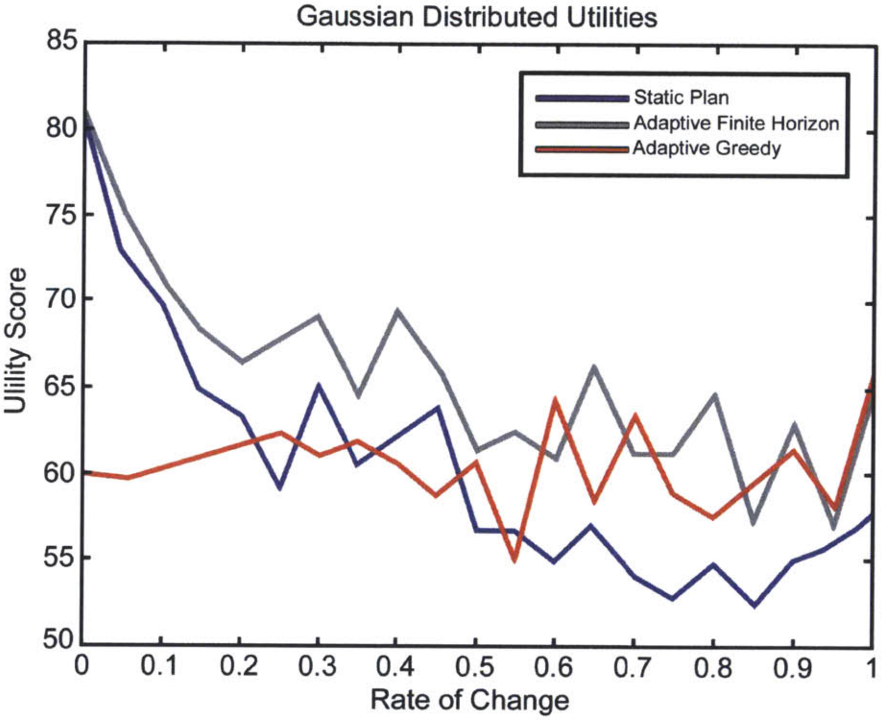


Figure 2-4: The graph shows the results results of plan execution for three strategies. The rate of change is along the x-axis and the average total utility gained is on the y-axis. The blue line is the static finite-horizon plan, the green line is the adaptable finite-horizon plan and the red line is the greedy plan (adaptable one-step receding-horizon plan). The static planner outperforms the greedy planner under stable conditions and the greedy planner outperforms the static planner under unstable conditions. The adaptive planning strategy performs well in all situations. It performs at the same level as the static planner under stable conditions, outperforms either planner under moderately changing conditions, and matches the performance of the greedy planner under very unstable conditions. (See Appendix B for more details)

usage may lead the planner to construct and alternative plan. Figure 2-5 shows the operator interface for the simulated system. The simulated environment includes geographic features, such as mountain ranges, that the UAVs should avoid, as well as craters, outcrops and alluvial fans that have potential information value.

The sites are color coded in Fig. 2-5 along with the UAV starting location (a white square), a set of utilities for each science site (yellow numbers) and the travel distance implicit in the map area. As the planner receives the actual travel time to the current site and the updated utilities, it creates a new plan (Fig. 2-4 e and f), which optimizes utility and fuel use according to the new information and constraints. In this situation, the adaptive planner can accrue greater utility while operating within the fuel constraints relative to that accrued by a static plan across a range of environments that were changing at different rates (see Appendix B for further details.)

2.4 3-level Hierarchy

The next set of challenges in framing a planetary survey in terms of an info-MDP concern how to cover a large geographic area and integrate multiple sensor types in the active sensing task. We designate a 3-level spatial hierarchy that allows us to address the unique spatial and sensor integration challenges that arise in sample spaces of different scales. The 3-level hierarchy shown in Figure 5-3 includes a synoptic level, which represents a wide area (10^3 km or more), a mesoscale level, representing an intermediate-sized area and a microscale level, covering a small, local area (less than one km). This hierarchy allows us to implement an integrated system of info-MDP algorithms to adaptively survey the sample space according to basic information objectives.

The instruments, vehicles, objective and operations at each level are different, but may overlap. A satellite provides coverage and support at the synoptic level, while UAVs operate at the meso- and microscale levels (Fig. 5-3). The satellite coarsely surveys a large swath of the terrain and then relays information to a group of UAVs. The UAVs deploy to the areas that appear to have high value of information. Each level requires a different optimization strategy due to the different sensor functionalities and the abstraction of the environmental

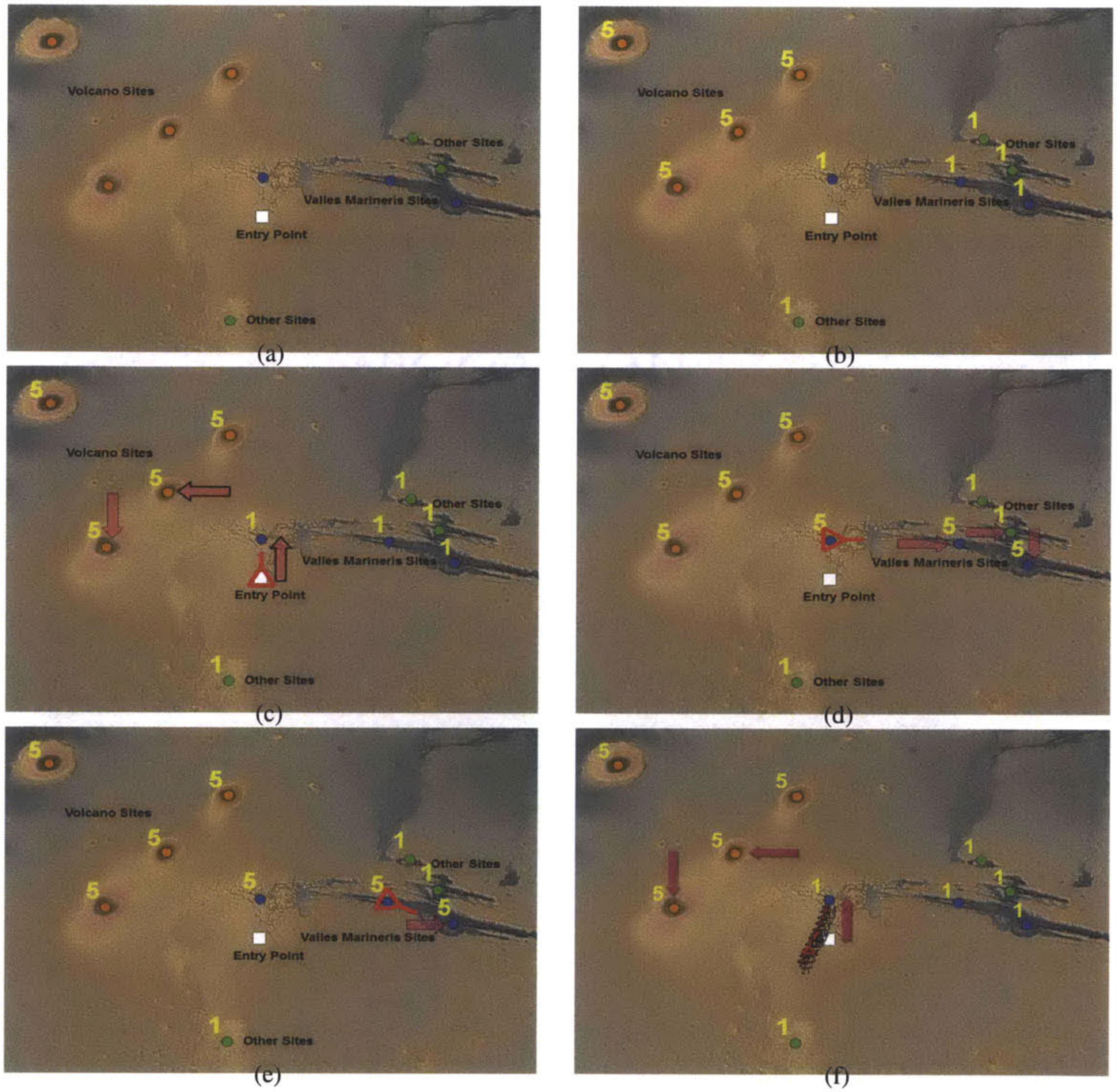


Figure 2-5: Mars Airplane Adaptive Mission Planner: The ARES aircraft could explore a diverse terrain in the equatorial Valles Marineris region shown in (a) at a vantage point 1.5 km above the surface. Our high-level adaptive mission planner performs site selection based on an evolving science value estimate. A lower level kino-dynamic path planner executes the resulting plan. The map shows the aircraft entry point and 3 categories of sites to explore. (b) The numbers indicate the science value estimate (utility) of the 3 site categories. (c) The Adaptive Mission Planner uses these utilities and fuel constraints to construct a plan. The kino-dynamic path planner executes the plan. When the aircraft visits and observes a site, it updates the utility estimate for that particular site category. Panel (d) reflects the effect of observing a blue site. The adaptive Mission Planner uses the new utility values to formulate a new plan. (e) The Adaptive Mission Planner also updates its time horizon based on the actual distance traveled. A significant deviation from the estimated distance results in a new plan. In this example, the Mars airplane took longer than expected to reach a planned site and a new, shorter plan was constructed. (f) Our system includes a low level path planner that can simultaneously avoid obstacles (weather, mountains etc.) and navigate according to flight dynamics (Léauté and Williams [2005a]).

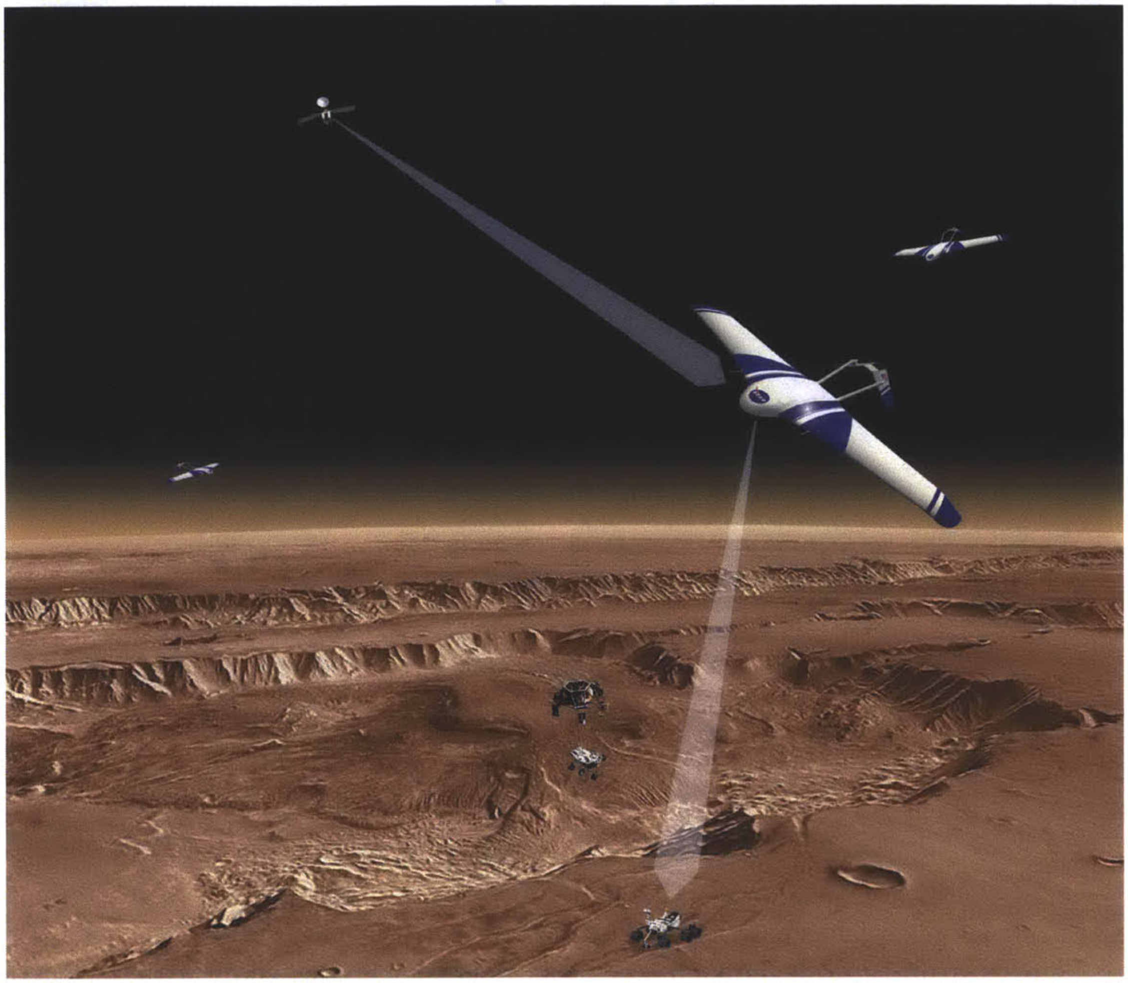


Figure 2-6: The Autonomous Mars Exploration Network: We extend the notion of autonomously exploring Mars to a multi-level framework incorporating as satellite surveyor, which provides synoptic information to a team of UAVs. These UAVs explore the surface of Mars for features of scientific interest. Once located, the UAV can map out a set of traverses that can be executed by a ground rover to the site.

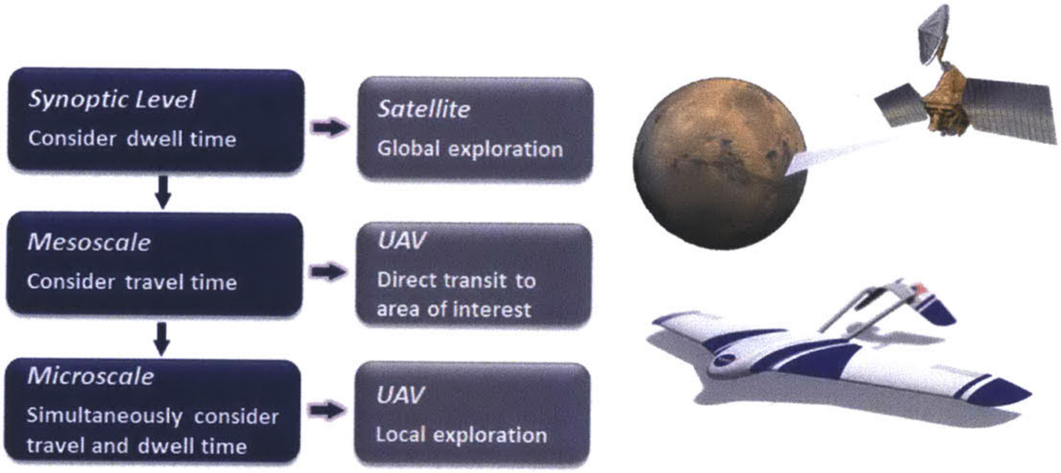


Figure 2-7: The 3-level hierarchy shown above includes a synoptic level (covering a wide area), a mesoscale level (covering a medium sized area) and a microscale (covering a small local area). The synoptic sensing (satellite) algorithm concentrates on dwell time and the value of the collected information. The mesoscale system is optimized with respect to vehicle travel time. Travel and dwell time are simultaneously optimized at the microscale during free-form AUV exploration.

model. A satellite can quickly refocus its survey within its sensor footprint. The sensing algorithm therefore concentrates on dwell time and the value of the collected information. The dwell time refers to how long the satellite looks at a particular location. The satellite is therefore optimized to focus on areas of high interest and high uncertainty.

At the mesoscale level, the UAVs transit directly to the locations of interest. The information gathered during transit is considered negligible, and is ignored in the formulation, thus simplifying the model of reward. The mesoscale system is optimized with respect to travel time and fuel constraints. The satellite provides a set of target areas that are divided up among UAVs according to an optimized vehicle routing plan. The microscale planner handles the local sensing tasks. At the microscale level, the UAVs perform free-form exploration, collecting data as they move. Travel and dwell time are thus simultaneously optimized at the microscale. Next we examine and formulate the info-MPD for each levels of the hierarchy.

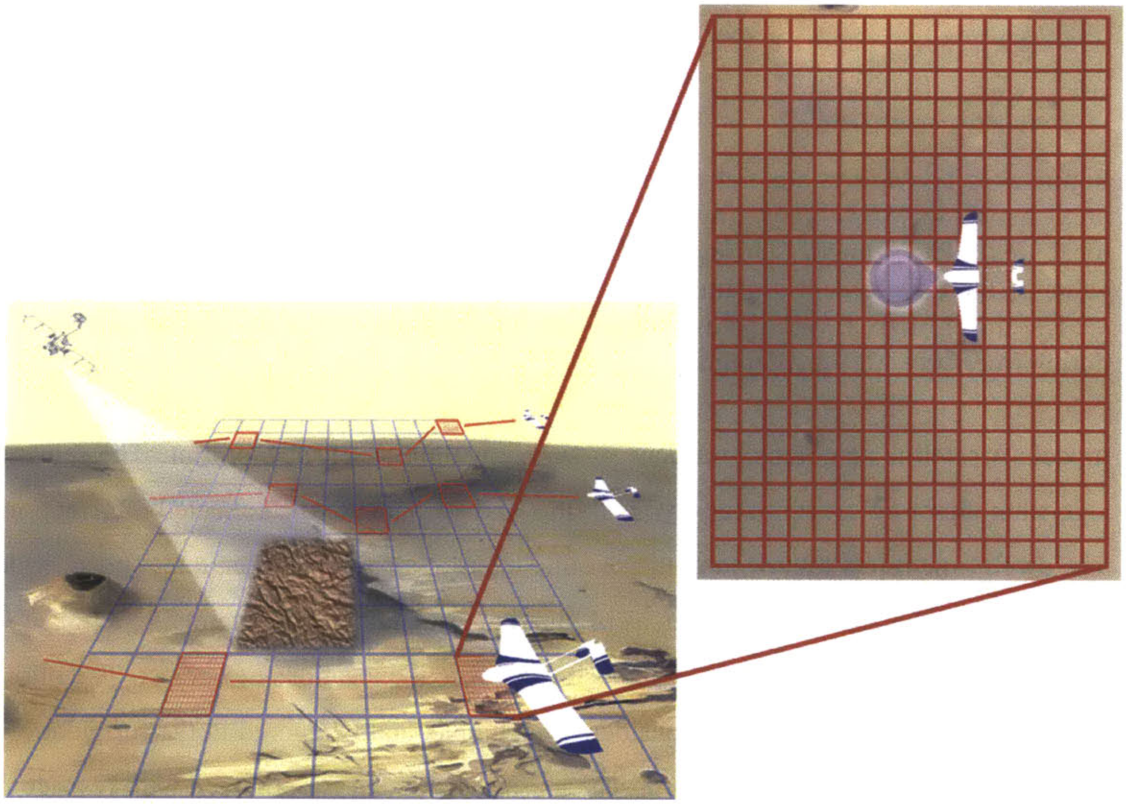


Figure 2-8: Satellite-UAV collaboration can be broken into different levels. At the mission level, we route the UAVs to the mapping task locations. In the local level, each location is its own information gathering task with the goal of estimating key environment variables in the area. Both problems can be solved with an uncertainty guided planning algorithm.

2.4.1 Synoptic Level Optimization (coarse global satellite surveying)

At the synoptic level, satellites identify areas of potential scientific interest that can then be addressed at the meso- and micro-scale levels, using different sensors. The satellite operates with a large footprint and a relatively low slew cost (the cost to change the platform attitude and sensor aim-point). The survey is therefore optimized purely with respect to the value of the information it collects.

Figure 5-7 depicts an occupancy grid representation of the satellite active sensing problem. The occupancy grid divides up the area into discrete locations, which we represent as a belief map recording the probability of "occupancy" for each grid cell. Occupancy means that the phenomena of interest is present. When we select a grid cell to point the sensor at, the sensor observes that grid cell and the surrounding grid cells. We model the sensor's capability as a discrete Bayesian detection model, which has a probability of detecting the activity of interest, and a probability of false alarm. The sensor will return a one if it detects activity of interest and a zero if it does not. The process works as follows. We take a sensing action which returns a one or a zero for each of the observed grid cells. We then update or belief about the world. We then use that updated information to select the next most informative sensing action. Our overall objective is to maximize map quality. That is to maximize the certainty of our belief regarding the activity in the area. This framework maps exactly to the information state MDP framework already described.

We specifically developed a fast adaptive sensing algorithm and then augmented the results via online rollout based reevaluation. The results show clear improvement of reevaluation over approximate dynamic programming by itself.

Synoptic problem model

At the synoptic level, our task is to map an area and sequentially decide where to sense next. We model the environment as an occupancy grid, as shown in Figure 2-8. The stochastic sensor measurements cause uncertainty regarding the map state. Therefore, we infer the probability of activity in a given cell using a Bayes filter:

$$p_i = P(i|w_i) = \alpha P(w_i|i)P(i) \tag{2.5}$$

where p_i represents the posterior probability that a grid cell is occupied, w_i is the sensor measurement for cell i, $P(w_i|i)$ is the sensor model and $P(i)$ is the prior probability of activity at cell i. Recall that the probabilities of the entire grid represent a "belief map." The exploration process of building an accurate activity map works as follows. We take a sensing action, which returns a measurement for each of the 3×3 observed grid cells. For each sensed cell, we then update the belief map s_i using Equation 5.1. We then use the updated belief map to select the next most informative sensing action.

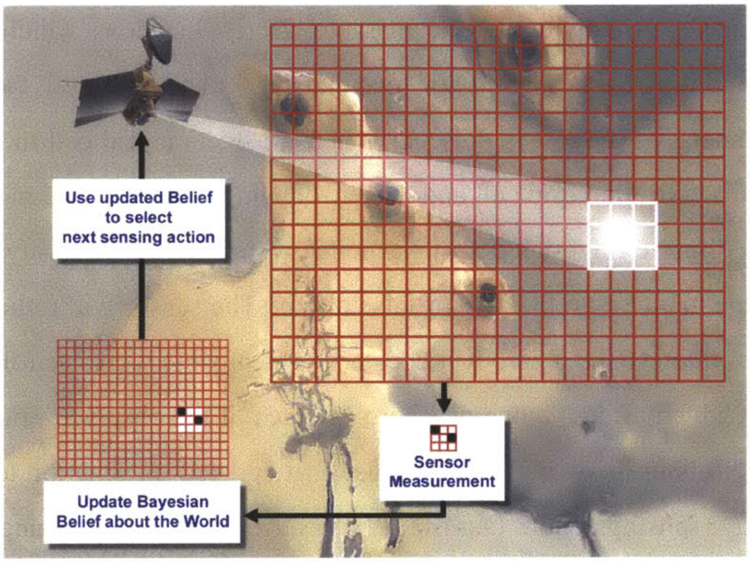


Figure 2-9: Occupancy grid representation of our satellite active sensing problem depicting Bayesian belief updating.

2.4.2 The Mesoscale Problem (routing UAVs to survey locations)

At the mesoscale level, multiple UAVs coordinate with external mission requirements to accomplish an information-gathering task. To accomplish these goals, a high-level mission planner assigns a set of mapping tasks to a UAV team (based on data, models and other constraints). Each task includes a beginning and ending time constraints, assuming limited fuel and energy resources, and each location becomes its own mapping task. At the mission level, the autonomy objective is to minimize total travel time between locations, while the local mapping objective is to maximize value of information.

We may view this also as in info-MDP. In this action hierarchy we abstract away the mapping problem to a single temporally extended action (see Figure 2-10) with a stochastic reward outcome. That reward is based on what we find at the exploration site. A discovery at one site can have an impact on how we value other potential exploration sites. The mesoscale level rewards evolve based on the information state and thus represent an info-MDP.

For tractability, the following problem setup focuses on solving for the instant information state. The problem is treated strictly as a vehicle routing problem with a set of informational rewards. The optimization of that vehicle routing problem is framed as a plan-space (Ghallab *et al.* [2004]) MDP, where the MDP state is actually a plan, actions are changes to the plan and the reward reflects improvements to the plan. A DUM related algorithm is then applied to that MDP to efficiently guide that process toward a good plan.

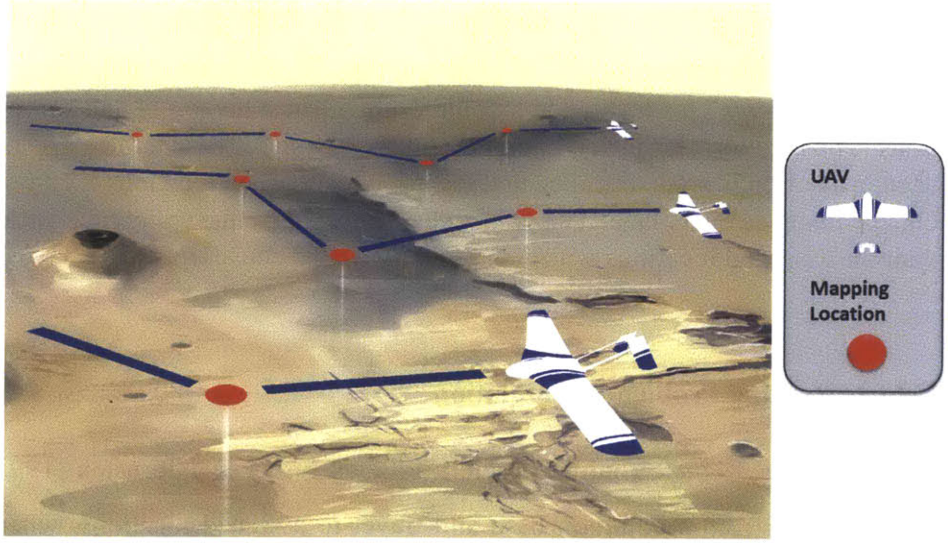


Figure 2-10: Mission level data collection planning: A set of vehicles must perform sensing at various locations. An autonomous agent chooses a set of tasks and plans a route for each UAV. The plan maximizes science gain while adhering to high-level scientific guidance. The plan may need to adapt to mission changes.

Mesoscale multi-vehicle aerial survey coordination

The aerial survey at the mesoscale level of a Mars survey requires the survey application to determine the most efficient way to route vehicles to the survey locations. We tested an aspect of our algorithm on this problem, by framing it as a Markov decision process, learning an off-line approximate policy, and refining the policy online using nested rollout (See Sec. 3.6.2). Our approach starts with an off-the-shelf heuristic, which we enhance using approximate dynamic programming, and further augment online through decision uncertainty minimization planning.

Variable	**Definition**
m	number of vehicles
n	number of locations including the UAV base
c_{ij}	travel cost from location i to j
t_{ij}	travel time from location i to j
s_i	required mapping time at location i
e_i	earliest time allowed for arriving at location i
u_i	latest time allowed for arriving at location i
N	$\{1\ldots\ n\}$ the set of locations to visit
N_0	$N \cup \{0\}$ location set including the UAV base
M	$\{1\ldots m\}$

Figure 2-11: We formalize the mission-level mesoscale survey as a vehicle routing problem. A set of vehicles located at a common base must deploy to and survey various locations. The location arrival and departure times may be restricted, the mapping time allowed at each location is predetermined and the distance between locations is computed as the Euclidean distance on a plane. The objective is to minimize the travel time such that the UAVs visit the assigned locations within the specified time constraints.

Vehicle Routing Problem Formulation

Our mesoscale vehicle routing problem follows Fisher and Jörnsten [1997]. The UAVs are deployed from a common base and travel to assigned locations, where the distance between the locations is computed as Euclidean distance on a plane. The times of their departure from base and arrival at their destinations may be restricted, and the amount of time they can spend at each site may be limited. The vehicle routing problem formulation is optimized to minimize their travel time, so that they can visit all of the assigned sites within the specified time window.

We have developed an approach to this problem based on approximation dynamic programming (ADP). This method frames the planning problem as a sequential decision making problem, and constructs the route in stages. It formalizes the problem in terms of states, actions, rewards, system dynamics, and as an optimization function. Finally, it reformulates the optimization as a dynamic program and defines an approximation architecture to represent the cost-to-go.

2.4.3 Microscale Level Problem : local free-form exploration

At the microscale level of the hierarchy, autonomous agents must make decisions based on the information they collect while exploring the local area. We use adaptive sampling to maximize utility within the constraints with each location representing a separate mapping problem.The planner guides the UAV to map the terrain, and determines its sequence of operations. We model the environment as an occupancy grid (Fig. 2-11), and divides up the area into discrete locations. When the UAV is at a particular grid cell i, it observes that grid cell and each of the surrounding cells. Each of the measurements conducted by a single sensing action is modeled with some probability of detecting informational rewards with some probability of false positives. We then represent the map internally as a belief map that records the probability that each grid cell contains an information reward, based on the sensor measurements. This info-MDP representation is almost identical to the synoptic scale representation, except that the vehicle (a UAV) is constrained to following a path. Therefore, actions reflect local location changes.

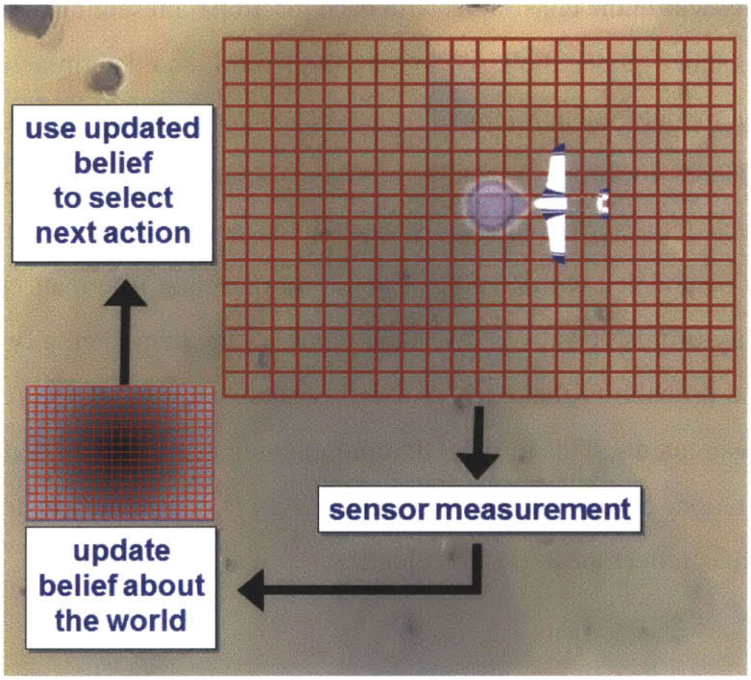

use updated belief to select next action
sensor measurement
update belief about the world

Figure 2-12: Adaptive Sampling: Autonomous agents need to make decisions as the system is running, in order to take advantage of newly collected information. We model the environment as a 3×3 occupancy grid and represent the map internally, as a belief map, which records the probability of informational rewards at each grid cell. When we take a sensing action, we update our belief about the world, then use that updated information to select the next most informative sensing action.

3 An Introduction to Decision Uncertainty Minimization Based Planning

Remote sensing and exploration tasks are guided by mission objectives which can be parameterized as an information policy. Autonomous agents make sequential decisions regarding how to operate in and interact with the environment according to this policy. Real world environments present a large amount of information and a number of different actionable possibilities. How can we program autonomous agents to make the best decisions in remote sensing and exploration tasks? This chapter explores how agents can use decision uncertainty to improve decision-making. Appendices A and B provide detailed background information on decision uncertainty minimization (DUM) planning algorithm. The critical contribution of decision uncertainty minimization planning is the use of uncertainty in guiding the online simulation-based planning process. DUM algorithms select the online planning steps that maximize the probability of discovering the optimal action.

3.1 Sources of Uncertainty in Sequential Decisions

The primary sources of uncertainty affecting sequential decision making include a) the stochastic dynamics of the environment and b) sensor measurement noise. We average these stochastic elements out of our decision by computing the expected value (E[V]) under a given policy. We build a value function covering the entire state-space, which provides a basis for decisions. Uncertainty persists regarding the expected value however, due to the impracticality of learning, storing or calculating the E[V]. We work instead with an approximation of E[V], represented as a distribution.

The uncertainty associated with the approximate E[V] can be translated to decision uncertainty. Simulation allows us to derive the expected value of a set of stochastic outcomes upon which decisions can be based. Each course of action (COA) can be assigned an expected value. A COA consists of an immediate action plus subsequent actions dictated by a given policy, which may include predetermined actions and conditional decisions. In our case, the policy is governed by a value distribution. Each COA therefore has a unique E[V].

DEFINITION: EXPECTED VALUE E[V]

A policy determines and assigns an action to each problem state. The value of a state (s) under a policy is the expected value of the discounted sum of rewards for implementing the policy, starting from state s. The optimal policy maximizes the expected value for every state. The optimal value function solves the Bellman optimality equations (e.g., Bertsekas and Tsitsiklis [1996a]). This function defines the optimal action for each state. In this context, the value function is approximate but not optimal.

SELF AWARENESS

Decision Uncertainty Minimization (DUM) planning is aware of its own incomplete information and makes decisions accordingly. If DUM planning is unsure about the precise value of a potentially valuable alternative for example, it will simulate it in order to determine its value more accurately. This process of projecting states into the future improves decision making.

Decision uncertainty specifically refers to the case in which an agent faces a range of possible actions with unknown expected values. The algorithms presented here provide uncertainty estimates and other guidance concerning these expected values. Uncertainty regarding expected values corresponds to an agent's awareness of how well it knows the answer. DUM uses model-based search to derive this information (in appropriate situations), which can then help guide the agent's actions. Because the agent optimally manages its own incomplete information using DUM, it classifies as a self-aware entity.

3.2 DUM Planning Algorithm Outline

DUM quantifies the uncertainty of a policy, which can then be used to guide online planning. The following description of DUM assumes that the reader has a good understanding of Markov Decision Processes (MDP) and online search. The appendices cover these topics in more detail. In the first part of DUM, we use ADP to determine a policy offline that offers the highest probability of achieving mission objectives. This step creates an approximate solution. The quality of this approximation is typically ignored. DUM however quantifies the uncertainty of this policy and codifies it as a value distribution, as outlined in Section 3.5. The second part of DUM involves online reoptimization of the policy according to the immediate situation using Monte Carlo sampling based planning. Section 3.3 reviews Monte Carlo sampling based planning as well as other less complex approximate methods including certainty equivalence, open-loop feedback control and rollout. These planning methods do not include uncertainty estimates. The DUM representation provides us with information about the accuracy of that value function, which can then be used to guide our online COA selection. This mechanism, referred to as an information guide, focuses and improves the reoptimization process by quantifying the accuracy of our estimates and reducing uncertainty.

Decision uncertainty algorithms consist of steps to construct the value function distribution and control the search process. As a secondary objective, the algorithm seeks to find a basis function that approximates the value function with low estimation uncertainty. Finally, the algorithm's tertiary objective is to determine the online search depth and the branching factor for each depth. Later in the thesis we construct and compare algorithms to demonstrate their operation.

3.3 Offline Approximate MDP policies

This section defines the Markov decision processes, which we use as our framework. The section also defines the relationship between the value function and the state-action value function. The approximate dynamic programming approach learns a policy offline. The suboptimality of this process can be modeled as representational uncertainty. This uncertainty can then be used in our online reoptimization algorithm via decision uncertainty minimization planning. Appendix A provides a review of offline policy learning through the MDP. Section A.1 defines an MDP; Section A.2 reviews the value iteration algorithm as it applies to small discrete problems. Section A.3 presents the more general framework of policy iteration, and Section A.4 reviews approximate value iteration known as approximation dynamic programming (ADP).

3.3.1 Markov decision process defined

A MDP is defined by its state set S, action set A, state transition probability distribution P, and reward function R. When executing action a, in state s, the probability of transitioning to state s' is denoted as $\mathcal{P}^a_{ss'}$ and the expected reward associated with that transition is denoted $\mathcal{R}^a_{ss'}$.

A policy assigns an action to each state of the MDP. The value ($V_\pi(s)$) of a state under a policy π is the expected value of the discounted sum of rewards obtained when π is followed, starting in state s. The objective is to find an optimal policy π^*, that maximizes the value of every state ($V_{\pi^*}(s) = V^*(s)\ \forall s$).

The optimal value function (V^*) is the solution to the Bellman optimality equations (e.g., Bertsekas and Tsitsiklis [1996a]): $\forall s \in S$

$$V(s) = \max_{a \in A} \sum_{s' \in S} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')] \tag{3.1}$$

where the discount factor $0 \leq \gamma < 1$ discounts future payoffs relative to immediate payoffs. The Bellman equation (Eq. 3.1) is also referred to as the dynamic programming equation, and describes a one-step future projection with respect to the value function. The optimal policy π^* can be determined from V^* as follows: $\pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^*(s')]$. Therefore, the optimal value function is virtually the MDP solution.

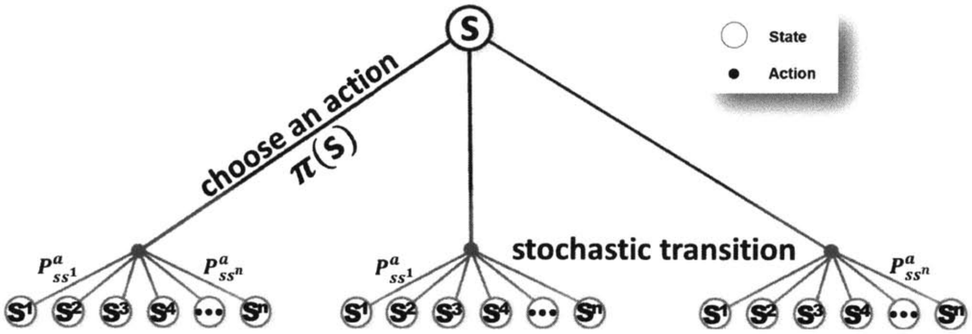


Figure 3-1: MDP mechanics: In the MDP framework, we find ourselves in state s where policy π chooses an action to execute. Although the action is chosen, we do not know for sure which state we will transition into. When executing action a in state s, the probability of transitioning to state s' is denoted as $\mathcal{P}^a_{ss'}$. The MDP optimization finds the best policy, i.e., the policy that maximizes the value of every state.

3.3.2 Relationship between value function $V(s)$ and state-action value function $Q(s,a)$

The state-action value function $Q(s,a)$ provides the value in state s of any action a, whereas the value function $V(s)$ provides the value in state s of the best action selection, as determined by policy $\pi(s)$. This policy is as follows:

$$\pi(s) = \arg\max_{a \in A} Q(s,a) = \arg\max_{a \in A} \sum_{s' \in S} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')] \tag{3.2}$$

Thus, the value in state s is equal to the maximum Q-value $\forall a \in A$

$$V(s) = \max_{a \in A} Q(s,a) \tag{3.3}$$

or simply the Q-value of the action selected by policy $\pi(s)$.

$$V(s) = Q(s, \pi(s)) \tag{3.4}$$

Similarly, the state-action value function can be defined as the value obtained by actions that maximize the value function.

$$Q(s,a) = \sum_{s' \in S} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]. \tag{3.5}$$

3.3.3 Offline approximate dynamic programming solution

The above formulation and algorithms assume a discrete state set S. If the state space is discrete, then V and Q can be represented as value arrays, one record for each discrete state. The arrays are initialized arbitrarily and iteratively optimized. Many real word state spaces however are continuous such that a discrete representation of them becomes intractable with respect to current computational and memory resources. The approach also requires us to perform a time-consuming value iteration backup for each state.

We address these complexities by representing Q using an estimation architecture ($\widehat{Q}$). The estimation architecture stores the state action-value function in a compressed form, such as a linear function over problem variables, rather than an explicit combination of each possible variable assignment. When used in approximate value iteration (Bertsekas and Tsitsiklis [1996a]), the estimation architecture alleviates the storage problem,shares information across state variables and thus reduces learning time. The estimation architecture therefore resolves both the value iteration validation and the state space proliferation problems inherent in the overall approach.

Algorithm 1 outlines the approximate value iteration algorithm where $\widehat{Q}$ is the approximate value function represented by an estimation architecture. Algorithm 1 takes as input the MDP tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where $|\mathcal{S}|$ is very large, possibly infinite. Algorithm 1 begins by randomly initializing state-action value table $\bar{Q}(s,a)$ over state subset $\bar{S}$. The approximation architecture $\widehat{Q}$ is then initialized based on the $\bar{Q}$ table. We then perform a Bellman backup over state subset $\bar{S}$, using $\widehat{Q}$ for future state-action value estimates. The newly computed state-action values are stored in table $\bar{Q}$ and the policy, approximation architecture and state value table are then updated; this process repeats until a convergence threshold is met. Algorithm 1 returns approximation architecture $\widehat{Q}_t$.

Algorithm 1 Approximate state-action value iteration algorithm where $\widehat{Q}$ stands for an approximation architecture representation of the state-action value function, $\bar{Q}$ stands for a lookup table of state action values over a subset $\bar{S}$ of the full statespace S.

```
ApproxStateActionValueIteration($\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$) $\Leftarrow$ {
for all $s \in \bar{S} \subset S$ do
  for all $a \in A$ do
    Initialize ($\bar{Q}(s,a)$)
  end for
end for
InitializeApproximationArchitecture($\bar{Q}_t, \widehat{Q}_t$)
$t \Leftarrow 0$
repeat
  $t \Leftarrow t + 1$
  for all $s \in \bar{S} \subset S$ do
    for all $a \in A$ do
      $\bar{Q}(s,a) \Leftarrow \sum_{s' \in S} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma \max_a \widehat{Q}(s',a)]$
    end for
  end for
  $\widehat{Q} \Leftarrow$ UpdateApproximationArchitecture($\bar{Q}, \widehat{Q}$)
  for all $s \in \bar{S}$ do
    $\pi(s) \Leftarrow \arg\max_a \widehat{Q}(s,a)$
    $\bar{V}_t(s) \Leftarrow \bar{Q}(s, \pi(s))$
  end for
until $\max_{s \in \bar{S}} |\bar{V}_t(s) - \bar{V}_{t-1}(s)| < \epsilon$ , $\forall s \in \bar{S}$
return $\widehat{Q}_t$
}
```

3.4 Decision uncertainty minimization defined

Decision uncertainty minimization consists of model-based reasoning concerning the value of a decision and its leading alternatives, in order to remove as much uncertainty about the correctness of that decision as possible. A decision uncertainty method defines a reasoning process, identifies its steps, and constructs a definition of uncertainty for it to minimize.

In the context of our autonomous approaches to mission planning, ”reasoning” manifests itself as performing online simulation, for the purpose of disambiguating the relative value of an agent’s immediate options, such as real world control actions. The agent may be unsure about the value of each option. Yet, it can reevaluate a given option online by simulating the state in order to improve its value estimate. If the outcome is stochastic, it would simulate it multiple times and compute the expected outcome from multiple simulations. This reevaluation method is sometimes referred to as rollout Bertsekas [2005a].

DUM Planning example

Decision uncertainty minimization can be summarized as the question, “If an agent can reevaluate just one option, which one would most likely change the agent’s decision?” Before we formalize the answer to this question mathematically, it behooves us to concisely restate the DUM approach and walk through an example of how DUM applies to the real world. The DUM planner consists of an offline policy learning algorithm and an online reoptimization algorithm. The offline algorithm learns a policy before the mission starts, and the policy suggests paths for the agent to follow. In other words, the policy suggests what to do in a given situation. The reoptimization algorithm reevaluates these suggestions and selects the best one according to mission objectives and constraints.

To understand why we combine these two methods, consider the everyday task of commuting home from work. The regular nature of this task has led to development of a policy (conscious or otherwise) about what to do at a given intersection, such as turning left or right. On occasion, a commuter will encounter a situation about which their policy is uncertain. Traffic jams, construction or a funeral procession pose a unique challenge to the commuter’s policy. A commuter is likely to reason offline by referring to prior experience concerning these situations and reason online about their various options by checking traffic conditions, surveying the congestion, etc. These operations entail forward simulation or projection of the consequences of alternative actions. Congestion in a given direction would lead a commuter to project the consequences of traveling to the right, and what to do after that. The forward simulations are guided by a policy (e.g., wanting to get home in time for dinner), but they explicitly elaborate options and estimate their consequences.

Mathematical example

To translate these methods into mathematical language, we first assume that an agent's immediate options consist of action set $a \in A$. Its immediate choice will set the agent on a course of action dictated by a previously obtained offline policy $\pi(s)$, where s is the agent's current state. The agent may thus choose an initial action a^0, but all future actions $a^{t>0}|t > 0$ are selected by policy $\pi(s^{t>0})$, such that $a = \pi(s)$.

For every state, we also have a prior estimate of the value of each action $\widehat{Q}(s, a)$, where the state-action value captures the expected reward for the immediate action a, plus the future course of action determined by policy $\pi(s)$. We do not know the exact value that $\widehat{Q}(s, a)$ represents, primarily due to representational errors. The value function is approximated by an analytical function in order to contend with the large state space and memory limitations. Our uncertainty over estimate $\widehat{Q}(s, a)$ is defined by the probability density function $pdf_{\widehat{Q}(s,a)}$. The function $\widehat{Q}(s, a)$ is a random variable (see Figure 3-2).

Given the opportunity to perform an online simulation of an immediate action a_i and follow on actions $a_\pi = \pi(s)$, the simulation will provide us the true value (denoted as $Q_\pi(s, a)$), as previously estimated by $\widehat{Q}(s, a)$. But how do we decide which course of action to simulate? We do this by evaluating each challenge action and then determining the action expected to most strongly influence the value of an incumbent action. That action warrants reevaluation. In practice, this means determining whether the probability that the true value of an alternative action a_c (the challenger), is actually higher than the true value of the incumbent action $a_\pi = \pi(s)$. First, consider the probability that random variable $\widehat{Q}(s, a_c)$ exceeds random variable $\widehat{Q}(s, a_\pi)$ by exactly t:

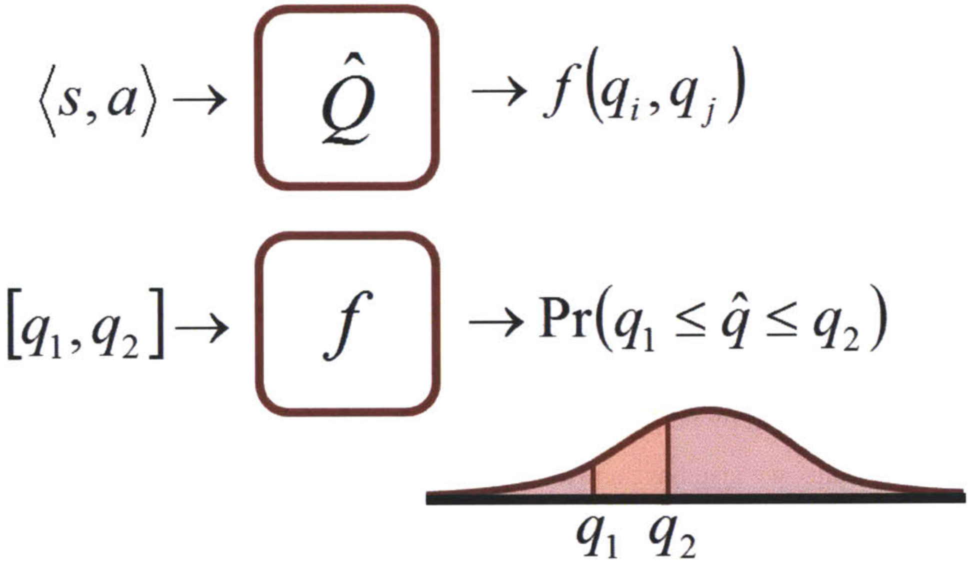

$\langle s,a\rangle \rightarrow$
$\hat{Q}$
$\rightarrow f(q_i,q_j)$
$[q_1,q_2]\rightarrow$
f
$\rightarrow \Pr(q_1 \leq \hat{q} \leq q_2)$
q_1 q_2

Figure 3-2: This figure shows how we map our value function into a more manageable function that gives us an actionable value. Our state-action value function, indicated by the letter $\widehat{Q}$, takes as input a state 's' (the situation an agent is in), and action 'a' (what we are going to do). $\widehat{Q}$ then provides the agent with a probability distribution function f (or $pdf_{\widehat{Q}(s,a)}$), which subsequently provides the probability distributed between two possible state-action values, q_1 and q_2.
The state-action value is the discounted future reward of a course of action that follows from taking the action under consideration from the initial state (assuming an optimal policy). The pdf in other words, describes the benefit of the projected future state. The function $Q(s,a)$ provides us with an additional function that describes the probability that the value of that state-action is between q_1 and q_2. The heuristic thus provides us with a PDF probability density function over the best action value. An agent can use this method to assess the relative merits of every projected state and then determine the optimal course of action from its current state.

$$pdf_{[\widehat{Q}(s,a_c)-\widehat{Q}(s,a_\pi)]=t} \quad = \int_{-\infty}^{\infty} pdf_{\widehat{Q}(s,a_c)}(\tau)\, pdf_{\widehat{Q}(s,a_\pi)}(t+\tau)d\tau$$

Integrating over positive values of t gives us the probability that a challenger's action value exceeds the incumbent action value:

$$\begin{aligned} &\mathrm{P}\left(\widehat{Q}(s,a_c) \geq \widehat{Q}(s,a_\pi)\right) \\ &\quad = \int_0^{\infty} \int_{-\infty}^{\infty} pdf_{\widehat{Q}(s,a_c)}(\tau)\, pdf_{\widehat{Q}(s,a_\pi)}(t+\tau)d\tau dt \end{aligned} \tag{3.6}$$

The above calculation allows us to then reevaluate $\widehat{Q}(s,a_c)$ via simulation if it is likely to supersede the incumbent. Simply exceeding the incumbent value however does not necessarily provide the most worthy reevaluation. Instead we wish to identify those actions whose value greatly exceeds that of the incumbent. Because we are optimizing relative to the challenger, we only address margins that are positive. A positive value merits further consideration, whereas a negative value does not. A layered rock formation imaged at a distance by an agent on Mars for example, might offer a significant probability of improvement over a path of action leading away from the formation. Expected marginal improvement t is averaged over situations that offer an improved value.

$$\begin{aligned} &\mathrm{E}\left[\widehat{Q}(s,a_c) - \widehat{Q}(s,a_\pi)|\widehat{Q}(s,a_c) \geq \widehat{Q}(s_i,a_\pi)\right] \\ &\quad = \frac{\int_0^{\infty} t\left(\int_{-\infty}^{\infty} pdf_{\widehat{Q}(s,a_c)}(\tau) pdf_{\widehat{Q}(s,a_\pi)}(t+\tau)d\tau\right)dt}{\int_0^{\infty} \left(\int_{-\infty}^{\infty} pdf_{\widehat{Q}(s,a_c)}(\tau) pdf_{\widehat{Q}(s,a_\pi)}(t+\tau)d\tau\right)dt} \end{aligned} \tag{3.7}$$

The above equation computes the degree of expected marginal improvement when there is an improvement. In the example of the layered rock formation, the agent would estimate the value of potential features associated with layered rocks (signs of water, chemical signatures) according to the probability of locating them, relative to the incumbent path of action that ignored the features. We also want to know the frequency of improvement; this requires the above to be normalized by the probability that there is an improvement:

$$\begin{aligned} &\mathrm{E}\left[Q_\pi(s,a_c) - \widehat{Q}(s,\pi(s))|\widehat{Q}(s,a_c) \geq \widehat{Q}(s,\pi(s))\right] \times \mathrm{P}\left(\widehat{Q}(s,a_c) \geq \widehat{Q}(s,\pi(s))\right) \\ &\quad = \int_0^{\infty} t\left(\int_{-\infty}^{\infty} pdf_{\widehat{Q}(s,a_c)}(\tau)\, pdf_{\widehat{Q}(s,\pi(s))}(t+\tau)d\tau\right) dt \end{aligned} \tag{3.8}$$

Finally, we wish to select an alternative action, relative to the incumbent, that maximizes the above equation. This equation represents the greatest value that the agent could possibly realize through an alternative action, or a reward integrated with the probability of realizing that reward.

$$\begin{aligned} &\arg\max\nolimits_{a_c} \mathrm{E}\left[Q_\pi(s,a_c) - \widehat{Q}(s,a_\pi) | \widehat{Q}(s,a_c) \geq \widehat{Q}(s,a_\pi)\right] \times \mathrm{P}\left(\widehat{Q}(s,a_c) \geq \widehat{Q}(s,a_\pi)\right) \\ &\quad = \arg\max\nolimits_{a_c} \textstyle\int_0^\infty t\Big(\int_{-\infty}^\infty pdf_{\widehat{Q}(s,a_c)}(\tau)\, pdf_{\widehat{Q}(s,a_\pi)}(t+\tau) d\tau\Big)\, dt \end{aligned} \tag{3.9}$$

Decision uncertainty minimization summary

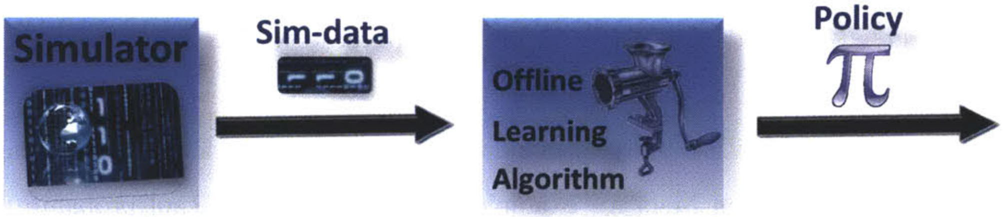


Figure 3-3: The offline policy learning algorithm develops a policy offline by simulating numerous trajectories through the world. That simulated data is collected and passed to a learning algorithm, which uses approximate dynamic programming or value iteration to learn a policy. The policy captures the value of taking an action in any situation, as well as the estimation accuracy of that value. By doing so, the policy can determine the uncertainty of selecting a given action over another. The policy is used and possibly reoptimized online for a given situation.

DUM seeks the optimal action to take and the optimal way to sample the unknown action values in order to make this action. At each step, DUM selects an action for a known reevaluation that is most likely to be better than the incumbent action. This choice favors high uncertainty actions with estimated expected values that are not significantly worse than the incumbent action. As a recursive process, these operations maximize the chance of discovering the best course of action with the highest value following from real-world observations. The *a priori* information about mean decision value and uncertainty provide a sampling that reveals the maximum posterior decision value.

This section began with the objective of minimizing uncertainty regarding the optimal-

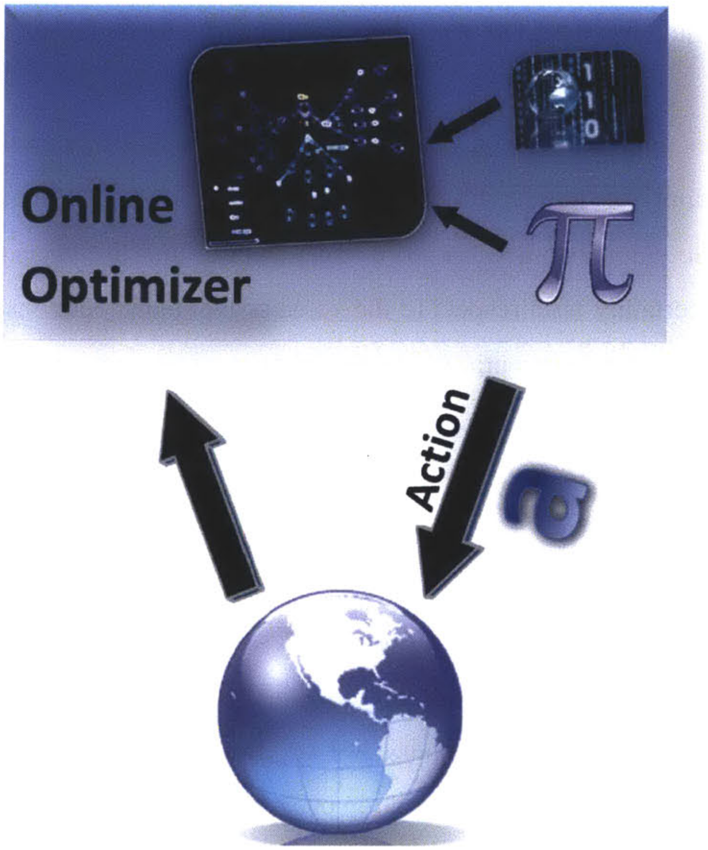


Figure 3-4: Online reoptimization algorithm: The online optimizer starts with policy π, which is modeled offline. Combined with the world simulation, the agent can reoptimize for the current situation. For the given situation, policy π can convey its confidence in choosing one action over another, as well as which action value is most in question. Using this information as a guide, the online optimizer will simulate that action followed by a series of subsequent actions (chosen in the same manner) in order to re-estimate the value of that trajectory.

ity of a given control action. The control action selections are provided by base policy $a_\pi = \pi(s) = \arg\max_a \widehat{Q}(s, a)$, which can reevaluate an action at run time. Given sufficient time to perform an online simulation of an immediate action a_i and execute actions a_π, the simulation will provide us the true value of $\widehat{Q}(s, a)$ denoted as $Q^*(s, a)$. The procedure now requires specification of which COA to simulate. The DUM algorithm provides this information by simulating the COA that minimizes uncertainty regarding the next optimal action.

DUM improves upon the accuracy-complexity tradeoff curve by measuring and representing the uncertainty of the learned off-line policy. This uncertainty can then guide the online Monte Carlo planner to actions more likely to both improve upon the current plan and reduce the uncertainty with respect to the value of that plan. Sections 3.7 and 3.8 describe the DUM algorithm in its entirety. Appendix B provides a review of online planning and search.

3.5 Estimating $\widehat{Q}(s, a)$

This section presents algorithms for estimating $\widehat{Q}(s, a)$ in terms of functions that are a normal distribution about a mean $Q - hat(s, a)$. ADP estimates the state action-value function in a compressed form (Appendix A.4 reviews ADP.) A common example of a compressed functional form is a linear function over problem variables. The linear function assigns one value to each variable, representing the state more concisely than if it stored a value for every possible combination of assigned variable values. This estimation alleviates the storage problem and reduces learning time by sharing information across state variables.

While the above approximation method reduces memory requirements and learning time, it may not adequately *represent* the state-action value function. This *representational uncertainty* results in a suboptimal policy solution. The ensuing chapter addresses the subject of identifying a good representation for input into ADP algorithms. The remainder of this chapter frames representational uncertainty as a concept, reviews existing online search algorithms, and uses representational uncertainty to blend offline and online computation.

3.5.1 The accuracy of a policy depends on how it is represented

Agents routinely consider uncertainty within the environment, in order to make better decisions. Agents make optimal decision by taking into account uncertainty both in the dynamics of the system and about the current state of the system. The value of an action outcome consists of the immediate benefit of taking a given action, plus the future benefit of subsequent actions naturally stemming from this initial action:

$$\begin{aligned} Q(s,a) &= \arg\max_{a \in A} \mathrm{E}[\mathcal{R}^a_{ss'} + \gamma V(s')] \\ &= \arg\max_{a_i \in A} \mathrm{E}[\sum_{i=0}^{T} \gamma^i \mathcal{R}^a_{s_i s_{i+1}}] \end{aligned}$$

$$Q(s,a) = \arg\max_{a_i \in A} \mathrm{E}[\sum_{i=0}^{T} \gamma^i \mathcal{R}^{a_i}_{s_i s_{i+1}}] \tag{3.10}$$

This formulation assumes that the expectation of an action outcome can be precisely estimated. However if an agent is uncertain about an expected action outcome value, then it is likewise uncertain of which action to take. When using ADP, the inaccuracy of the value function stems from the underlying representation and can be modeled. For any representation projected onto a reduced subspace, multiple points having different values can be mapped onto the same point in the subspace. This results in uncertainty regarding the true value of that point.

The notion of representational uncertainty is foreign to the standard model of the state-action value function. Representational uncertainty can easily be integrated into approximate state-action value iteration however, by using an approximation architecture that learns a value *distribution* rather than just an expectation function. The architecture would return a distribution estimate, such as a normal distribution parameterized by its mean and variance:

$$\mathcal{N}(\mu_{s,a}, \sigma^2_{s,a}) \Leftarrow \widehat{Q}(s,a) \tag{3.11}$$

3.5.2 Using the representational uncertainty of a state-action value function to enhance the blending of offline and online computation

We incorporate representational uncertainty into the approximate state-action value iteration process so as to learn a distribution estimate over the values. Our offline value function returns these distributions, not just an expectation function. This can be used by the online algorithm when deciding between possible actions to specifically investigate the values of less-known, yet viable alternative state-action pairs. In short, we can use an online search algorithm that is guided by the state-action value function. We discuss this mechanism in the next section.

Before moving on, we will first summarize the offline algorithm as background for the ensuing sections. The agents use approximate dynamic programming to create a policy for carrying out specific actions in the world. This operation uses a value function or policy that is computed offline and approximately through value iteration before the mission starts. The platform simulates the mission, saving simulation snapshots as data points. It then generates an approximation architecture for each iteration by data point regression, thus accounting for representational uncertainty. Before exploring how this value distribution can be combined with real-time search to select an action, we first review the set of relevant online search algorithms.

3.5.3 A value function distribution captures the representational uncertainty over the expected outcome of a course of action

The unmodeled and unrepresented component of the value function is a random variable. Suppose we wish to model a function given a complete set of inputs X and outputs y, with no observation error in X or y. For example, the state of an MDP is perfectly known and an expectation over the output value ($\bar{V}$)is computed for each state. Suppose that our value function (V) can be perfectly approximated by choosing from the set of linear functions estimated from inputs $X \in \mathbb{R}^{|x|}$ and outputs y as follows:

$$\widehat{\beta} = (X^{\mathsf{T}}X)^{-1}X^{\mathsf{T}}y$$
$$where \quad \hat{y} \Leftarrow X\widehat{\beta}$$

The vector of residuals $|y - X\widehat{\beta}|$ are orthogonal to the column space of X and arise from the representational limitations of the linear function. The errors can be described by a distribution function. Under ideal assumption, these errors are uncorrelated with the inputs. The estimate has a constant variance across the entire function.

A value-function learned from the *expected* outcomes thus can also capture the uncertainty of the value estimate. The residual is a random variable (ϵ) where:

$$\begin{aligned} y &= \hat{y} + \epsilon \quad and \\ y &= X\widehat{\beta} + \epsilon \end{aligned}$$

In particular, in our algorithm we describe the residuals of a state-action value function with a Gaussian distribution of zero mean and variance σ^2_{sa}. The function also returns the mean value:

$$\mathcal{N}(\mu_{s,a},\, \sigma^2_{s,a}) \Leftarrow \widehat{Q}(s, a) \tag{3.12}$$

However, unlike the linear least-squares regression model, our variance is conditioned on the function inputs. Specifically, the linear model $y = X\widehat{\beta} + \epsilon$ ascribes a variance of σ^2 where $\epsilon \sim \mathcal{N}(0,\, \sigma^2_\epsilon)$ ascribes a single consistent variance over the span of the function. However, we use a function without this limitation; a function-approximation method that is deliberately more certain at some parts of the state space than others.

3.6 Guided versus unguided search algorithms

Our operations with Equation 3.12 facilitate online re-estimation steps. Re-estimation computes an expected value over a simulated subset of possible future states that result from taking an action. Before exploring this subset or subtree, this section reviews guided and unguided search algorithms that can be combined with uncertainty estimates to perform simulation based re-estimation followed by action selection.

3.6.1 Unguided Algorithms

Policy tree: The fully elicited policy tree is the most general, unguided but complete algorithm. A policy tree consists of the simulation of every possible action and future state resulting from those actions. This full elicitation of all futures is evaluated and maximized at each time-step, starting from the final step and working backwards, such that the optimal action is chosen in each situation. The solution is equivalent to exact dynamic programming. The result is an optimal full-feedback policy. A walk-through of this algorithm is given in Subsection D.2. Although this algorithm is computationally intensive, it elucidates the general structure of the optimal full-feedback policy and how it is generated. Each of the algorithms described below emulate this policy tree solution, but in a more efficient way. Efficiency is achieved by sampling from the policy tree. The first of these methods is Monte Carlo sampling based planning.

A Monte Carlo planning algorithm samples from the policy tree by only simulating a limited number of randomly chosen future states. Monte Carlo planning is a combination of Monte Carlo analysis and forward search. While Monte Carlo analysis evaluates a system under a given policy, Monte Carlo planning compares different policies. Like the policy tree, Monte Carlo planning considers all actions within each state. It evaluates each action, as does the policy tree. Efficiencies are realized by limiting the degree to which we simulate the consequences of each action. Monte Carlo sampling limits the number of times it samples the consequences and value of a policy choice to n forward samples per evaluated action. Both of the algorithms above share these limitations in the number of samples n. Details of this algorithm are presented in Appendix D.3.

The open-loop approximation: Open-loop planning simplifies the Monte Carlo analysis by limiting it to specific plans. An open loop plan consists of a sequence of actions that does not adapt to feedback from the environment. An extension of open-loop planning, the open-loop feedback control algorithm, executes only a selected plan for a discrete time interval. It then replans using the same open-loop plan comparison methodology. Consequently, open-loop feedback control responds to stimuli in the environment, but does not make plans that optimize with respect to future contingencies. Rather, open-loop feedback control is optimized to specific hard future commitments. By doing so, the Monte Carlo simulation and evaluation process is much easier. By ignoring contingencies, evaluation consists solely of simulating an action sequence multiple times. The maximization is only performed over the whole sequence, not at each step. Therefore, the analysis increases linearly rather than exponentially with mission length. Details of this algorithm are presented in Appendix D.5.

The certainty equivalence approximation: Monte Carlo sampling limits how thoroughly we sample each policy choice to n forward samples per evaluated action. Open-loop feedback control limits the analysis by ignoring future contingent choices. The next algorithm reduces the complexity of its analysis by assuming that the utility of the expected outcome is equal to the average utility over all possible outcomes. The insurance industry developed this approach and refers to it as certainty equivalence. This assumption allows us to only simulate the nominal or expected path for a given policy or plan. It simplifies the analysis by converting it to a deterministic problem. Assuming certainty equivalence is tantamount to assuming that the system evolves deterministically. Policy and plan are the same in certainty equivalent environments. The certainty equivalent approximate solution is therefore more efficient to calculate than the open loop feedback control solution. The certainty equivalent approximate solution may perform poorly however, especially if system disturbances are chaotic, and the appropriate reaction to these disturbances drastically deviates from the nominal plan. This is common in information gathering tasks such as tracking, where the underlying process evolves chaotically and the uncertainty about that process follows a similar path. Details of this algorithm are presented in Appendix D.4.

3.6.2 Guided Algorithms

The techniques mentioned so far—policy tree, Monte Carlo planning and the open-loop and certainty-equivalent simplification—are all *unguided* search algorithms. These methods do not benefit from heuristics, value functions or other tools for guiding the forward search toward favored actions. The primary shortcoming of the algorithms described above is that, in the absence of guidance, they fail to optimize the simulation process. The remaining algorithms described in this section are guided algorithms (rollout, extended rollout and DUM). Guided search algorithms generally follow cues regarding an optimal direction, in the form of a value function. A guided algorithm can use past knowledge to inform where future simulations can be performed, thereby, focusing the search on options that matter, and areas where it is unclear which decision is best.

Standard Rollout

Rollout offers a way to compare polices that differ only in their initial action choice. The comparison requires a Monte Carlo analysis of each policy. The term "rollout" comes from the backgammon community, where it is sometimes used to compare two playing strategies. To do this, one would perform Monte Carlo analysis on each available action from the current board position and strictly following the original policy thereafter. In a backgammon rollout, the die is cast many times in order to execute this alternate strategy. Rollout results are computed by averaging the outcome of each game. Rollout is the preferred method of comparing two strategies, such that backgammon experts use it as a justification for strategy change.

Policy Iteration

In the MDP framework, we often learn a value function offline and determine the policy at runtime by maximizing the dynamic programming equation. If we compare via rollout all possible actions from a given state via rollout and then select the best one (Bertsekas and Castanon [1999]), the expected outcome is always better than, or at least as good as the choice made by the the base policy. Therefore, it stands to reason that choosing the action at runtime via rollout would be better than the traditional policy-extraction method.

Standard Rollout and nested rollout

Rollout emulates traditional search algorithms except that it only branches on actions at the very first level. It thus samples from the policy tree by severely limiting its policy choices. In addition, it utilizes Monte Carlo sampling of each policy, instead of a complete evaluation. These restrictions make rollout efficient but limited. Nested rollout extends the rollout concept by allowing branching at multiple levels (see Rosin [2011]). For example, nested rollout may branch at two or three levels. This makes nested rollout more costly, but also more effective. The secret is to make the right trade-off between cost and effectiveness. Details of this algorithm are presented in Appendix D.6 and Subsection D.7.

3.7 Using policy uncertainty to guide online planning

After consideration of the relevant algorithms that can perform evaluation, we want to devise an algorithm with the strengths of a policy tree, but one that is more efficiently calculated and guided by uncertainty. Two operational steps help achieve these goals. First, we reevaluate branches through Monte Carlo sampling and second, we only reevaluate some of the branches. Decision uncertainty minimization thus relies upon online reevaluation to compare policy choices at each sequential decision point, in order to enhance the base policy. What we have just described is nested rollout with limited branching. This thesis presents the novel innovation of utilizing the value function distribution to decide which branch to follow.

We want our approach to avoid the computational overhead of previous algorithms. Our approach uses Monte Carlo sampling when reevaluating a branch, because we want to measure the stochastic system dynamics, including the future policy choices, which are contingent on the state that results from an action. By performing Monte Carlo reevaluation, we avoid an open-loop approximation and a certainty-equivalent approximation. We use a sufficient number of Monte Carlo samples to measure the expected value accurately, but we avoid a costly complete reevaluation. Our algorithm evaluates choices more thoroughly than standard rollout, because it compares choices at every decision point. However, we compare fewer choices at each level than pure nested rollout. Yet our selection of actions to reevaluate is more intelligent, because it relies on the offline value distribution to make these choices.

During real-time search, our algorithm chooses which actions to explore by evaluating uncertainty over the value of each COA. This corresponds to uncertainty with respect to the optimal decision. We want to focus our computational energy on areas for which the optimal decision is unclear. For example, an action that is nearly as good in terms of expectation, as the best action, but whose action value is uncertain, would make a good candidate for reevaluation. By selecting in this way, we restrict the search to branching over actions and focus on actions of greater consequence, overcoming some of the shortcomings of the other methods.

DUM Outline

The process of online reevaluation involves generating a tree of candidate actions sequences to test. We generate the tree of paths with the help of the offline-generated state-action value-function. Starting from the current state, the algorithm selects b actions to simulate, where b is called the branching factor. Once an action is simulated we arrive at a new state. At that point, the algorithm will reach b new states. From each of those new states, the algorithm again selects b actions to simulate. This process proceeds to a predetermined planning horizon h. Collectively, the algorithm simulates $\frac{b^{d+1}-1}{b-1}$ paths. The algorithm's task is to re-estimate the value of each action sequence. Thus, the algorithm adds up the cumulative reward garnered from each path. If the planning horizon is long enough to reach the end of the mission, then the remaining value is estimated using the offline state-action value-function.

Branching Factor

The above algorithm uses branching factor b and horizon h. While h is a parameter of the algorithm, b is calculated by Algorithm 2 (see below). We select b and h such that we have enough time to reevaluate c paths. To do this, we first determine how many computations can be performed in the allotted time between decisions. We then select a planning horizon and calculate a branching factor that will result in that number of calculations. Algorithm 2 finds a branching factor b that matches our search algorithm to our computational allowance c, given search horizon h. We know that our search algorithm will perform $\frac{b^{d+1}-1}{b-1}$ computations because the search tree uses b as follows. At the initial state (first level), we explore b actions or branches. The term b can be a fraction, in which case we explore b branches on average. At the next (second) level, we explore b branches for each of those branches. Thus, we explore b^2 branches at level 2, and so on. Cumulatively, we explore $\sum_{i=1}^{d} b^i$ paths.

Algorithm 2 Compute Branching Factor: This algorithm computes the branching factor b that produces a tree with c nodes given a tree depth of d.

```
ComputeBranchingFactor($d \in \mathbb{N}, c \in \mathbb{N})$) $\Leftarrow$ {
    $\tilde{c}_{bd} = \sum_{i=0}^{d} b^i = \frac{b^{d+1}-1}{b-1}$
    $b \Leftarrow \arg\min_b |\tilde{c}_{bd} - c|$
return $b \Leftarrow\in \mathbb{R}$
}
```

$$c = \sum_{i=0}^{d} b^i \\ = \frac{b^{h+1} - b}{(b-1)} \tag{3.13}$$

Using a standard minimization algorithm we find a branching factor that most closely satisfies Equation 3.13. Figure 3-5 shows the range of values for a particular problem.

3.7.1 Selecting an action to reevaluate

A central question during each step of the simulation is how to choose the actions to reevaluate. We elect to evaluate actions that have a promising outcome, weighing the certainty of

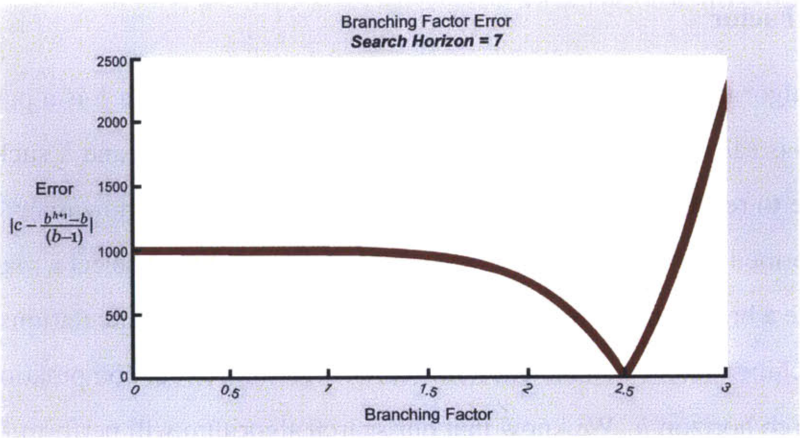


Figure 3-5: Branching Factor Graph: Shows the difference between the computation allowance and the number of computations that a range of branching factors would require. The example uses a search horizon of 7 and a planning tree of 1,000 computations.

the estimated outcomes. At each step in the algorithm described above, we select b actions to simulate, based on the offline state action value-function, which estimates the future cumulative rewards of each action, if taken from the current state. These estimate include a distribution, described by a mean and variance, which captures how well we know a given value. A high-variance distribution means that the value is not precisely known, while low variance means it is precisely known. We select a sample from these distributions, one for each action, and then choose the action with the highest sample value. If one of the action-distributions consistently produces a high sample value, this tells us that we have little reason to evaluate other options. However, if there is a state-action value distribution with an especially high variance, it sometimes produces a sample with the highest value, even though its mean is lower. This phenomenon exactly mirrors the probability that said action is the best, given what we know. In other words, we explore the actions in proportion to their viability and our certainty of it. We may also evaluate apparently less-viable actions, where high variance warrants reexamination.

Stepping through action Subset Selection The subset selection Algorithm 3 operates as follows. In Line 1, the mean branching factor, state-action value distribution function and the current state are passed in. Mean branching factor $\bar{b}$ is essentially the parameter of random variable B defined on Line 3, which returns the integer just above or just below

$\bar{b}$, with an expected value of $\bar{b}$. Line 2 initializes the action subset being selected. Line 4 through 9 is a loop that selects b actions, where b is the current value of random variable B. Line 5 through 7 loops through all action value estimates as determined by the state-action value distribution function $\widehat{Q}(s, a)$ on Line 6. Line 8 adds the new action to the subset. Line 9 returns the selected action subset.

Algorithm 3 Action Subset Selection

```
 1: ActionSubsetSelection($\bar{b}, \widehat{Q}, s$) $\Leftarrow \{$
 2: $\bar{A} = 0$
 3: $B \sim \mathcal{U}(\lfloor \bar{b} \rfloor \lceil \bar{b} \rceil)$
 4: for $count = 1 : b \sim B)$ do
 5:     for all $a \in A$ do
 6:         $\bar{Q}(s, a) = \widehat{Q}(s, a)$
 7:     end for
 8:     $\bar{A} \cup \arg\max_a \bar{Q}(s, a)$
 9: end for
10: return $\bar{A}$
```

3.8 Decision Uncertainty Minimization Based Planning

In this section we present the DUM planning algorithm based on the approach and equations outlined above. Algorithm 4 operates as follows. In Lines 1 and 2 enter the current state, model, search depth, state-action value distribution function, and mean branching factor are passed in. Again, the mean branching factor $\bar{b}$ is essentially the parameter of a random variable B, which returns the integer just above or just below $\bar{b}$, with an expected value of $\bar{b}$. On average, the branching factor is $\bar{b}$. If the recursive algorithm has reached the end of its search depth, then Lines 3, 4, and 5 return and estimate of the cumulative future rewards for the rest of the mission. Line 6 gets the set of actions to simulate using function `actionSubsetSelection`. Lines 7 and 11 loop through the process of exploring those actions. Line 8 simulates the action to get the next state. Line 8 recursively calls itself (function `DecisionUncertaintyMinimization`) in order to get the estimate of the cumulative future rewards. Line 10 sums up the current and future rewards for the current action. Line 12 returns the best action, together with the action's value estimate.

Algorithm 4 Decision Uncertainty Minimization via time limited branching : Deterministic System Dynamics

```
DecisionUncertaintyMinimization(
    $s \in S, \mathcal{M} \equiv \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle, d \in \mathbb{N}, \widehat{Q}, \bar{b}$    ) $\Leftarrow$ {
if $d = 0$ then
    return $\left\langle \max_{a \in A} \widehat{Q}(s,a), \arg\max_{a \in A} \widehat{Q}(s,a) \right\rangle$
end if
$\bar{A}$ =actionSubsetSelection($\bar{b}, \widehat{Q}, s$)
for all $a \in \bar{A} \subseteq A$ do
    $s'$ $\Leftarrow$DeterministicSimulation($s, a, M$)
    $\langle Q(s'a'), a' \rangle$ $\Leftarrow$DecisionUncertaintyMinimization($s', M, d-1$)
    $\bar{Q}(s,a) \Leftarrow R^a_{ss'} + Q(s',a'))$
end for
return $\langle \max_{a \in A} \bar{Q}(s,a), \arg\max_{a \in A} \bar{Q}(s,a) \rangle$
```

Algorithm 5 The simulation function entails simulating one step of the system, starting at state s and executing action a. The function assumes that the system is deterministic, and that the most likely next state s' has probability mass of 1. In the case that the system dynamics probability matrix $\mathcal{P}^a_{ss'}$ is not fully deterministic, the simulation function will choose the most likely future state. This behavior is consistent with a certainty equivalence perspective. If there is a tie for the most likely state, the function deterministically chooses the first encountered.

```
DeterministicSimulation ($s \in s, a \in A, M \equiv \langle S, A, P, R, V \rangle$) $\Leftarrow$ {
$\mathcal{P}^a_{ss'}$
Select action $s'|s, a$ such that $s' = \arg\max'_s \mathcal{P}^a_{ss'}$
$s' \in S$
where $P(s'|s,a) = \mathcal{P}^a_{ss'}$
return $s'$
}
```

Perspective

Our MDP planner consists of offline policy-learning and online reoptimization. The policy induces action-sequence suggestions for the agents to follow. Reoptimization reevaluates these suggestions and picks the best one. This process actually describes a family of algorithms, because changing the search horizon and branching factor fundamentally changes the algorithm. For example, if we use a very small branching factor with a long horizon, the algorithm closely resembles the rollout algorithm. On the other hand, using use a short horizon with a large branching factor closely resembles model-predictive control. Different configurations will work better for different applications as well as at different points in the mission. Towards the end of a mission for example, broader search could be helpful (larger branching factor) so that the plan properly considers the end goal. Further extensions to our algorithm may include using different branching factors at different levels of the search tree. The function can capture long term objectives more easily than short term details. A small extension would thus allow the algorithm to rely more on offline knowledge in the middle of the mission but more on simulation for starting and ending the mission. The method describing a broad family of algorithms via a set of configuration parameters enables us to tune it, making it broadly applicable.

Algorithm 6 Decision Uncertainty Minimization via time limited branching : Stochastic System Dynamics

```
DecisionUncertaintyMinimization($s \in S, \mathcal{M} \equiv \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle, d \in \mathbb{N}, \widehat{Q}, b) \Leftarrow \{$
if $d = 0$ then
    return $\left\langle \max_{a \in A} \widehat{Q}(s,a), \arg\max_{a \in A} \widehat{Q}(s,a) \right\rangle$
end if
$\bar{A}$ =actionSubsetSelection($b, \widehat{Q}, s$)
for all $a \in \bar{A} \subseteq A$ do
    $\widehat{Q}(s,a) \Leftarrow 0$
    for $n = 1 : N$ do
        $s' \Leftarrow$MonteCarloSimulation($s, a, M$)
        $\langle Q(s'a'), a' \rangle \Leftarrow$DecisionUncertaintyMinimization($s', M, d-1$)
        $\bar{Q}(s,a) \Leftarrow \bar{Q}(s,a) + R^a_{ss'} + Q(s',a'))/N$
    end for
end for
return $\langle \max_{a \in A} \bar{Q}(s,a), \arg\max_{a \in A} \bar{Q}(s,a) \rangle$
```

Algorithm 7 The Monte Carlo simulation function entails simulating one step of the system, starting at state s and executing action a

```
**MonteCarloSimulation** $(s \in s, a \in A, M \equiv \langle S, A, P, R, V \rangle) \Leftarrow \{$
$\mathcal{P}^a_{ss'}$
Select action $s'|s, a$ proportional to $\mathcal{P}^a_{ss'}$
$s' \in S$
where $P(s'|s, a) = \mathcal{P}^a_{ss'}$
**return** $s'$
}
```

Algorithm 8 Compute Monte Carlo Search Branching Factor: This algorithm computes the branching factor b that produces a tree with c nodes given a tree depth of d and a sampling rate of n. Tree depth refers to the lookahead of the search algorithm and therefore includes a layer of action branching and simulation sampling branching.

```
**ComputeMonteCarloSearchBranchingFactor**$(d \in \mathbb{N}, n \in \mathbb{N}, c \in \mathbb{N}) \Leftarrow \{$
    $\tilde{c}_{bdn} \equiv \sum_{i=0}^{d} (bn)^i + \sum_{i=1}^{d} b(bn)^{i-1}$
    $b \Leftarrow \arg\min_b |\tilde{c}_{bdn} - c|$
**return** $b \in \mathbb{R}$
}
```

Monte Carlo Policy Tree Branching Factor

Algorithm 2 (Sec. 3.7 above) computes the appropriate branching factor, when simulating multiple trajectories from a given state-action. The calculation takes a different form however if we are going to simulate and average over n samples at each level. Construction of the policy tree was described above. Construction of a partial policy tree entails simulating only some of the actions available. Because the model analyzes a stochastic system, the outcomes will differ with each trial. We therefore simulate each outcome multiple times in order to determine the best option.

Algorithm 8 computes a branching factor for creating a restricted policy graph. Given computation allowance c, search horizon h, and sample size n, we would like to select an appropriate b. Computation allowance c specifies how many computations can be performed in the allotted time between decisions. The planning horizon is an algorithm parameter meant to be an effective compromise between long-term reasoning and decision speed. We know that our search algorithm will do $\frac{(bn)^{h+1}-bn}{(bn-1)}$ computations because the search tree uses b and n as follows. At the initial state (first level), we explore b actions or branches. Each action is simulated n times. That creates bn branches. At the next (second) level, we explore bh branches for each of those branches. Thus, we explore $(bn)^2$ branches at level 2, and so on. Cumulatively, we explore $\sum_{i=1}^{h}(bh)^i$ paths.

$$c = \sum_{i=0}^{h}(bn)^i \tag{3.14}$$

$$c = (bn)^1 + (bn)^2 \dots (bn)^h \tag{3.15}$$

Multiplying (3.14) by (bn) yields:

$$(bn)c = (bn)^2 + (bn)^3 \dots (bn)^{h+1} \tag{3.16}$$

$$\tag{3.17}$$

Subtracting (3.15) from (3.16) yields:

$$((bn) - 1)c = (bn)^{h+1} - bn \tag{3.18}$$

$$\tag{3.19}$$

Simplify:

$$c = \frac{(bn)^{h+1} - bn}{(bn - 1)} \tag{3.20}$$

$$\tag{3.21}$$

Using a standard minimization algorithm we find a branching factor that most closely satisfies equation 3.20.

3.9 Experiment

We applied DUM to the microscale problem presented in Section 2.4.3. We used two different configurations. In the first configuration (3-6) the mission performance is measured for 100 steps. In the second configuration, mission performance is measured for 200 steps. Four algorithms are compared. The first algorithm is rollout. The second algorithm is DUM with a branching factor of 1, meaning that it only re-examines one alternative. The re-evaluated alternative is compared to the incumbent action value and the apparent best action is selected for execution. The third algorithm is DUM, with a branching factor of 3 at the first step and a branching factor of 1 thereafter. The fourth algorithm is DUM with a branching factor of 3 at the first step, a branching factor of 2 at the second step and a branching factor of 1 thereafter. The DUM algorithm results show a very significant improvement over rollout.

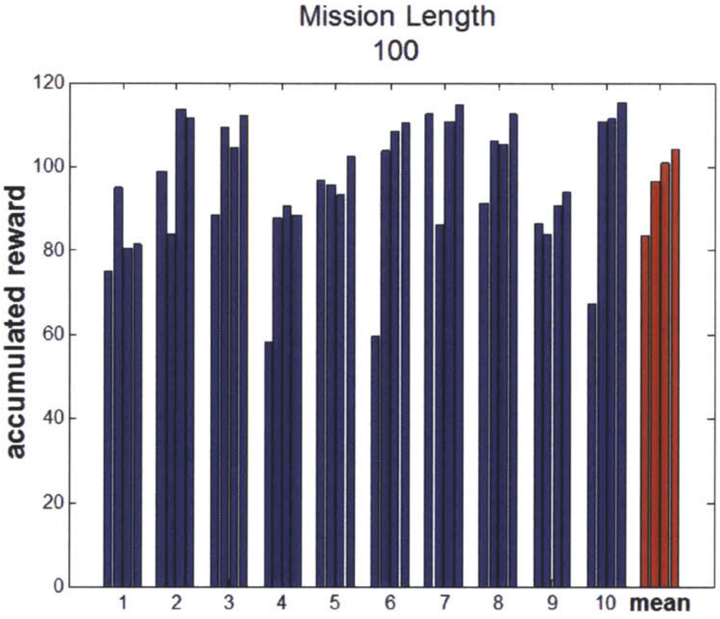


Figure 3-6: Rollout and DUM experiment results from 10 problems with a mission length of 100.

3.10 Summary

Our online planning and re-optimization algorithm uses decision uncertainty to incorporate offline information into online computation. This approach characterizes the state-action value-function as a distribution and uses that uncertainty to guide online reoptimization. It captures this representational uncertainty of the function over the predicted value. Value uncertainty translates this into policy uncertainty or uncertainty with respect to the optimal action. We can reduce the uncertainty regarding our decision via online reoptimization,

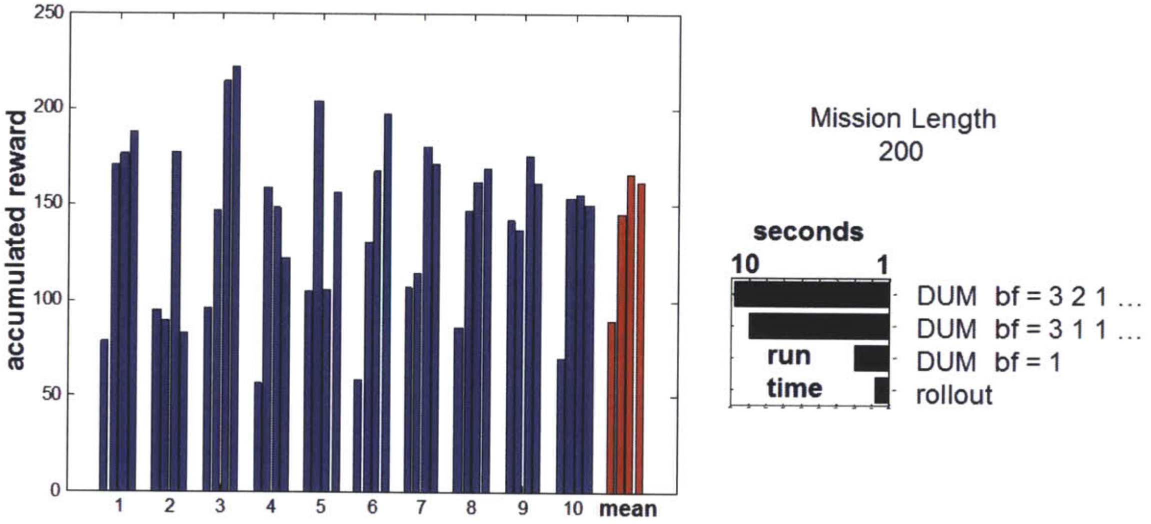


Figure 3-7: Rollout and DUM experiment results from 10 problems with a mission length of 200.

directing computational operations towards decision uncertainty minimization. In other words, we not only optimize the agent's actions, but also its root thinking. Our objective function seeks to perform select computations that contribute most to the decision at hand. To do so, we must characterize our decision uncertainty.

This chapter focused primarily on the framework that integrates online and offline planning and the mechanics of the guided online reevaluation. Representational uncertainty (introduced in Sec. 3.5) refers to the inadequate representation the state-action value function, framed as uncertainty about the respective quantities. Representational uncertainty is a unique form of uncertainty that requires different treatment than other types of uncertainty. The next chapter explores the use of representational uncertainty and ADP to generate the most accurate offline policies and value functions as inputs to the online framework described in this chapter.

4 Representational Uncertainty Minimization

Context: autonomous agents blend online and offline decision methods using value function uncertainty

Intelligent agents make sequential decisions regarding how to interact with the environment by learning a policy offline that can be used online. The policy provides a path of action for any given situation. Section C.2 explains how agents make these decisions using a state-action value function. A state-action value function maps state-action combinations to values which represent the discounted future reward for taking action a in state s. The value of an action outcome consists of the immediate benefit of taking a given action plus the future benefit of subsequent actions stemming from this initial action. State-action value functions can be computationally intensive, so we calculate them offline approximately through value iteration prior to mission deployment. We do so by simulating mission operations, sampling the state and then saving these simulation snapshots as data points.

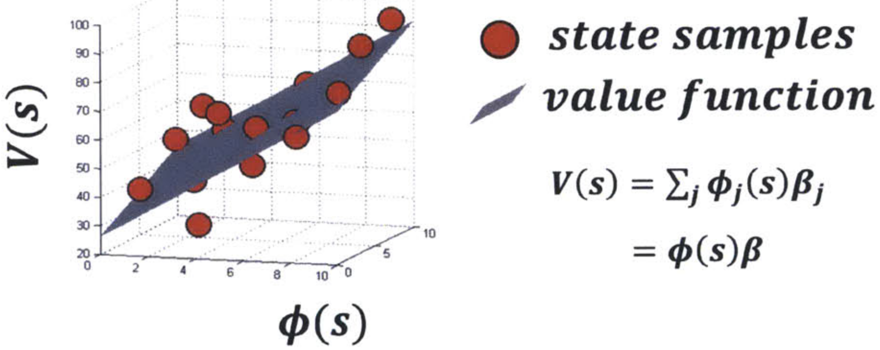


Figure 4-1: The accuracy of approximate methods depends on how they are represented. The linear value function shown here is represented over the basis ϕ. A linear architecture approximates $V(s)$ by first mapping state s to the new basis $\phi(s) \in \Re^k$, then computing the linear combination: $\phi(s)\beta$, where β is a parameter vector.

For most applications the value function is too large to calculate in its entirety. We therefore approximate the function as shown in Figure 4-1. Approximation methods however cannot adequately represent the state-action value function, and this shortcoming can lead to a suboptimal policy solution. The inability to adequately represent the state-action value function can be interpreted as uncertainty about the respective quantities. Uncertainty concerning an expected action outcome value will lead to uncertainty regarding which action to take. The accuracy of a policy thus depends on its representation.

EPISTEMIC VERSUS ALEATORIC UNCERTAINTY

Aleatoric uncertainty (Kiureghian and Ditlevsen [2009]) manifests itself as unknown stochastic outcomes, which differ each time a simulation or experiment is run. For example, our sensing problem system dynamics inject a form of aleatoric uncertainty. It simply exists as part of our dynamical system. We can average out the aleatoric uncertainty, to make a decision, but we cannot get rid of it.

On the other hand, epistemic uncertainty (Ferson *et al.* [2004]) is due to things we theoretically could know but for practical purposes do not precisely know. This is because we have not learned, measured or modeled the value sufficiently accurately. The fact that our value function does not know or represent the true value is a form of epistemic uncertainty. In principle, we could know the expected outcome value of an action and subsequent actions, after averaging out the aleatoric uncertainty. Yet due to the inherent complexities of doing so, we do not.

Decision uncertainty minimization (DUM) methods use uncertainty regarding the optimal action through integration of offline and online computational modelling. Previous sections of this thesis describe how value distributions can be combined with real-time search to select an action. The representational uncertainty of a state-action value function can then be minimized according to online / offline parameters. We incorporate representational uncertainty into the approximate state-action value iteration process so as to model a distribution over state-action values, rather than making do with a mean value point estimate. Our offline value function models these distribution and thus contains more information than a simple mean estimate, as shown in Figure 4-2.

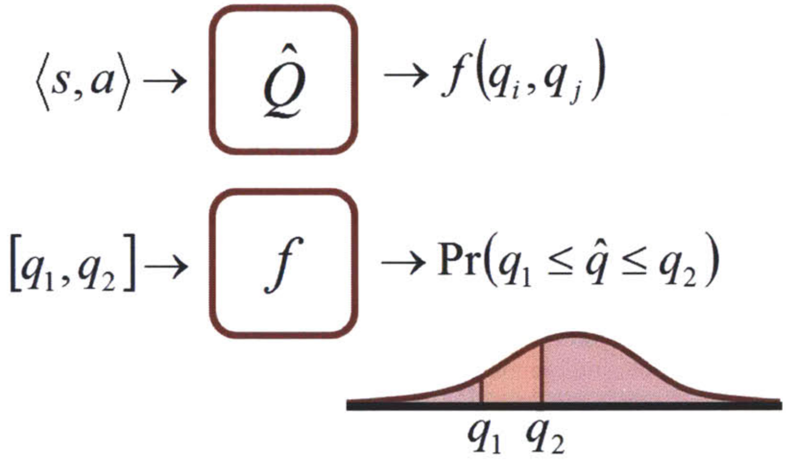

$\langle s,a\rangle \rightarrow$
$\hat{Q}$
$\rightarrow f(q_i, q_j)$
$[q_1, q_2] \rightarrow$
f
$\rightarrow \Pr(q_1 \leq \hat{q} \leq q_2)$
q_1 q_2

Figure 4-2: The figure shows that our state-action value function, indicated by the letter $\widehat{Q}$, takes as input a state 's' (the situation an agent is in), and action 'a' (what we are going to do). $\widehat{Q}$ then provides us with a probability distribution function f (or $pdf_{\widehat{Q}(s,a)}$), which subsequently provides us with the probability mass between two state-action values. The state-action value is the expected discounted future reward of acting out a mission starting from that state and taking that action (presumably under an optimal policy). While the actual outcome is stochastic, the expected outcome is not. Nevertheless, $pdf_{\widehat{Q}(s,a)}$ is a probability distribution function, over what we know the expected future rewards to be. Stated differently, the value is deterministic yet we know that value imprecisely. Based upon that which we know, f can tell us the probability that the value is between two numbers indicated in the picture as q_1 and q_2.

We can then use an online search algorithm that uses the uncertainty when deciding between possible actions. For example, the algorithm may investigate the values of lesser-known, yet viable alternative state-actions.

Minimizing uncertainty representation and estimating the remaining uncertainty

The decision uncertainty minimization algorithm requires an accurate state-action value function representation that communicates its accuracy as value prediction uncertainty. A function approximation approach can provide us with a low uncertainty representation that also captures the remaining uncertainty, which in turn informs us as to the certainty of our prediction. From this point we can identify a representation that minimizes the uncertainty.

As a causal phenomenon, uncertainty in the value function can be modeled. Linear least squares regression for example includes uncertainty measures but can suffer from simplistic assumptions. These types of models calculate uncertainty as an estimate of residuals. When representations are mapped into a reduced subspace, uncertainty arises from points representing different values being projected onto the same point in the subspace. The uncertainty is a trace of information regarding the true value of that point. State-action value functions do not include terms for representational uncertainty. Methods such as linear least squares regression can minimize a measure of uncertainty but may not interpret other distributional parameters according to their implicit uncertainty. Linear least squares regression also assumes that uncertainty is uniform over the entire supported function. It cannot express uncertainty as a variable that is potentially 'surer' of some parts of the function than others.

Fortunately, representational uncertainty can be easily integrated into approximate state-action value iteration by using an approximation architecture that learns a value *distribution* rather than just an expectation function. The architecture would return a distribution estimate when used, such as a normal distribution parameterized by its mean and variance and is defined as follows:

$$\langle \mu_{\hat{q}}, \sigma_{\hat{q}} \rangle \leftarrow \widehat{Q}(s, a) \tag{4.1}$$

$$\hat{q} \sim \mathcal{N}(\mu_{\hat{q}}, \sigma_{\hat{q}}^2) \tag{4.2}$$

In order to apply this decision uncertainty minimization (DUM) to a given value function distribution, the distribution must have a manageable degree of variance. Excessive variance trivializes distribution parameters and renders real world guidance based on those parameters a virtually random activity. More manageable degrees of variance mean a more accurate representation of the state-action value function. To whatever extent the function is inaccurate, we need to quantify its uncertainty about the posterior value estimate to preserve its informational value and avoid risks.

Including representational uncertainty in the approximate state-action value iteration process provides a distributional estimate of uncertainty for each value of the function. Offline value functions described here return this distribution rather than a single expected value. The distribution is thus more useful to the online algorithm as it decides among possible actions to specifically investigate the values of lesser-known, alternative state-action pairs.

Quantifying Uncertainty: Gaussian processes and non-negative matrix factorization of the transition matrix to find a representative architecture

Autonomous exploration operates with an online search algorithm that is guided by uncertainty in the state-action value function. This function must include uncertainty representation and comply with operations that minimize that uncertainty. This chapter explains how to find a state-action value function representation that has low posterior uncertainty with respect to the value estimate. We use a Gaussian process to learn and represent uncertainty distributions over a set of state-action values. We then use non-negative matrix factorization to identify basis functions that accurately capture the structure of the underlying data. This method is prediction-directed rather than operating unsupervised, and can thus capture structure related to value prediction. We therefore apply non-negative matrix factorization to the transition matrix in the search for the best basis. Combining these methods allows us to minimize representational uncertainty.

Section 4.4.2 discusses the origin of estimation uncertainty, while Sections 4.3 and 4.4 explain its mathematical representation and minimization, respectively. Our method combines a basis discovery mechanism (e.g., pattern recognition and clustering) with a smoothing step (interpolator). We specifically use non-negative matrix factorization coupled to a Gaussian process.

4.1 Constructing an approximate architecture

Approximate Dynamic Programming

Approximate dynamic programming (ADP) combines function approximation and dynamic programming to learn the value function of a Markov decision process (MDP). While information-state MDPs (info-MDPs) offer a useful framework, their relatively large state spaces make them impractical for solving real-world problems. If the problem state is represented as discrete variable combinations for example, the number of states will then increase exponentially with respect to the number of state variables. This "curse of dimensionality" leads to prolonged computation time and memory bottlenecks. Value iteration also requires additional backup steps, which also lengthen computation time and memory requirements. Given its size, currently available memory devices cannot store the type of state representation envisioned here. As shown in Bertsekas and Tsitsiklis [1996b]; Farrell and Polycarpou [2006]; Powell [2007]; Schweitzer and Seidmann [1985] and Tsitsiklis and Van Roy [1996], ADP can circumvent these issues.

ADP identifies a compact representation of the value function rather than representing each state value individually. The representation captures the value of interrelated parts of the state space. This facilitates sharing of information between states, which in turn enhances learning.

ADP also represents the value function as a continuous distribution. ADP incorporates calculated model data with every algorithmic iteration and therefore requires an appropriate approximation architecture. Conventions such as neural networks, support vector machine or Gaussian mixture are non-ideal as architecture models due to their computationally requirements. These models also lack properties desirable for ADP, namely self-regularization, tendency to converge and adaptability. Furthermore, the approximator must be self regularizing or resistant to over-fitting by design because cross validation with each iteration can become too computationally intensive. The approximation architecture also requires adequate dynamic programming convergence properties (i.e., a linear architecture) and adaptive capabilities that can capture the initial value function (V^0), the optimal value function (V^*) and all V^k in between. A linear architecture with an adaptive basis function satisfies these requirements.

Linear Architecture

In a linear architecture, the term V is a linear function of some combined representation of the inputs s. In parametric regression we can express our unknown function $V(s)$ in terms of a nonlinear function $V(s;\beta)$ parameterized by β (MacKay [1998]). The linear architecture approximates $V(s)$ first by mapping state s into the new basis $\phi(s) \in \Re^k$. We then compute the linear combination: $\phi(s)\beta$, where β is a parameter vector. An intercept is also typically used. Using basis function set $\phi_i(x)_{i=1}^{|\phi|}$, we can write

$$\hat{V}(s;\beta) = \sum_{i=1}^{|\phi|} \beta_i \phi_i(s) \tag{4.3}$$

DEFINITION

$$\bar{V}_i \leftarrow \tilde{v} \tag{4.4}$$

where

$$\tilde{v} \leftarrow \max_{a \in A} \sum_{s' \in S} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma \hat{V}(s')], \qquad \forall s \in \bar{S} \tag{4.5}$$

and

discount factor $0 \leq \gamma < 1$ makes future payoffs
less valuable than more immediate payoffs.

The parametric function is subsequently used in value iteration to iteratively improve the value estimate. In this case, a value iteration backup is performed over a sampling of states $\bar{S} \subset S$ resulting in the updated set of values $\bar{V}$ and β is subsequently updated to satisfy:

$$\beta \leftarrow \arg\min_{\beta} \sum_i \left\| \bar{V}_i - \hat{V}(\bar{S}_i;\beta) \right\|^2 \quad \text{where } \bar{S} \subset S \tag{4.6}$$

The basis functions are typically nonlinear functions of s. Consider the following radial basis function, which uses cluster centers $\{\mu_i\}_{i=1}^{|\phi|}$:

$$\phi_i(s) = \exp \frac{-(s-\mu_i)^2}{2\sigma^2} \tag{4.7}$$

Basis Function Discovery

Thus, $V(s;\beta)$ is a nonlinear function of s, but a linear function of β. Many other possible sets of fixed basis functions exist. A linear function has the advantage of simplicity compared to, for example, a neural network. Basis functions however can contain extensive representation of other functions. Given its suitability, we dedicate the solution effort primarily towards finding the right basis function for our application, In this case, the basis function should accomplish the following objective:

$$\langle\beta,\phi\rangle \leftarrow \arg\min_{\beta,\phi} \sum_i \left\|\bar{V}_i - V(\bar{S}_i;\beta,\phi)\right\|^2 \quad \text{where} \quad \bar{S} \subset S \tag{4.8}$$

The above equation minimizes the function with respect to a particular set of samples. In actuality, we want to minimize the function over the whole population space from which the data points are drawn:

$$\langle\beta,\phi\rangle \leftarrow \arg\min_{\beta,\phi} \int_{s\in S} \left\|\bar{V}_s - V(s;\beta,\phi)\right\|^2 f(s)ds \tag{4.9}$$

where density function $f(s)$ is the probability that the random variable S falls within a given infinitesimally small range.

.

$$\Pr\left[a \leq S \leq b\right] = \int_a^b f(s)ds \tag{4.10}$$

The integral above weighs each point according to density and assumes that the updated $\bar{V}$ can be calculated from a sampling of the distribution using a value iteration backup,. Various regularization methods allow us to deal with the fact that we do not and cannot use the above idealized optimization function.

Prior research into basis function discovery

Previous research has explored basis function generation in great detail. One of the earliest works described a polynomial basis value function (Schweitzer and Seidmann [1985]). Tsitsiklis and Van Roy [1996] have more recently popularized *feature*-based linear methods, wherein a feature is defined as a function of state space S, projected onto a set of feature values. Some researchers have focused on specific feature types such as Gaussian linear models (Roweis and Ghahramani [1999]), whereas others have focused on how state values direct basis learning (Poupart and Boutilier [2003]), on state space topology (Johns and Mahadevan [2007]; Mahadevan [2005]), and on temporal relationship among variables (Ghahramani [1998]).

4.2 New transition matrix factoring basis discovery method

One way to improve value function approximation accuracy within a linear architecture is to:

- partition the state-space,
- discovering better basis functions within each partition and
- learning a specialized function for each partition.

Basis function discovery serves the same purpose as humans learning a simplified representation of their environment through salient feature combinations. State-space partitioning allows us to build simpler sub-functions instead of a complicated holistic function. Similarly, state-space partitioning enables replacing a monolithic strategy with situation dependent strategies. In the next subsection I discuss the rational for a new discovery mechanism to accomplish both tasks.

Prediction Directed State-space Partitioning and Basis Function Discovery

Various methods have been used to construct expressive basis functions. One important theme among methods is to select basis that relate to the prediction of interest. For example, Johns and Mahadevan [2007] exploits the fact that value functions are related to the geometry of the state-space as captured by the graph Laplacian, a relationship that can be approximately derived from the Bellman equation. Other methods seek basis correlated to the Bellman error. The commonality of these ideas is that they all have some notion of supervision. In other words, they are not merely looking for prominent variable combinations, rather they are looking for variable combinations that help predict.

Transition Matrix Factoring

Likewise, I look for basis that help predict state variable transitions. Specifically, I factor the state variable transition matrix. The state variable transition matrix is constructed from state sequences. Factoring this matrix creates a low-rank state transition model, which can be used for state clustering and function approximation. Furthermore, if we constrain the process to non-negative values, then the components can be interpreted as probabilities. Thus, each component represents a different part of the transition model. Similarly, each component represents a set of variable combinations. The method simultaneously discovers a good representation and learns transition probabilities.

Non-negative Matrix Factorization

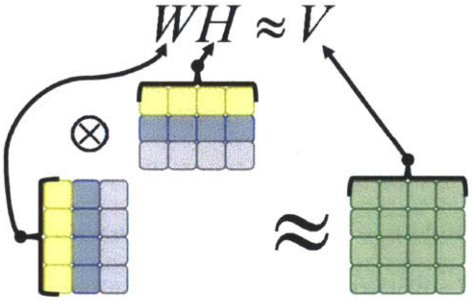


Figure 4-3: Three component matrix factorization: NMF decomposes a non-negative multivariate data matrix V such that $V \approx WH$ as shown.

Non-negative Matrix Factorization (NMF) was presented in Lee and Seung [1999] as an unsupervised way to elicit a low-order representation of facial image data. NMF basis can be interpreted simultaneously as latent attributes, variable combinations, conditional probability models or simply clusters. In short, NMF produces mixture models. Due to the non-negativity constraint, each component can be interpreted as a probability model.

Approximate Non-negative Matrix Factoring NMF decomposes a non-negative multivariate data matrix V such that $V \approx WH$ as shown in Figure 4-3. Specifically, Lee and Seung [2000] indicates that NMF can compress matrix V, comprised of m, n-dimensional data samples, into an $n \times r$ matrix W and an $r \times m$ matrix H, where $n < m$. In general, the r basis found via NMF represent latent variables, which constitute a low-order data representation of the data.

Optimality Function The approximate factorization $V \approx WH$ seeks to Minimize $D(V||WH)$ subject to $W, H \geq 0$, where $D(A||B) = \sum_{ij}(A_{ij}log\frac{A_{ij}}{B_{ij}} - A_{ij} + B_{ij})$. The optimization function is lower bounded by zero, and is minimal if $A = B$. The function reduces to the Kullback-Leibler divergence when $\sum_{ij} A_{ij} = \sum_{ij} B_{ij} = 1$. In that case, A and B can be thought of as probability distributions.

Gradient Descent Algorithm NMF iteratively reduces $D(V||WH)$ using update rules:

$$H_{au} \leftarrow H_{au} \frac{\sum_i W_{ia} Viu/(WH)_{iu}}{\sum_k W_{ka}}$$
$$W_{ia} \leftarrow W_{ia} \frac{\sum_u H_{au} Viu/(WH)_{iu}}{\sum_v H_{av}}$$

Uncertainty Limited NMF

NMF can be made to observe the simplicity principle by starting with a simple model that generalizes to new data, then cautiously increasing model complexity via feature combinations as the data permits. A model is made more complex by allowing additional mixture components. Choosing the right number of mixture components is equivalent to the ubiquitous issue of choosing the right number of clusters. I solve this problem by considering the likelihood of the posterior state estimates.

Specifically, NMF produces an ensemble of transition models. The posterior estimate is a multinomial distribution. Each posterior is supported by prior data. The model becomes more precise as the number of ensemble components increases because each component becomes more specialized. The model likelihood tends to increase as a result. However, the amount of data supporting each component decreases. The reduced support causes the overall model likelihood to decrease because the within component uncertainty increases. The correct trade-off is captured by the likelihood of the data given each model, computed by integrating over the model prior. In this way, the proper number of components can be determined.

Value Function Distribution for Uncertainty Based Restricted Search

Given the accuracy limitations of function approximation, it is helpful to augment the policy via simulation-based reasoning. Humans develop their own policies, which tell them how to behave in a given situation. However, when people are having trouble deciding between two actions, they may reason forward in time about the consequences of each action. A person typically has limited time to make a decision. Therefore, he or she must use the time judiciously to resolve the uncertainty.

Because of this need, our function approximation technique uses a Gaussian Process smoother to statistically combine the basis functions into a state-action value prediction. The Gaussian Process produces a value distribution over the mean value of the action outcome rather than a simple mean estimate.

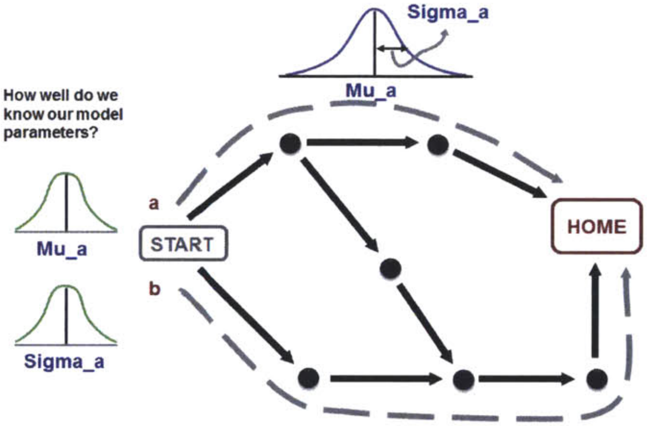


Figure 4-4: Suppose we are estimating how good it is to take path A. Our dynamics model provides a mean and variance about that number (shown in blue). We only care about the mean. We are risk neutral, so we use the mean to make decisions. (Sometimes you are not risk neutral, such as when you are going to the airport, and you may take a taxi to make sure you are not late.) But we don't exactly know what the mean is. But we have learned distributions over those model parameters, shown in green. We have a distribution over model parameter μ_a and σ_a. We only care about σ_a, but we do want to consider the uncertainty over that parameter when making decisions. If we are positive about it, than our decision is easy. If we are unsure about it, then we don't know if we are making the right choice.

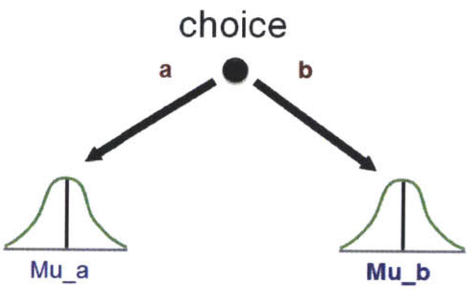


Figure 4-5: This picture now shows that we have the choice of path ‘a’ or path ‘b’, based upon the mean cost of each path, where that mean cost is not precisely known. We would pose the question: “What is the probability that path ‘a’ is a better option than path ‘b’, based on how well we know the expected outcome of a?” As a point of clarification, even if travel time (cost) was deterministic, there would still be some uncertainty about what that deterministic cost is.

4.3 Representational Uncertainty Defined

Representational uncertainty is caused by the inability of a functional form to accurately map a set of inputs to a set of outputs. This inability is due to an insufficient representation. The effect of this mapping inaccuracy is uncertainty in the posterior value estimate. The sum of squared residuals is a common measure of representational uncertainty.

$$\epsilon^2 \leftarrow \sum_i \|f(X_i) - y_i\|^2 \quad \text{where} \quad y_i \approx f(X_i) \tag{4.11}$$

If we consider the linear case with the following functional form

$$y_i \approx f(X_i) = \sum_j X_{ij}\beta_j \tag{4.12}$$

then the residual is defined as

$$\epsilon^2 \leftarrow \sum_i \left\| \sum_j X_{ij}\beta_j - y_i \right\|^2 \tag{4.13}$$

thus implying that dependent variable y_i can be modeled as

$$y_i = \sum_j X_{ij}\beta_j + \epsilon \tag{4.14}$$

where ϵ is a random variable. We can view the inputs to above solution in two different ways, as statistical units or as an overdetermined system of linear equations.

We are given a set of n samples from a system, where each sample X_i is a feature vector consisting of f features and 1 output y_i. X represents the entire sample set and X_{ij} denotes feature j of sample i. We wish to learn the unknown parameters β with which to predict dependent variable y given independent variable X. Assuming that the estimation errors are uncorrelated with each other or with X_i and have equal variance, the lowest residual unbiased linear estimator of is:

$$\widehat{\beta} \Leftarrow (X^{\mathsf{T}}X)^{-1}X^{\mathsf{T}}y \tag{4.15}$$

Statistical view

In the statistical view, ϵ is typically assumed to be a Gaussian random vector of mean zero and variance σ^2.

$$\epsilon \equiv \mathcal{N}(0, \sigma^2) \tag{4.16}$$

This of course presupposes the above "best linear unbiased estimator" assumptions. Regardless of the distribution type, the statistical view is that there are inherent system dynamics that are stochastic. Therefore, we could not possibly model these stochastic dynamics with a more sophisticated model. The only option is to model it with a representative distribution.

Overdetermined system view

But seriously, that disturbance term would account for noise in the process but it also accommodates the inadequacy of the linear model to capture the system dynamics. Given the following overdetermined system of f unknown coefficients $\beta_1 \ldots \beta_f$ and n linear equations where $n > f$:

$$\sum_j X_{ij}\beta_j = y_i \ \text{ written as } \ X\beta = y \ \text{ where} \tag{4.17}$$

$$X = \begin{pmatrix} X_{11} & X_{12} & \dots & X_{1f} \\ X_{21} & X_{22} & \dots & X_{2f} \\ X_{31} & X_{32} & \dots & X_{3f} \\ \vdots & \vdots & \ddots & \vdots \\ X_{n1} & X_{n2} & \dots & X_{nf} \end{pmatrix}, \quad \beta = \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_f \end{pmatrix}, \quad y = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{pmatrix}, \quad \epsilon = \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \vdots \\ \epsilon_n \end{pmatrix}$$

The system usually has no solution, so the goal is to find coefficients β which fit the equations best. In this view we are fitting the output from a non-linear system, onto a linear model. The system cannot be perfectly described using a linear model, resulting in some error. The error has nothing to do with system stochastics. If we assume the system to be deterministic, the same error exists. Therefore, the error is due to an insufficient representation.

Homoscedasticity Functions

The above representation only allows for a homoscedastic errors. It assumes that the variance of the error is constant across observations as depicted in Figure 4-6. Put another way, the function does not have the capability of attributing a different level of uncertainty for different parts of the prediction space.

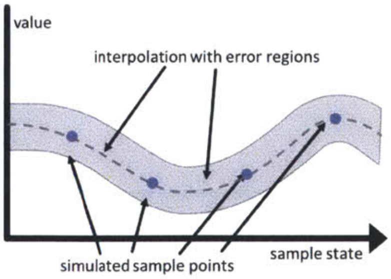


Figure 4-6: One dimensional homoscedastic function reflecting a constant variance across all predictions

Heteroscedasticity Functions and Autonomy

But sometimes the prediction variance is correlated to the input variables. In other words, the prediction uncertainty is different depending on what part of the input space you are working in as depicted in Figure 4-7. This relates to the problem of autonomy in the following way. Suppose we are using a function to predict the value of taking a given action. Our function conveys how certain we are about that value. That value certainty translates to how certain we are about the what to do. Clearly, your level of certainty and confidence varies based on the situation. However, if that value is estimated with a homoscedastic function, we would not be able to represent that difference.

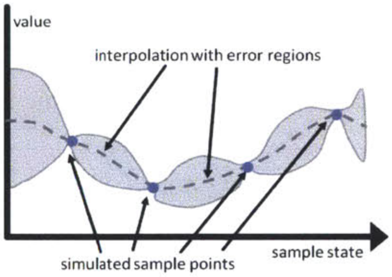


Figure 4-7: One dimensional heteroscedastic function showing tie-down points and variance for a given prediction

Next we will review some methods that allow heteroscedasticity.

Kernel Smoothing Equations

$$\hat{P}(x) = \frac{1}{nh} \sum_{i=1:n}^{n} K\left(\frac{x - X_i}{h}\right) \tag{4.18}$$

$$K(x) = \frac{1}{(2\pi)^{\frac{p}{2}}} \exp -\frac{1}{2} \sum_{i=1}^{p} (x_i^2) \tag{4.19}$$

The kernel trick

The kernel trick is a calculation that accomplishes the above kernel smoothing in a very elegant way. It also allows for various kinds of kernels. It is called the kernel trick because it allows you to create a function in a higher dimensional space without explicitly elaboration the higher dimensional basis or features. In other words it allows you to create a more expressive, more complicated mapping with a relatively simple calculation. That calculation does not involve transforming your basis into a more complicated basis. When using a Gaussian kernel, the function is theoretically infinite dimensional. Here we will go through an example using a polynomial kernel. The reader may refer to MacKay [2003] regarding the infinite dimensional case.

Non-parametric Least Squares Regression with a polynomial Kernel

Non-parametric least squares regression explicitly estimates future data values using weighted combinations of the training data-set. The method is fast and effective given certain conditions. Its accuracy is good but limited by the number of samples that can be handled by the regression method. It works as follows.

Suppose we have a set of state samples X and a set of target values y. X is an n by f matrix, where n is the number of state samples and f is the number of features (variables used to characterize the state). y is an n by 1 matrix where n is the number of state samples. We compute the function as follows:

$$\alpha = y^{\mathsf{T}} \left(\left(X X^{\mathsf{T}} + 1 \right)^{d} + \lambda I \right)^{-1} \tag{4.20}$$

where λI is a ridge penalty, and d is the kernel degree. We can then estimate a new target value target value $\hat{y}$ given a new feature vector x_{new}. x_{new} is a single feature vector of size $1 \times f$.

$$\hat{t} = \alpha \left(X x_{new}^{\mathsf{T}} + 1 \right)^{d} \tag{4.21}$$

where X is a set of training samples and x_{new} is an new sample.

Complexity Penalty Matrix and Kernel Degree

A ridge penalty, λ multiplied by the identity matrix I, is a common regularizer, which first and foremost ensures that the Gram matrix is invertible. Secondly, the ridge penalty can be increased to reduce estimation variance at the expense of increased bias to control the complexity of the function. For convenience we will call $\mathbb{R} = \lambda I$. In general any regularizing prior $\mathbb{R}$ can be used.

d is the polynomial degree of the kernel. Thus, XX^{T} is the Gram matrix and $\left(XX^{\mathsf{T}} + 1\right)^d$ is the result of applying a polynomial kernel to the Gram matrix. Essentially, this is a non-parametric way to create and use a set of polynomial basis vectors for function approximation.

One interesting aspect of this function approximation method is that if we

let $K = inv\left(\left(XX^{\mathsf{T}} + 1\right)^d + \mathbb{R}\right)$,

where $\mathbb{R}$ is a regularizing matrix. $k(X, X)$ performs the following on each element of the Gram matrix, k

$k(x, x) = (xx^{\mathsf{T}} + 1)^d$

Suppose $\mathbb{R} = 0$ and $d = 1$, then we have a linear kernel.

we can break up the equation

$$\alpha = y^{\mathsf{T}} \left(k(X, X) + \mathbb{R}\right)^{-1}$$

such that

$$\alpha = y^{\mathsf{T}} K$$

Thus, if we perform a value iteration backup (where our function approximator represents our value function) using the exact same state samples X as our original function, we can easily and quickly recreate the function approximation simply by replacing y with our updated one. In other words, the expensive part of the function (K) remains the same.

This is called the kernel trick because it allows you to work in the higher dimensional space without ever explicitly representing that space. This is an easy way to get a more expressive function.

Extracting the covariance

$$kK^{-1}k^{\mathsf{T}} \tag{4.22}$$

$$f(x) = \sum_{i=1}^{n} \alpha_i k(x, x_i) \quad \text{where} \quad n \in \mathbb{N}, x_i \in X \quad \text{and} \quad \alpha_i \in \mathbb{R} \tag{4.23}$$

Gaussian kernel

$$k(x, x') = \sigma_f^2 \exp -\frac{1}{2l^2} \sum_{i=1}^{p} (x_i - x_i')^2 \tag{4.24}$$

We then use cross validation to learn the best band width (l) for the kernel based estimator. This idea can be expanded into the Gaussian process smoother, to capture and express its own representational uncertainty within the function.

4.4 Function approximation is the source of uncertainty in ADP solutions

Our task is to find the policy (π) for selecting control action a in state s which maximizes the expected cumulative reward over the entire mission. Naturally, we can formulate the problem as a dynamic program, which maximizes the expected one-stage reward r plus the expected cumulative future reward V Bellmann [1957],

$$V(s) = \max_a E_{s'} \left[\mathcal{R}^a_{ss'} + V(s') \right]. \tag{4.25}$$

The dynamic programming equation is recursive. However, given the value function V, we could compute the optimal control action a. Therefore, knowing V is tantamount to knowing the optimal control policy. Our objective is, therefore, to find the optimal future reward function V.

4.4.1 Value Iteration

Our objective is to learn value function V. For a discrete state space, V is represented as a table of values (one record for each discrete state). The table of values can be initialized arbitrarily and improved iteratively. To iteratively improve the value function estimate, for each s, we find control action a that maximizes the expected $\mathcal{R}^a_{ss'}$ plus the expected value of next state $V(s')$.

$$V^{i+1}(s) \leftarrow \max\nolimits_a E_{s'} \left[\mathcal{R}^a_{ss'} + V^i(s') \right] \tag{4.26}$$

The action outcomes are stochastic; therefore, the expectation entails summing over the possible rewards and next states, times their corresponding probability.

$$V^{i+1}(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + V^i(s') \right) \tag{4.27}$$

We compute this summation for each control action. We then replace the current table entry $V(s)$ with the new maximum value corresponding to the best control action. The value iteration process is shown in Figure 5-11. The above process constitutes one iteration.

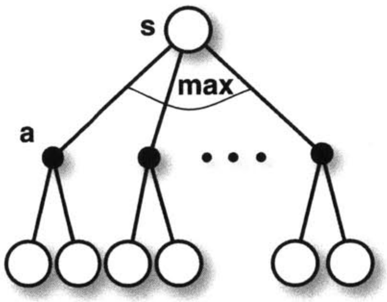


Figure 4-8: Value iteration backup diagram for discrete state-space: white circles represents states while black circles represent actions..

We repeat the process until $V(s)$ converges.

The main problem with this approach is that we must perform a value iteration backup for every single state, which would take far too long. Since our real world problems are often described by more than 100 state variables, then if we discretized each of these continuous variables by 10 we would end up with 10^{100} discrete states. This is far too many states to perform a backup over. Furthermore, we cannot store the huge table representation of V in memory.

We address these complexities by representing V as a parametric function. Next, we will discuss how V would be represented as a linear function. Then we will discuss the consequences. Specifically, a linear representation is too simple to perfectly represent V, consequently there will be a lot of uncertainty surrounding that estimate.

Linear Architecture: Formally, a linear architecture approximates $V(s)$ by first mapping the state s to feature vector $\Phi(s) \in \Re^k$ and by computing a linear combination of those features $\Phi(s)\beta$, where β is a function parameter vector. We compose $\Phi(s)$ by applying basis function $\phi()$ to the probability of occupancy of each grid cell, such that

$$\Phi(s) = \left[\begin{array}{ccc} \phi(s_1) & ... & \phi(s_n) \end{array}\right] \tag{4.28}$$

We also use an intercept that our description ignores for simplicity. We compute an improved value estimate $\hat{V}(s)$ for a finite set of states $S_i \in X$ via value iteration backups, such that

$$\hat{V}(s) = \max_a E_{s'}\left[\mathcal{R}^a_{ss'} + V_{approx}(s')\right], \tag{4.29}$$

where $V_{approx}(\cdot) = \Phi(\cdot)\beta$.

We then update our estimate of V_{approx} using regression, where

$$\Phi(X) = \begin{bmatrix} \Phi(S_1) \\ \vdots \\ \Phi(S_\nu) \end{bmatrix} \tag{4.30}$$

is a set of feature vectors for all state samples and

$$\hat{V}(X) = \begin{bmatrix} \hat{V}(S_1) \\ \vdots \\ \hat{V}(S_\nu) \end{bmatrix} \tag{4.31}$$

is the updated value estimate for all state samples.

We re-estimate the parameters β of our function as

$$\beta = \left(\Phi(X)^T \Phi(X) + \lambda I\right)^{-1} \Phi(X)^T \hat{V}(X) \tag{4.32}$$

where ridge penalty λ ensures invertability and can be increased to reduce estimation variance at the expense of increased bias. Our updated $V_{approx}(s) = \Phi(s)\beta$ now estimates our new target value $\hat{V}(s)$ in a least squares sense.

To summarize, we simulate a next state and plug that state into V_{approx}, our approximate value function. As the sensor measurements are stochastic, we must sum over the rewards of each possible future state.

4.4.2 Identifying the source of the uncertainty in MDP Solutions

In this section we will identify the source of uncertainty and outline how we will get rid of it or at least mitigate and deal with it.

Key point: The uncertainty in the posterior estimate stems from the the inability of

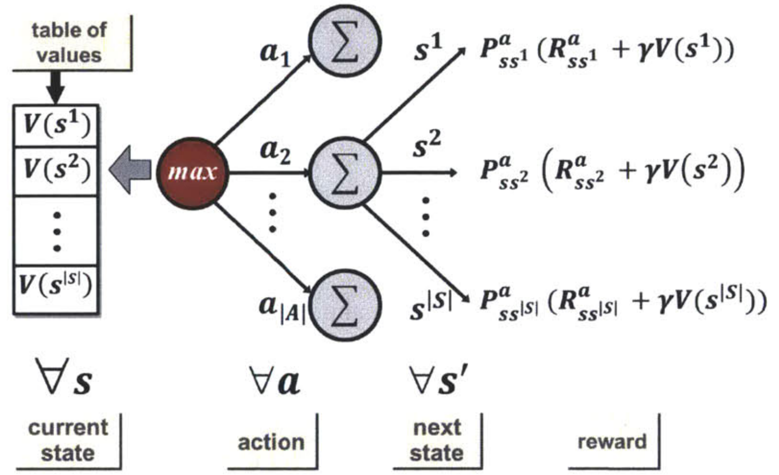


Figure 4-9: Value iteration algorithm: 1. Initialize J( ), 2. Loop (Backup all states) 3. Converges to optimal. Cannot Compute backup for all states or store value table. Function Approximation is needed. The cumulative future reward for an MDP can be reformulated as a dynamic programming problem where you minimize over the expected one-stage reward plus the expected future reward starting from the next state. The dynamic programming equation is recursive, however, if you know the optimal future reward function J then you can compute the control action u which maximizes the dynamic programming equation. Therefore, knowing the optimal future reward function J is tantamount to knowing the optimal control policy. Thus, our objective boils down to finding the optimal future reward function J. I assume you are familiar with the value iteration approach to dynamic programming Sutton and Barto [1998]. I will start the discussion by outlining the basic component of the algorithm, the Bellman backup. I will then identify the computational complexities, and how we address them. Our objective is to learn J. We start by initializing J arbitrarily and we represent J as a table of values. We then improve upon our estimate of J by performing a Bellman backup. Specifically, for a particular state x, we find control action u which maximizes the current reward g plus the future reward J (using our current approximation of J. The sensor returns are stochastic, therefore we need to sum over the rewards times their probability of occurring. We compute this summation for each control action. We then replace J(x) with the maximum value. The above process constitute one iteration. We repeat the process until J(x) converges. The main problem with this approach is that you must perform a Bellman backup for every single state, which would take way too long even for discrete beliefs. Furthermore, we cannot store the huge table representation of J in memory. We address this complexity by representing J as a parametric function.

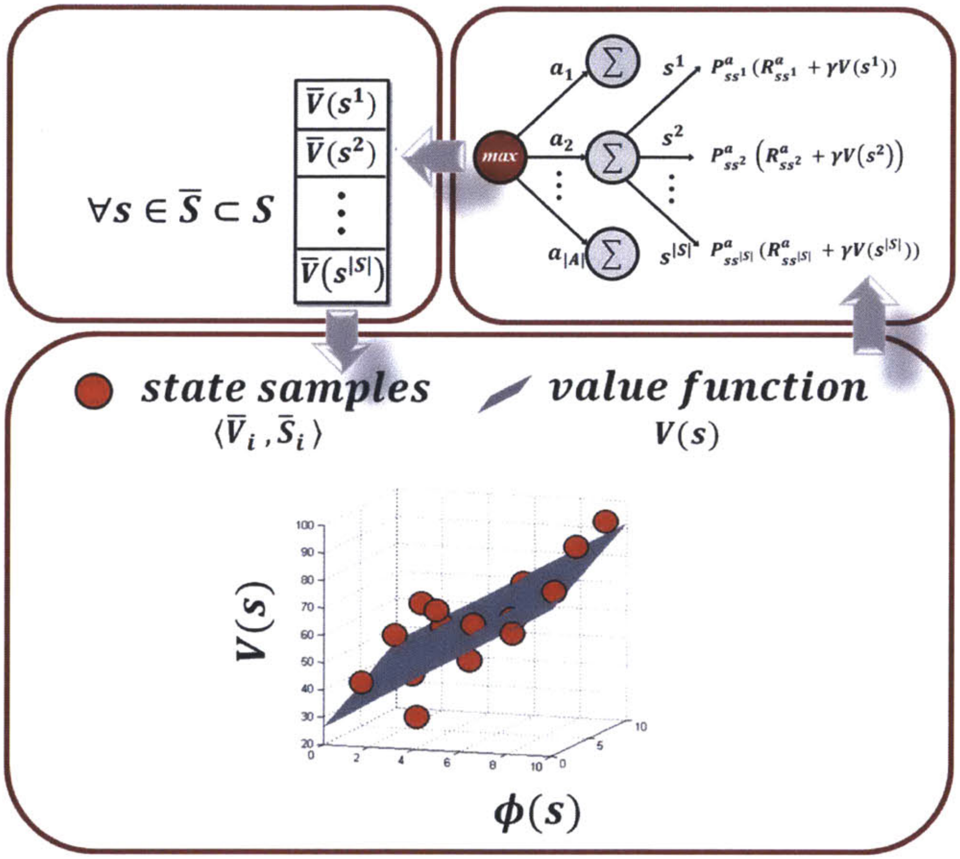

$\bar{V}(s^1)$
$\bar{V}(s^2)$
$\vdots$
$\bar{V}(s^{|S|})$
$\forall s \in \bar{S} \subset S$
max
a_1
a_2
$a_{|A|}$
Σ
s^1
s^2
$s^{|S|}$
$P^a_{ss^1}(R^a_{ss^1} + \gamma V(s^1))$
$P^a_{ss^2}\left(R^a_{ss^2} + \gamma V(s^2)\right)$
$P^a_{ss^{|S|}}(R^a_{ss^{|S|}} + \gamma V(s^{|S|}))$
state samples
$\langle \bar{V}_i, \bar{S}_i \rangle$
value function
$V(s)$
$V(s)$
100
90
80
70
60
50
40
30
20
0
2
4
6
8
10
0
5
10
$\phi(s)$

Figure 4-10: Represent J( ) as a function. Select Features. Perform regression. Perform approximate value iteration. Avoids J( ) table representation and avoids backing up all states. Which is why our approach is to approximate J as a parametric function, and learn an approximation of J using dynamic programming. This slide depicts how function approximation works. Again, our objective is to learn J, and we start by initializing J arbitrarily. In the table lookup approach, we represent J as a table of values. However, in the function approximation approach, we extract features of our state, then approximate J using linear regression over those features. We then improve upon our estimate of J by performing a Bellman backup. In the table lookup approach, we performed a Bellman backup for every single state. However, in the function approximation approach, we sample the states, and perform a backup just for those sampled states. By doing so, we avoid the table lookup representation of J and we avoid backing up all states. To review, in the table lookup approach, when performing a Bellman backup, we simulate a future state. We then lookup the value of that state (J) in a table. However, in the function approximation approach, we simulate a next state. We then plug that state into $\widehat{V}$, our approximate future reward function.

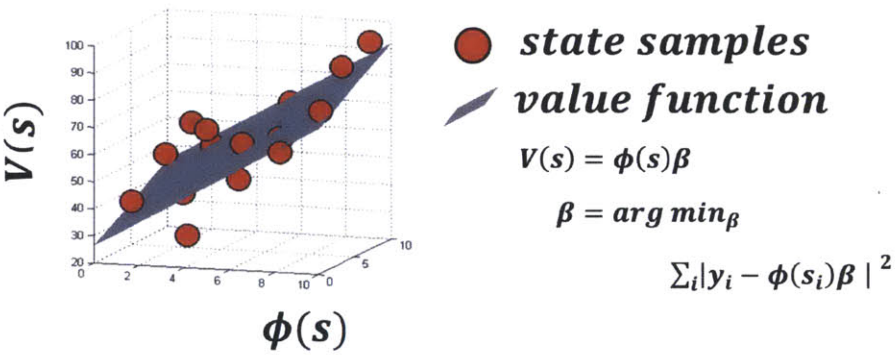


Figure 4-11: Want compact and accurate state-action value function representation. Partition the state space. Find basis functions specialized for each partition. Learn a function for each partition.

the function to express the real-world phenomena that it is modeling. The uncertainty is a measurement of the residuals (estimation errors). However, the increased uncertainty is *not* due to the dynamical stochastics. We know this because the residuals are computed from the expected system outcome. Put another way, the function is specifically estimating the expected system outcome and it is compared to the actual expected system outcome. Thus, the uncertainty stems from the estimation architecture's inability to estimate an accurate expected outcome. If we control for the number of training samples, what remains is the representational capability.

The agents use approximate dynamic programming to create a policy for acting in the world. A policy is a mapping from states to actions, which tells the robot what to do in any situation. Computing a value function can be computationally expensive, so we compute this offline approximately through value iteration before the mission starts. We do so by simulating the mission, saving simulation snapshots as data points. We then generate an approximation architecture on each iteration by data point regression, accounting for representational uncertainty.

A suboptimal offline policy for a realistically large MDP can be found via approximate value iteration

The above formulation and algorithms assume a discrete state set S. If the state space is discrete, V and Q can be represented as a table of values, one record for each discrete state; the table of values is initialized arbitrarily and improved iteratively. The problem with this approach is that many real world state spaces are continuous and an acceptable discrete representation is intractably large, exceeding memory capacity. Additionally, we must perform a time-consuming value iteration backup for each state.

Algorithm 9 Approximate state-action value iteration algorithm where $\widehat{Q}$ stands for an approximation architecture representation of the state-action value function, $\bar{Q}$ stands for a lookup table of state action values over a subset $\bar{S}$ of the full statespace S.

```
ApproxStateActionValueIteration($\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$) $\Leftarrow$ {
for all $s \in \bar{S} \subset S$ do
  for all $a \in A$ do
    Initialize ($\bar{Q}(s,a)$)
  end for
end for
InitializeApproximationArchitecture($\bar{Q}_t, \widehat{Q}_t$)
$t \Leftarrow 0$
repeat
  $t \Leftarrow t + 1$
  for all $s \in \bar{S} \subset S$ do
    for all $a \in A$ do
      $\bar{Q}(s,a) \Leftarrow \sum_{s' \in S} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma \max_a \widehat{Q}(s',a)]$
    end for
  end for
  $\widehat{Q} \Leftarrow$ UpdateApproximationArchitecture($\bar{Q}, \widehat{Q}$)
  for all $s \in \bar{S}$ do
    $\pi(s) \Leftarrow \arg\max_a \widehat{Q}(s,a)$
    $\bar{V}_t(s) \Leftarrow \bar{Q}(s, \pi(s))$
  end for
until $\max_{s \in \bar{S}} |\bar{V}_t(s) - \bar{V}_{t-1}(s)| < \epsilon$ , $\forall s \in \bar{S}$
return $\widehat{Q}_t$
}
```

We address these complexities by representing Q using an estimation architecture ($\widehat{Q}$). The estimation architecture stores the state action-value function in a compressed form, such as a linear function over problem variables rather than an explicit combination of each possible variable assignment. When used in approximate value iteration (Bertsekas and Tsitsiklis [1996a]), the estimation architecture alleviates the storage problem and shares information across state variables, decreasing learning time. The estimation architecture, therefore, solves both problems inherent to exact value iteration.

Algorithm 9 outlines the approximate value iteration algorithm where $\widehat{Q}$ is the approximate value function represented by an estimation architecture. Algorithm 9 takes as input the MDP tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where $|\mathcal{S}|$ is very large, possibly infinite. Algorithm 9 begins by randomly initializing state-action value table $\bar{Q}(s, a)$ over state subset $\bar{S}$. The approximation architecture $\widehat{Q}$ is then initialized based on the $\bar{Q}$ table. We then perform a Bellman backup over state subset $\bar{S}$, using $\widehat{Q}$ for future state-action value estimates. The newly computed state-action values are stored in table $\bar{Q}$ and the policy, approximation architecture and state value table are then updated; this process repeats until a convergence threshold is met. Algorithm 9 returns approximation architecture $\widehat{Q}_t$.

Quantifying the uncertainty of a policy

The preceding approximation saves time and space but may not adequately *represent* the state-action value function, which results in a suboptimal policy. Uncertainty about the respective quantities results in uncertainty about the policy decisions. In Chapter 3 we presented a method for dealing with this decision uncertainty. Given an available model $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, the aforementioned policy could be augmented with an online search algorithm. This concept requires finding a good value function representation. Moreover, we need to represent a distribution over values, capturing our uncertainty about the specific expected outcome. The remainder of this chapter walks through how we find a good set of basis functions for our representation and how we train it to output a predictive distribution over the average or expected system outcome.

5 Applications

Chapter 2 introduced a 3-level active sensing (See Lermusiaux [2007], Mihaylova *et al.* [2003], Mihaylova *et al.* [2002], Ahmad and Yu [2013], Murphy [1999] and Krause and Guestrin [2009]) hierarchy motivated by planetary exploration. In this chapter we cover in detail three applications (Figure 5-1), which populate the hierarchy discussed in Chapter 2. These include a synoptic, satellite-based, application for ocean sensing and algae bloom mapping; a mesoscale, multi-vehicle airplane surveying application; and a microscale sensing application in which an unmanned aerial vehicle (UAV) maps out relevant traversable roads to provide ground vehicle routing support for disaster relief.

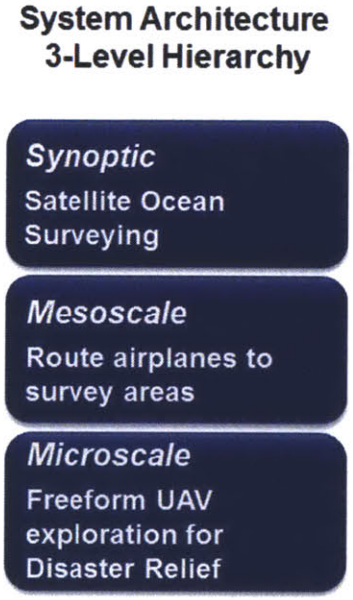

System Architecture
3-Level Hierarchy
Synoptic
Satellite Ocean Surveying
Mesoscale
Route airplanes to survey areas
Microscale
Freeform UAV exploration for Disaster Relief

Figure 5-1: Outline of how the three applications discussed in this chapter populate the 3-level hierarchy.

In the microscale part of our architecture, we abstract away the sensor, which detects traversable roads. This thesis focuses on sensing optimization. However, a key component of the system is the sensor-processing package. Our algorithms are designed with an eye toward genericity. To be generic, we need a clean interface between the optimization algorithm and the real world. In Appendix F, we develop a sensor processing package for detecting roads for disaster relief routing support. This example shows how the sensor processing package can be abstracted away such that the sensor planning algorithm seems to directly interface with measurements of the phenomena of interest, in this case the presence of traversable roads. The synoptic and microscale technologies are fleshed out for two earthly applications. The mesoscale technology is fleshed out on a multi-vehicle airplane surveying mission directly applicable to planetary exploration. When combined with an appropriate abstract sensor processing package, all three applications map to the planetary exploration application.

Our previously described 3-level hierarchy addresses the narrow focus issue. Combined with adaptive algorithms and a planetary architecture, it provides a framework for planetary exploration.

5.1 Synoptic Application (Ocean Sensing)

Ocean Sensing We can demonstrate the synoptic level of the hierarchy with an ocean survey application that scans a wide area for algae blooms, identifying areas of heightened scientific interest. This application also illustrates the mesoscale and microscale levels of the hierarchy, as the satelliate identifies other places of interest for future study by closer-in sensor platforms, such as unmanned aerial vehicles (UAVs) and autonomous underwater vehicles (AUVs).

Autonomous agents are already very good at collecting a lot of information; our research is intended to help them collect the right information. The optimal adaptive exploration problem is to sequentially select observations that minimize the uncertainty of a state estimate. A common example of this problem is occupancy map exploration, in which a sensor seeks to minimize the uncertainty over which grid cells are occupied, using a finite set of sequential observations. The tools we have at our disposal include a sensing platform, a generic signal processor, and a pattern recognition engine to detect activities of interest.

A concrete example is autonomous ocean sampling (Monterey Bay Aquarium Research Institute [2006]), where networked sensing platforms detect activity of interest over a geographic area (Figure 5-2; Monterey Bay Aquarium Research Institute [2006]). For example, a satellite could collect ocean surface color data, which provides indications of algae bloom formation. The data could be automatically analyzed and aggregated into an algae bloom formation activity map.

At this level, we generate a synoptic activity map. Our probabilistic activity map provides a holistic approach to characterizing ocean observations. The map can subsequently be used to direct the attention of other platforms (e.g. autonomous underwater vehicles (AUVs)) for focused exploration.

Because the method of searching the area affects the quality of the map, the problem is to generate the most informative search plan. Due to the overall problem complexity, researchers settle for sub-optimal adaptive greedy strategies for choosing sensing locations. In other words, rather than considering the combined consequences of the current and potential future actions, they evaluate only the first action. While greedy strategies perform reasonably well (Guestrin *et al.* [2005]) with a worst case optimality of $(1 - \frac{1}{e})$ (Krause and Guestrin [2005]), for many applications expensive resources are deployed based on the information collected, amplifying the impact of sub-optimality. Therefore, a better solution is very desirable. In this section, we show how domain specific problem structure can be exploited, and a better sensing strategy can be efficiently learned.

Figure 5-2: The autonomous ocean sampling network (AOSN) in Monterey Bay integrates a variety of modern platforms (e.g. satellites and AUVs) to produce a comprehensive regional view of the ocean.

Method We review the use of open-loop feedback control (Bertsekas [2005b]) and discrete state policy learning. Although both methods would improve optimality, they are intractable. Specifically, they solve problems that are infinitesimal in size relative to relevant real-world problems.

Thus, our approach is to model the problem as an "information state" partially observable Markov decision process (POMDP). In a POMDP, the underlying state is partially observable and decisions must be made using an external distribution over possible states. However, in our version, that distribution (or belief) is the state (or part of the state) and therefore the reward can be a function of the belief uncertainty.

Our key contribution is the closed form backup of the state estimate uncertainty, enabling a non-myopic approximate policy to be computed via least-squares value iteration (LSVI). Specifically, we compute an entire value iteration backup for all state samples and control actions as a single compact matrix multiplication. We then use linear function approximation to compute an approximate value iteration in closed form. We can then execute the policy using a rollout algorithm to select the control action, which maximizes the expected outcome with respect to the base policy.

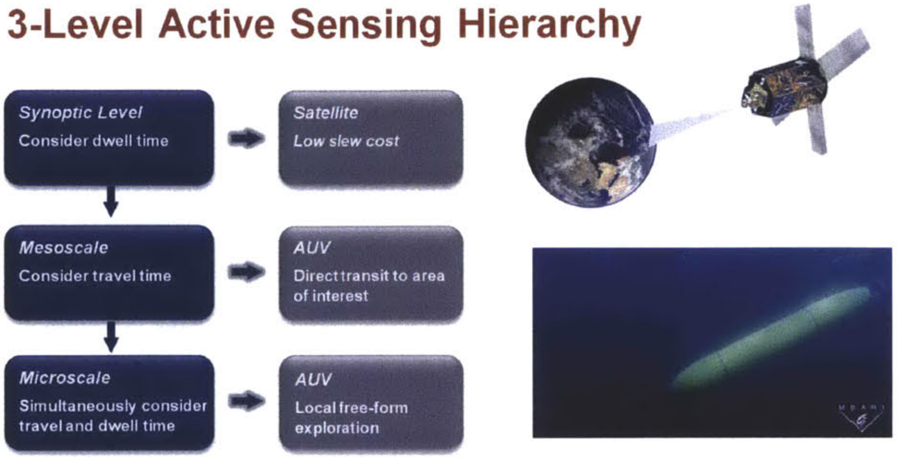
Organization In the following section, we describe our formulation of the problem and characterize the optimal solution. We next review possible approaches and why they are intractable, We then describe our learning algorithm and performance results. Specifically, we show how the expected reward can be computed in closed form, and demonstrate how this enables efficient value function learning. We conclude with a demonstration of our approach on a set of example activity detection problems of up to 1,089 grid cells, and show that we are able to find policies efficiently that are within 2% of the optimal solution.

5.1.1 The 3-level Hierarchical Framework Applied to an Ocean-Sensing Application

Both off-world exploration problems and terrestrial sensing problems cover large geographic areas and require multiple sensor types. The complexities of the real world require a complex architecture for handling real problems. Described here is a blueprint for designing real world active sensing systems. The hierarchical framework applies to these problems and many more.

3-level Hierarchy

In the 3-level hierarchy as presented in Figure 5-3, sensor coverage at the synoptic level is provided by a satellite. Coverage at the mesoscale and microscale is provided by an autonomous underwater vehicle (AUV). The synoptic sensing (satellite) optimization algorithm concentrates on dwell time and the value of the collected information. The mesoscale system is optimized with respect to (the AUVs') travel time. Travel and dwell time are simultaneously optimized at the microscale, to optimize the free-form (AUV) exploration. The satellite coarsely surveys a large swath of the Earth, then relays its information to a



3-Level Active Sensing Hierarchy

Synoptic Level
Consider dwell time

Satellite
Low slew cost

Mesoscale
Consider travel time

AUV
Direct transit to area of interest

Microscale
Simultaneously consider travel and dwell time

AUV
Local free-form exploration

Figure 5-3: The 3-level hierarchy shown above includes a synoptic level (covering a wide area), a mesoscale level (covering a medium sized area) and a microscale level (covering a small local area).

group of AUVs. The AUVs deploy to the areas that look most interesting and survey them.

SEAWIFS SATELLITE

The SeaWiFS (Sea-viewing Wide Field-of-view Sensor) sensor flown on GeoEye's OrbView-2 "SeaStar" satellite, shown in Figure 5-4, operated from September 18th, 1997 until December 11th, 2010 (NASA [2011b]). The sensor resolution is 1.1 km (LAC), 4.5 km (GAC). Designed to monitor ocean characteristics such as chlorophyll-a concentration. The sensor recorded information in eight optical bands at a resolution of $\tilde{1}$km.

The optimization at each level is different. A satellite can quickly change where it is looking, within the sensor's footprint. Therefore, the sensing optimization algorithm concentrates on dwell time and the value of the collected information. The dwell time refers to how long the satellite looks at a particular location. The satellite is optimized to look longer at areas of high interest and high uncertainty. The SeaWiFS sensor is shown in Figure 5-4.

At the mesoscale level, the AUVs transit directly to the locations of interest. Therefore, the mesoscale system is optimized with respect to travel time. Specifically, a set of target areas are selected using the satellite data. The target areas may have time constraints. The target areas are divided up between AUVs, where a desired assignment minimizes travel. Each AUV directly transits to its assigned area, according to the optimized vehicle routing plan.

The microscale optimizer handles the local sensing tasks. At the microscale level, the AUVs are allowed to explore the local area freely, collecting data as they move. Therefore, travel and dwell time are simultaneously optimized at the microscale level.

Figure 5-4: Shown above is GeoEye's OrbView-2 "SeaStar" satellite carrying the SeaWiFS ocean color sensor.

5.1.2 The Synoptic Problem (coarse global satellite surveying)

Active sensing example

While future scientists will be interested in mapping extra-terrestrial surfaces and oceans, modern scientists care about detecting and monitoring algae blooms and other ocean phenomena. To accomplish these objectives, Monterey Bay Aquarium Research Institute is building an autonomous ocean sampling network shown in Figure 5-5 (see Monterey Bay Aquarium Research Institute [2006]). The network enables collaboration between different kinds of sensors. It builds upon the work discussed in Chapter 2 (see Figure 2-3, which used a non-adaptive bathymetric survey, adding an adaptable mission planning layer that allows the underwater sensors to receive and respond to information from above-water sensors. For example, an algorithm could use the network to direct autonomous submarines to explore algae blooms detected by satellites. To create such an algorithm, we divide the problem into three parts and develop a single algorithm that works as the core for all three parts. In this way, we develop a good framework for controlling and directing mobile sensors for oceanographic missions while also developing an algorithm that is useful for other autonomy problems.

Ocean Sensing Our ocean sensing application, described in Section 5.1, is introduced below. At the synoptic level, this research focused on surveying the ocean for algae blooms, in order to identify places of heightened scientific interest. Once these sites have been identified, they are ready for future study at the mesoscale or microscale level, using a close-up sensor platform such as a UAV or AUV. This approach is directly applicable to complex planetary exploration problems requiring coordination between sensors at all three levels of the hierarchy. In addition, we tested an aspect of decision uncertainty minimization (DUM), which is fully defined in Chapter 3, on this problem. Specifically, we developed a fast adaptive sensing algorithm, then augmented the results via online rollout based reevaluation. The results show clear improvement of reevaluation over approximate dynamic programming by itself.

3-Level Problem

Figure 5-6 shows the three levels of an ocean mapping problem. The synoptic sensor data is provided by a satellite shown in the figure. That data is used to identify and select the best local areas to survey. The best locations are those with the highest scientific value; for example, the locations are likely to contain an algae bloom. Figure 5-6 likewise depicts routing the AUVs to the best locations. As depicted, each location represents a local mapping problem. In the local mapping level, each location is its own information gathering problem, with the goal of estimating key environment variables in the area.

Synoptic problem model

At the synoptic level, our task is to map the area and sequentially decide where to sense next. We model the environment as an occupancy grid, as shown in Figure 5-7. The

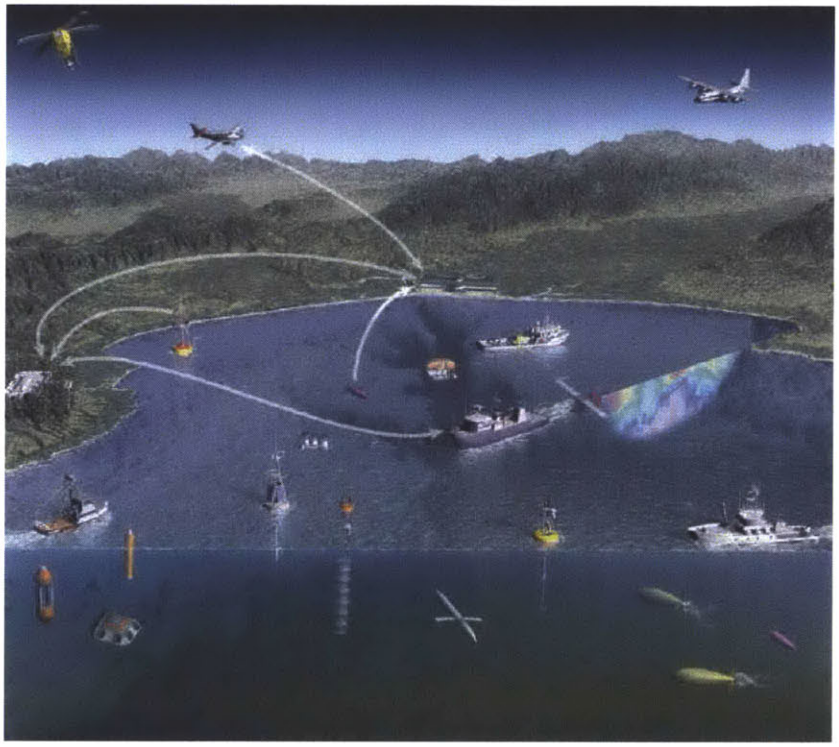


Figure 5-5: The autonomous ocean sampling network (AOSN) in Monterey Bay integrates a variety of modern platforms (e.g. satellites and AUVs) to produce a comprehensive regional view of the ocean. The network enables collaboration between different kinds of sensors. For example, an algorithm could use the network to direct autonomous submarines to explore algae blooms detected by satellites.

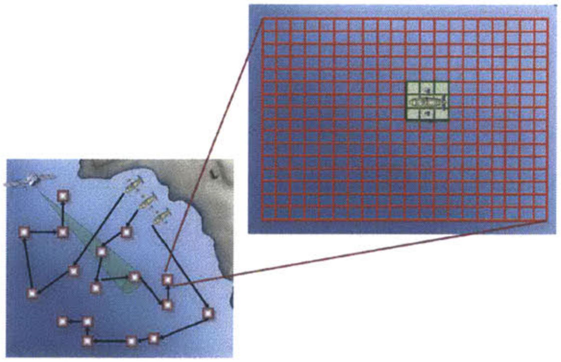


Figure 5-6: Satellite-AUV collaboration can be broken into two levels. At the mission level, we route the AUVs to the mapping task locations. In the local level, each location is its own information gathering task with the goal of estimating key environment variables in the area. Both problems can be solved with an uncertainty guided planning algorithm.

stochastic sensor measurements cause uncertainty regarding activity map state. Therefore, we infer the probability of activity in a given cell using Bayes filtering, i.e.,

$$p_i = P(i|w_i) = \alpha P(w_i|i)P(i) \tag{5.1}$$

where p_i represents the posterior probability that a grid cell is occupied, w_i is the sensor measurement for cell i, $P(w_i|i)$ is the sensor model and $P(i)$ is the prior probability of activity at cell i. We refer to the probabilities of the entire grid as a "belief map." The exploration process of building an accurate activity map works as follows. We take a sensing action, which returns a measurement for each of the 3×3 observed grid cells. For each sensed cell, we then update the belief map s_i using Equation 5.1. We then use the updated belief map to select the next most informative sensing action.

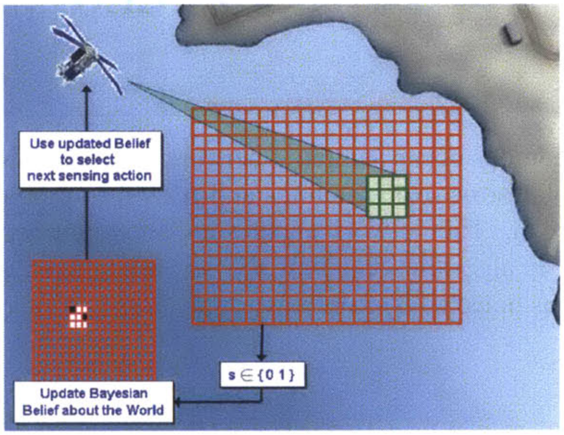


Figure 5-7: Occupancy grid and Bayesian belief representation of our satellite sensing problem.

The satellite can slew anywhere in its footprint with low cost. Therefore, there is no constraint as to where to sense next. At this level, the application is optimized purely with respect to the value of the information it collects.

Future applications of this framework include wide-scale surveys of planets and moons. Because the system is more efficient than non-adaptable approaches to autonomous or semi-autonomous surveys, it will allow future scientists to conduct remote sensing missions in a way that saves fuel and builds up detailed maps of the most interesting parts of the target area.

5.1.3 Problem Model

Our task is to map the activity in an area and sequentially decide where to sense next. We model the environment as an occupancy grid, as shown in Figure 5-8. An occupancy grid divides the area into discrete locations. The activity is assumed to be located at specific "occupied" grid cells. Without loss of generality, we assume that the activity does not change location, and the sensor can be steered from one aimpoint to any other aimpoint (i.e., the sensor is not constrained to follow a contiguous path).

We model the sensor as a 3×3 grid. When the sensor is aimed at a particular grid cell i, it observes that grid cell and the eight surrounding cells. Modeling the sensor in this way allows two actions to partially overlap. Therefore, the sensor can plan two actions that deliberately overlap in order to collect more information at the overlapping grid cell. Each of the nine measurements created by a single sensing action is modeled as an independent Bernoulli distribution, with some probability of detecting the activity of interest and some probability of false alarm.

The stochastic sensor measurements cause uncertainty regarding activity map state. Therefore, we infer the probability of activity in a given cell using Bayes filtering, i.e.,

$$p_i = P(i|w_i) = \alpha P(w_i|i)P(i) \tag{5.2}$$

where p_i represents the posterior probability that a grid cell is occupied, w_i is the sensor measurement for cell i, $P(w_i|i)$ is the sensor model and $P(i)$ is the prior probability of activity at cell i. We refer to the probabilities of the entire grid as a "belief map."

The exploration process of building an accurate activity map works as follows. We take a sensing action, which returns a measurement for each of the 3×3 observed grid cells. For each sensed cell, we update the belief map s_i using Algorithm 5.2, then use the updated belief map to select the next most informative sensing action.

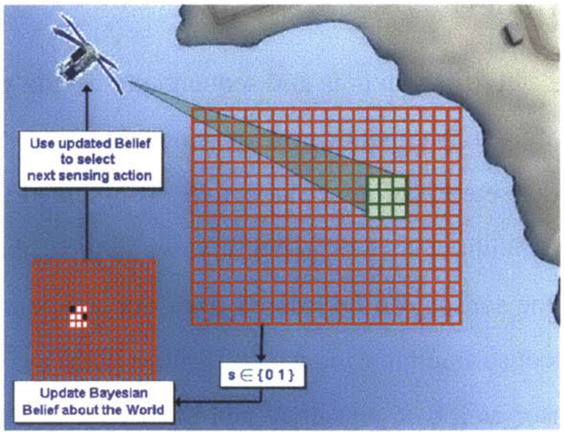

Use updated Belief to select next sensing action
s ∈ {0 1}
Update Bayesian Belief about the World

Figure 5-8: Occupancy grid and Bayesian belief representation of our satellite sensing problem.

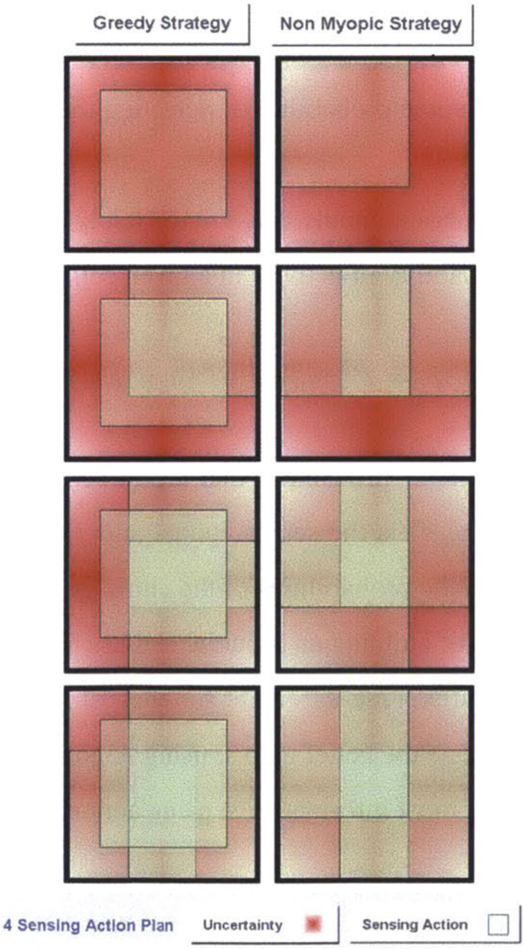


Figure 5-9: Example comparison of greedy and non-myopic sensing action selection strategies.

5.1.4 Motivation For Our Approach

Our objective is to find the most informative sensing plan describing the sequence of sensing actions that minimizes the error in our estimate of the belief map. Since the true activity state is unobservable, a common approach is to minimize the uncertainty of our belief regarding the activity in the area (MacKay [1992a]). In this section, we examine different existing policies before describing our new approach.

Greedy Strategy

A common sub-optimal approach to the problem is a greedy strategy (Denzler and Brown [2002]; Guestrin *et al.* [2005]), which looks one step ahead and selects the sensing action that reduces uncertainty the most. In the example shown in Figure 5-9, the boxes represent the map and red indicates the amount of uncertainty. The greedy strategy takes a sensing action right in the middle of the map, where uncertainty is greatest. A non-myopic strategy, on the other hand, considers the entire mission duration, consisting of four sensing actions in this example. As the problem proceeds, we can see that the greedy strategy made a poor initial decision, leaving part of the map completely unobserved in the end. Consequently, the non-myopic strategy leads to reduced map uncertainty over the entire mission duration. Greedy strategies have a known worst case optimality of $(1 - \frac{1}{e})$ (Krause and Guestrin [2005][1]), which may often be acceptable for single-query problems. However, for repeated-query problems where the activity map of an area may need to be remeasured regularly (Sukhatme *et al.* [2006]), improving the policy is likely to be extremely valuable over the lifetime of the agent.

Open-loop Feedback Control

One non-myopic approach to map exploration is open-loop feedback control (OLFC). This algorithm was covered in Chapter 3. We review it briefly within the context of our current application. To select a sensing action, OLFC generates an open loop plan (via search) for the mission duration, which minimizes uncertainty. After the first action is executed and

[1] $e \approx 2.71828...$

the belief is updated, a new open loop plan is generated for the remaining mission. The process continues for the mission duration.

While OLFC is effective, it is also intractable. For example suppose we have a problem with 1,000 locations and we make 100 sensing actions per mission. In this case, there would be 10^{300} possible plans, which is intractable even with the fastest of today's computational resources. To make matters worse, the detections are probabilistic, which makes evaluating plan outcomes more expensive. Furthermore, we need to update our belief about the world after each sensing action, which requires us to re-plan in real-time. Clearly this is not the right approach. Next, we formulate the problem as a POMDP and analyze the complexity of finding an optimal control policy.

Exploration: POMDP and MDP formulations

As discussed in Chapter 2, the most general formulation of planning under uncertainty is the partially observable Markov decision process (POMDP). Since the agent is unable to observe the state, it keeps a probability distribution over states called the *belief*.

We previously noted that if the reward function depends on the belief—which is often the case for the exploration task—the state must include the belief in itself. On the other hand, if the observation model is known a priori, the belief can be updated on each time step given the observation. These facts allow us to operate in a fully observable domain where the new state is the belief itself and the new transition model incorporates the observation model. This formulation is also known as an information state MDP, where an MDP is defined as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'}, s_0, \gamma)$ following the POMDP notation.

Solution Complexity

Solving for the optimal sensing policy involves computing a value function over the entire belief space, or space of possible probability distributions $p(s)$. Examining the structure of the POMDP value function illustrates the intractability of using standard POMDP solution techniques to address adaptive sampling problems. A POMDP value function is given by the supremum of a set of vectors over the belief space, where each vector has dimension m (m is the size of the state space). For an activity map of size n, there are 2^n states, which means that each value function component is a vector of length 2^n. For problems larger than a few hundred states, the problem of even representing the value function becomes challenging. Similarly, there are 2^n possible observations; since the complexity of computing a POMDP solution grows at least exponentially with the number of observations, for exploration problems the complexity of the solution grows doubly exponential with the number of locations. Note that even state-of-the-art approximation techniques such as belief compression or point-based value iteration (Pineau *et al.* [2003]) cannot scale to these problem sizes. Our approach, described next, provides a non-myopic solution while maintaining tractability.

Exploration as an information-state MDP

A general model of the active sensing problem as an info-MDP was provided in section 2.2. Here, we tailor the model to the application at hand.

The value of the POMDP approach is that the policy describes how to act given any probabilistic state estimate, or belief $p(s)$. The drawback is that any decision-theoretic formulation of the exploration problem runs into the inevitable complexity of computing an expected cost, which for a single belief requires evaluating the cost and probability of all possible states (2^n states for n locations). We can, however, evaluate the *uncertainty* of the belief relatively easily. The calculation is a linear operation with respect to the number of locations (n). Hence we follow our info-MDP formulation with an information state reward function:

- **State Space:** $\mathcal{S} = [0, 1]^n$ where n is the total number of grids and s_i indicates the probability of grid cell i occupancy.
- **Action Space:** $\mathcal{A} = \{1, 2, \ldots, n\}$, where an action a is to sense at a particular grid location a and the surrounding cells.
- **System Dynamics:** Instead of calculating the whole $\mathcal{P}^a_{ss'}$ function, we introduce $\mathcal{P}^a_{s_i s'_i}$ that specifies the belief transition for grid cell i. If the action does not include observing grid cell i, then $s'_i = s_i$ with probability 1. On the other hand if grid cell

i is observed as o_i where g_i corresponds to the actual grid occupancy, the transition following Equation 5.2 is calculated as:

$$\begin{aligned} s_i' &= \alpha T_+ s_i \\ s_i'' &= \alpha F_- s_i \\ \mathcal{P}^a_{s_i s_i'} &= s_i T_+ + (1 - s_i) F_+ \\ \mathcal{P}^a_{s_i s_i''} &= s_i T_- + (1 - s_i) F_- \end{aligned}$$

where

$$\begin{aligned} T_+ &= P(o_i = 1 | g_i = 1) = P(true - positive), \\ F_+ &= P(o_i = 1 | g_i = 0) = P(false - positive), \\ T_- &= P(o_i = 0 | g_i = 0), \\ F_- &= P(o_i = 0 | g_i = 1), \end{aligned}$$

and α is the Bayesian parameter.

- **Reward Model:** Since we cannot compute the expected cost of state estimation errors, we use the information-state approach of maximizing the reduction in uncertainty. One of the most common measures of reduction in uncertainty is reduction in entropy,

$$\mathcal{R}^a_{ss'} = H(s) - H(s'), \tag{5.3}$$

where

$$H(s) = -\sum_i \big(s_i \log_2 s_i + (1 - s_i) \log_2(1 - s_i)\big). \tag{5.4}$$

In the design of experiments literature, this is known as a “D-Optimal” objective function (Fedorov [1972a]). Another possible choice of information-state objective is the “E-Optimal” reduction in expected root mean squared error (RMSE). For a single variable binomial distribution with probability p, such as each grid cell in the activity map, the RMSE is

$$\begin{aligned} RMSE(p) &= \sqrt{(1-p)^2 p + (0-p)^2(1-p)} \\ &= \sqrt{p(1-p)} \\ \mathcal{R}^a_{ss'} &= \sum_{i=0}^{n} \left(RMSE\left(s_i\right) - RMSE\left(s'_i\right) \right), \end{aligned}$$

Figure 5-10 shows that RMSE is very similar to entropy. Furthermore, RMSE is directly related to expected map error, computed as the true occupancy (1 or 0) minus our estimate p squared, multiplied by the probability with respect to occupancy. Thus, RMSE is not just a reasonable proxy for entropy, it is actually a more appropriate reward, as it is directly related to our intended decision. However, the main benefit of using RMSE comes from its computational properties, which are described in the next section.

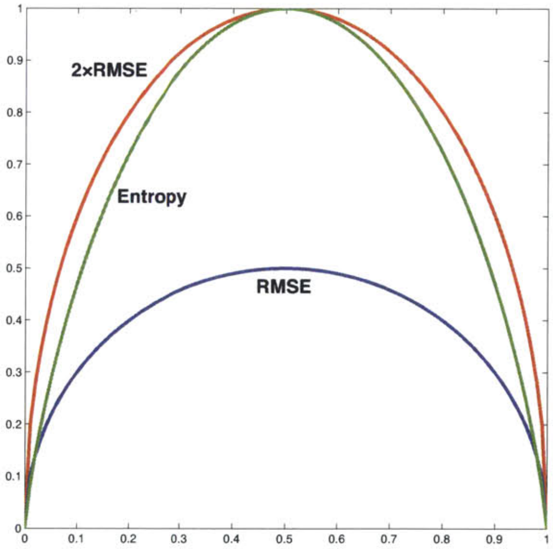


Figure 5-10: RMSE versus Entropy.

Our task is to find the policy (π) for selecting control action a in state s that maximizes the expected cumulative reward over the entire mission. Naturally, we can formulate the problem as a dynamic program, which maximizes the expected one-stage reward r plus the expected cumulative future reward V (Bellmann [1957]),

$$V(s) = \max_a E_{s'} \left[\mathcal{R}^a_{ss'} + V(s') \right]. \tag{5.5}$$

The dynamic programming equation is recursive. However, given the value function V, we could compute the optimal control action a. Therefore, knowing V is tantamount to knowing the optimal control policy. Our objective is, therefore, to find the optimal future reward function V.

Unfortunately, there are no known methods for solving a continuous Markov decision process exactly. Discretizing the belief space grows exponentially with the number of grid cells and discrete probability intervals. Even with a relatively coarse discretization and state-of-the-art algorithms for solving discrete MDPs, such as SPUDD and APRICODD (based on Abstract Decision Diagrams) (Hoey *et al.* [1999]; St-Aubin *et al.* [2000]) and (factored MDPs) (Guestrin *et al.* [2003]), cannot compute policies for problems with more than a few dozen states. We therefore approximate the value function V as a parametric function, and learn an approximation of V using dynamic programming.

5.1.5 Closed Form Least Squares Value Iteration

We outline our closed form LSVI approach in this section. We begin our discussion by outlining exact discrete value iteration backup. We then discuss the involved computational complexity and how our approach overcomes this hurdle. Specifically, we detail our function approximation architecture and alternate reward function. Our main contribution is that we can compute a value iteration backup in closed form (with respect to the current and approximate value functions). We prove our claim with a derivation.

Value Iteration

Our objective is to learn value function V. For a discrete state space, V is represented as a table of values (one record for each discrete state). The table of values can be initialized arbitrarily and improved iteratively. To iteratively improve the value function estimate, for each s, we find control action a that maximizes the expected $\mathcal{R}^a_{ss'}$ plus the expected value of next state $V(s')$.

$$V^{i+1}(s) \leftarrow \max_a E_{s'}\left[\mathcal{R}^a_{ss'} + V^i(s')\right] \tag{5.6}$$

The sensor measurements are stochastic; therefore, the expectation entails summing over the possible rewards and next states, times their corresponding probability.

$$V^{i+1}(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'}\left(\mathcal{R}^a_{ss'} + V^i(s')\right) \tag{5.7}$$

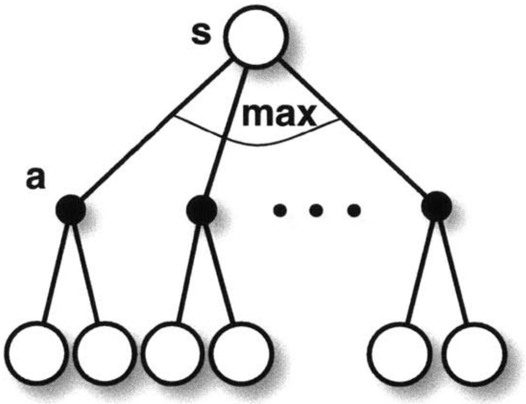


Figure 5-11: Value iteration backup diagram for discrete state-space: white circles represents states while black circles represent actions..

As shown in Figure 5-11, we compute this summation for each control action, then replace the current table entry $V(s)$ with the new maximum value corresponding to the best control action. The above process constitutes one iteration. We repeat the process until $V(s)$ converges.

Recall that the main problem with this approach is that we must perform a value iteration backup for every single state, which would take far too long (*e.g.* for 0.1 discretization of each belief and $1,000$ cells, we will end up with 10^{1000} discrete states). Furthermore, we cannot store the huge table representation of V in memory. We address these complexities by representing V as a parametric function. Furthermore, we employ a better reward function, which serendipitously allows us to compute the summation over possible next states in closed form.

Linear Architecture: Formally, a linear architecture approximates $V(s)$ by first mapping the state s to feature vector $\Phi(s) \in \Re^k$ and by computing a linear combination of those features $\Phi(s)\beta$, where β is a function parameter vector. We compose $\Phi(s)$ by applying basis function $\phi()$ to the probability of occupancy of each grid cell, such that

$$\Phi(s) = \begin{bmatrix} \phi(s_1) & \ldots & \phi(s_n) \end{bmatrix} \tag{5.8}$$

We also use an intercept that our description ignores for simplicity. We compute an improved value estimate $\hat{V}(s)$ for a finite set of states $S_i \in X$ via value iteration backups, such that

$$\hat{V}(s) = \max\nolimits_a E_{s'}\left[\mathcal{R}^a_{ss'} + V_{approx}(s')\right], \tag{5.9}$$

where $V_{approx}(\cdot) = \Phi(\cdot)\beta$.

We then update our estimate of V_{approx} using regression, where

$$\Phi(X) = \begin{bmatrix} \Phi(S_1) \\ \vdots \\ \Phi(S_\nu) \end{bmatrix} \tag{5.10}$$

is a set of feature vectors for all state samples and

$$\hat{V}(X) = \begin{bmatrix} \hat{V}(S_1) \\ \vdots \\ \hat{V}(S_\nu) \end{bmatrix} \tag{5.11}$$

is the updated value estimate for all state samples.

We re-estimate the parameters β of our function as

$$\beta = \left(\Phi(X)^T \times \Phi(X) + \lambda I\right)^{-1} \Phi(X)^T \hat{V}(X) \tag{5.12}$$

where ridge penalty λ ensures invertibility and can be increased to reduce estimation variance at the expense of increased bias. Our updated $V_{approx}(s) = \Phi(s)\beta$ now estimates our new target value $\hat{V}(s)$ in a least squares sense.

To summarize, we simulate a next state and plug that state into V_{approx}, our approximate value function. As the sensor measurements are stochastic, we must sum over the rewards of each possible future state.

Closed Form Value Iteration Backup

During a value iteration backup, for each control action we compute the expected reward plus expected (approximate) value, as a weighted sum over potential sensor outcomes. It is exactly here that we can realize the advantage of using the RMSE as a reward function, in that we can compute the expected $\mathcal{R}^a_{ss'}$ in closed form. Furthermore, we can also compute the sum over V_{approx} in closed form, allowing us to compute the entire expectation with respect to s' as a single dot product.

Closed Form E[RMSE]: Here we derive a closed form expression for the expected RMSE, given a set of t observation from action a and our prior belief s. We then show how this general derivation can be applied to a value iteration backup.

We first observe that the expected reward per grid cell is conditionally independent given t. Therefore, from t observations, we count the n observations of a particular grid cell due to the overlapping 3×3 footprints. We then compute the E[RMSE] for that cell. We define n as the number of observations for grid cell i, and k as the corresponding number of detections where a detection is an occupied grid cell. Hence:

$$P(occupancy|detection) = \frac{P(detection|occupancy)P(occupancy)}{P(detection)}. \tag{5.13}$$

Thus, a Bayesian update, computing the posterior probability that a cell is occupied given just one single detection ($n = 1, k = 1$), is expressed using the above notation as:

$$p_{post|n=1,k=1} = \frac{T_+ s_i}{T_+ s_i + F_+(1 - s_i)} \tag{5.14}$$

Next we will compute a Bayesian update for a particular cell, given a possible *set* of n sensor returns. We observe that the computed posterior is the same regardless of the sensor return order. Thus, the posterior probability that a cell is occupied given k detections (p_{post})

is

$$p_{post} = \frac{(1-T_+)^{(n-k)}\,T_+^k s_i}{(1-T_+)^{(n-k)}\,T_+^k s_i + (1-F_+)^{(n-k)}\,F_+^k\,(1-s_i)} \tag{5.15}$$

Similarly, the posterior probability a cell is unoccupied is

$$1 - p_{post} = \frac{(1-F_+)^{(n-k)}\,F_+^k\,(1-s_i)}{(1-T_+)^{(n-k)}\,T_+^k s_i + (1-F_+)^{(n-k)}\,F_+^k\,(1-s_i)} \tag{5.16}$$

Thus, E[RMSE] is computed as the sum over the posterior RMSE times the likelihood of getting k detections and $n - k$ non-detections, given prior s_i. We weight the likelihood given occupancy by the prior probability of occupancy (and vice versa), then multiply by the binomial coefficient (the number of k-element subsets of an n-element set) to get

$$\begin{aligned} &E\left[RMSE\right] = \\ &\sum_{k=0}^{n} \left\{ \begin{array}{l} \left((1-T_+)^{(n-k)}\,a^k s_i + (1-F_+)^{(n-k)}\,F_+^k\,(1-s_i)\right) \\ \times \frac{n!}{k!(n-k)!}\sqrt{p_{post}\,(1-p_{post})} \end{array} \right\} \end{aligned} \tag{5.17}$$

Notice that the normalizing denominator of a Bayesian update is also the likelihood of the data given the model. Therefore, when we spell out the posterior RMSE

$$\sqrt{p_{post}\,(1-p_{post})} = \frac{\sqrt{(1-T_+)^{(n-k)}\,T_+^k s_i\,(1-F_+)^{(n-k)}\,F_+^k\,(1-s_i)}}{\left((1-T_+)^{(n-k)}\,T_+^k s_i + (1-F_+)^{(n-k)}\,F_+^k\,(1-s_i)\right)} \tag{5.18}$$

we see that its denominator conveniently cancels out the observation likelihood, resulting in:

$$\begin{aligned} &E\left[RMSE\right] = \\ &\sum_{k=0}^{n} \tfrac{n!}{k!(n-k)!}\sqrt{(1-T_+)^{(n-k)}\,T_+^k s_i\,(1-F_+)^{(n-k)}\,F_+^k\,(1-s_i)} \end{aligned} \tag{5.19}$$

The key reason we are able to reduce equation (5.17) to equation (5.19) is that both posterior denominators are the same and $\sqrt{d^2} = d$, a reduction particular to RMSE. We then pull out the prior and reorganize the sensor parameters.

$$\begin{aligned} &E\left[RMSE\right] = \\ &\sqrt{s_i\,(1-s_i)}\sum_{k=0}^{n} \tfrac{n!}{k!(n-k)!}\sqrt{T_+F_+}^{\,k}\sqrt{(1-T_+)\,(1-F_+)}^{\,n-k} \end{aligned} \tag{5.20}$$

We then apply the binomial theorem:

$$(x+y)^n = \sum_{k=0}^{n} \frac{n!}{k!\,(n-k)!} x^k y^{n-k}, \tag{5.21}$$

If we let $x = \sqrt{T_+ F_+}$ and $y = \sqrt{(1-T_+)(1-F_+)}$ then

$$E\left[RMSE\right] = \sqrt{s_i\,(1-s_i)} \left(\sqrt{T_+ F_+} + \sqrt{(1-T_+)(1-F_+)} \right)^n \tag{5.22}$$

Set $c = \sqrt{T_+ F_+} + \sqrt{(1-T_+)(1-F_+)}$ (a constant of our sensor model). Note that equation (5.22) allows us to compute the expected RMSE reduction (our reward) for a given cell after n measurements. The computation is simply our prior RMSE times $(1 - c^n)$. Therefore, when performing a value iteration backup, we can quickly compute the expected current reward for the entire grid by applying $(1-c) \times \sqrt{s_i\,(1-s_i)}$ at the observed grid cells. Furthermore, we show that, using RMSE as our basis functions, we can compute the sum over V_{approx} in closed form. As it turns out, we also achieve good results by doing so.

Closed Form E[$V_{approx}()$] $V_{approx}\,()$ is our approximate value function, and β is the parameter set of our linear function over our features. We intentionally pick the RMSE formulation for our basis function: $\phi\,(\cdot) = \sqrt{\cdot\,(1-\cdot)}$. Specifically, we compute the RMSE per grid cell of the resulting posterior state. Serendipitously, these features allow us to approximate the value function reasonably well. Furthermore, they allow us to compute the expectation in closed form. First, we calculate the expectation over a single cell, then we extend the derivation for the entire state.

$$\begin{aligned} E_{s_i'}\left[V_{approx}(s_i')\right] &= \underset{s_i'}{E}\left[V\,(p_{post})\right] = \underset{s_i'}{E}\left[\beta_i \phi\,(p_{post})\right] \\ &= \beta_i \underset{s_i'}{E}\left[\sqrt{p_{post}\,(1-p_{post})}\right] \end{aligned}$$

where β_i is the ith element of the β vector. As shown above, we can compute $E_{s_i'}\left[\sqrt{p_{post}\,(1-p_{post})}\right]$ in closed form as $c \times \sqrt{s_i(1-s_i)}$. Consequently, we can combine the closed form expectation of $\mathcal{R}^a_{s_i s'_i} + V_{approx}(s'_i)$ into a single expression, such that

$$\sqrt{s_i(1-s_i)} + E_{s'_i}\left[-\sqrt{p_{post}(1-p_{post})} + \beta_i\sqrt{p_{post}(1-p_{post})}\right]$$

$$\begin{aligned} &= \sqrt{s_i(1-s_i)} + -c\sqrt{s_i(1-s_i)} + c\beta_i\sqrt{s_i(1-s_i)} \\ &= \phi(s_i)\big(1 + c(\beta_i - 1)\big) \end{aligned}$$

The above expressions are given for a single grid cell. We can compute a backup for all grid cells, given action a, as a dot-product by representing $\Phi(x)$ as a row vector and by constructing a column vector, containing a 1 for each unobserved grid cell and $\sqrt{T_+F_+}+\sqrt{(1-T_+)(1-F_+)}$ for each observed grid cell. Thus we can concatenate the multiplier vectors for all actions into a single square matrix C and compute an entire backup as a max over a single dot product:

$$\hat{V}(s) = \max \Phi(s)^T\big((1-C) + (\otimes\beta)\big)$$

where $\otimes$ is an element-wise multiplication. Finally, if we concatenate our feature vectors vertically, we can compute an entire backup for all *belief-state* samples X as a max over a single matrix multiplication (note addition of γ).

$$\hat{V}(X) = \max_{rows} \Phi(X)^T\left((1-C) + (C \otimes \beta\gamma)\right) \tag{5.23}$$

For additional efficiency, we approximate the finite stage problem as an infinite horizon problem with discount rate $\gamma = 1/T$, where T is the mission horizon. The mean of a geometric distribution with success probability $1/T$ is T. Thus, the discount rate was chosen such that the equivalent stochastic shortest path problem would have an average of T stages or actions. The infinite horizon approximation allows us to compute just one value function V_{approx}. Our results show that the approximation is effective. Algorithm 10 shows the closed form LSVI pseudo-code.

Algorithm 10 Closed Form Least Squares Value Iteration

```
**Input:** $(\mathcal{S}, \mathcal{A}, \mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'}, T)$
**Output:** Policy $\pi$
Initialize $\beta = [0...0]^T$
Construct $C$
**repeat**
  $X = samples(\mathcal{S})$
  $\hat{V}(X) = max_{rows}\Phi(X)((1 - C) + (C \otimes \beta \times \gamma))$
  $\beta = \left(\Phi(X)^T \Phi(X) + \lambda I\right)^{-1} \Phi(X)^T \hat{V}(X)$
**until** near-convergence
**return** greedy policy w.r.t $\beta$
```

5.1.6 LSVI Results

Our results demonstrate the significant benefit of our approach. We tested our method on problems with uniform prior beliefs (see test problems inset for details). Our method scaled from 25 grid cells to $1,089$ grid cells. We measured performance in terms of the normalized RMSE reduction over the mission horizon averaged over 15 $runs$. Figure 5-13 depicts the LSVI method on par with optimal solution, greedy, and random algorithms. As one can see, our method performed much better than the greedy approach on all problems. LSVI also performed optimally up to problem size 225, where "optimally" is computed using a Lagrange relaxation method applicable to this special case problem (Note that we did not compute an optimal POMDP policy. For even problems of size 20, a POMDP policy would have state, action and observation spaces of size 2^{20}). The performance of our algorithm tapers gracefully and levels off as the linear function approximation error starts to increase. These initial results are very encouraging; however, we are able to improve upon them using the rollouts algorithm (Bertsekas and Castanon [1999]).

Test Problems

It is important to measure the optimality gap reduction of our approach with respect to other approaches. While it is challenging to calculate the optimal solution value in general, we tested all methods on problem instances for which the optimal solution value could be calculated via Lagrange relaxation with our closed form $E[RMSE]$ expression. Our test problems have uniform 0.5 prior beliefs for all grid cells and each problem allows n sensing actions that can evenly cover the map. For example, we tested a 33×33 problem allowing 121 sequential sensing actions. Thus, the optimal open loop plan is to evenly tile the sensing area with 11 rows of 11 sensor footprints. We can prove this property via Lagrangian relaxation. General problems with unequal priors and an arbitrary number of sensing actions do not have trivial solutions. Our policies were trained on arbitrary problems. They performed well on both arbitrary and special case problem, with respect to the greedy solution. However, in order to provide optimality comparison, we focused our attention for the specific set of problems.

Lagrange Relaxation We solve the test problems using Lagrange relaxation, by relaxing the sensor footprint and integrality constraints. Thus, the relaxed problems allow a single sensing action to cover non-adjacent cells as well as a non-integer number of sensing actions per cell. The Lagrangian relaxation solutions satisfy the original constraints, thus are optimal.

5.1.7 Combining LSVI and Rollout

Rollout is a method for extracting a policy at runtime, which uses some additional computation but tends to make better action selections (Bertsekas and Castanon [1999]). For the full details of rollout, see Bertsekas [2007]. Here we provide an intuitive explanation. Typically, we determine the policy at runtime by finding the control action a which maximizes the dynamic programming equation (5.9), with respect to our approximate value function V_{approx}. On the other hand, the rollout algorithm selects the control action that maximizes the expected outcome with respect to a multi-stage rollout of the base policy. A multi-stage rollout computes (via simulation) the expected outcome of the base policy over the remaining mission horizon (Algorithm 11). In other words, rollout maximizes over the initial action choice based on where the base policy will lead (see Figure 5-12). By doing so, rollout makes better control action selection with one step look ahead. While rollout is somewhat expensive for stochastic systems (Bertsekas and Castanon [1999]), our closed form expected reward expression makes rollout fast. Figure 5-13 illustrates the result of fusing rollout method with LSVI. As expected, LSVI+Rollout boosted the performance more compared to the LSVI method. The average performance of LSVI+Rollout was always within 2% of the optimal solution on all problem cases.

Algorithm 11 Rollout Algorithm

```
  **Input:** $(\mathcal{S}, \mathcal{A}, \mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'}, T), s_t$, base-Policy $\pi$
  **Output:** Policy $\pi^+$
  $\tilde{Q}^\pi(s_t, a) = \mathop{\mathrm{E}}_{s'}\left[\mathcal{R}^a_{s_t s'} + \sum_{k=t+1}^{T-1} \mathcal{R}^{\pi(s_k)}_{s_k s'}\right]$
  $\pi^+ \leftarrow \pi$
  $\pi^+(s_t) \leftarrow argmax_a \tilde{Q}^\pi(s_t, a)$
  **return** $\pi^+$
```

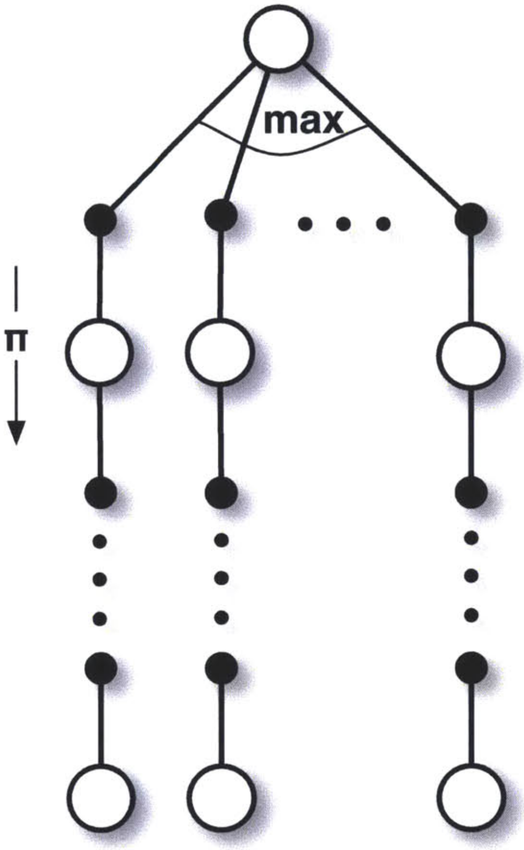


Figure 5-12: Rollout considers each action and simulates where the approximate policy will lead.

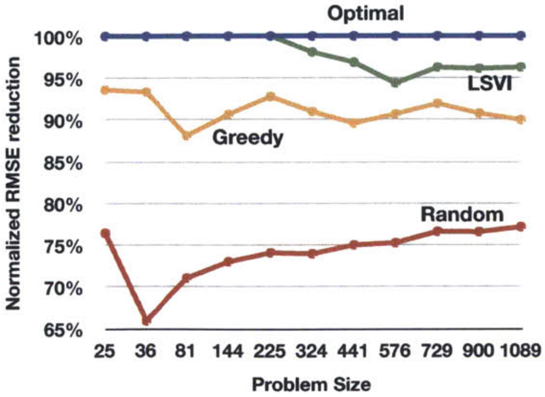


Figure 5-13: Normalized RMSE reduction for LSVI, Greedy, and Random methods. LSVI performed optimally up to problem size 225. Performance tapers and levels off as the problem size increases.

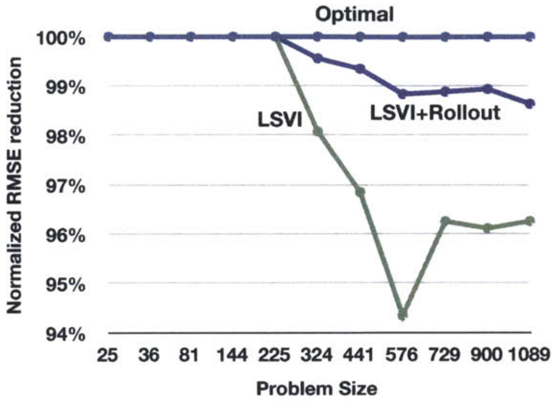


Figure 5-14: Normalized results: LSVI + Rollout performed within 2% of optimal on all problems.

5.1.8 Impact

Current methods employ greedy sensing action selection. The non-myopic approach has been avoided historically, since no exact or approximate MDP solver has been shown to work on relevant problem sizes in this domain. Our contribution is an information-state POMDP sensor tasking problem formulation and an efficient function approximation dynamic programming scheme for control policy learning. The result is efficient non-myopic sensing action selection.

Our key innovation, which makes function approximation dynamic programming efficient, is our derivation of a closed form expected reward expression for the information-state reward function RMSE. Likewise, we enable a closed form value iteration backup by selecting basis functions for our future reward representation, which coalesce with the current reward function. We further improve the performance of the resulting policy using rollout. Our method solved large state space problems within 2% of optimal, garnering significant improvement over the state-of-the-art of both optimal exploration and dynamic programming.

5.2 Mesoscale Application (Multi-Vehicle Aerial Survey Coordination)

Mesoscale multi-vehicle aerial survey coordination We can illustrate the mesoscale level of the hierarchy with a multi-vehicle aerial survey, which routes vehicles to multiple survey locations (See Soloman [1987], Lolla *et al.* [2012], Desrochers *et al.* [1988], Kilby *et al.* [2000], Kolen *et al.* [1987], Larsen [1999], Ralphs *et al.* [2003], and Fisher and Jörnsten [1997]). We tested an aspect of our DUM algorithm on this problem, by framing it as a Markov decision process, learning an off-line approximate policy, and refining the policy online using nested rollout. More specifically, our approach started with an off-the-shelf heuristic, which we enhanced using approximate dynamic programming, and further augmented online through decision uncertainty minimization planning. The results showed a clear improvement over current ways of approaching this problem.

We examine the complementary strengths of value function based policy learning and guided search. Our work unifies rollout based open-loop feedback control (outlined by Bertsekas [2005a]) and plan-space approximate dynamic programming (studied by Boyan [1998]).

Open-loop feedback control builds and executes a sequence of actions (an open-loop plan) at runtime. When the resulting real-world situation deviates from what was expected, the system replans. Reinforcement learning solves the same problem by learning an approximate value function, which is subsequently used to select actions. We typically cannot learn exact value functions for interesting real-world problems, as we observed in Section 5.1. Therefore, we use rollout (value function based limited search) to construct an open-loop plan that improves upon value function based action selection, using an approximate value function.

The above method is performed in the real-world state-space and actions are added to the plan in time order, the same order as they are executed in the real-world. On the other hand, plan-space search constructs an open-loop plan out of order. For example, a future action may be added to a plan even though our immediate action has not yet been determined. The term *plan-space* refers to the fact that we are searching though a space of plans. We can likewise guide plan-space search by learning a value function. A plan-space value function informs us of actions that should be added to our plan.

Rollout-based open-loop feedback control and plan-space approximate dynamic programming are generally considered orthogonal (Traverso *et al.* [2004]). However, we have constructed an inclusive framework that incorporates both methods. We present experiments to find natural planning action spaces, which prescribe allowable plan changes that meaningfully reflect their impact on global optimality. We demonstrate our approach on multi-UAV adaptive mission planning.

5.2.1 Application & Problem Formulation

Autonomous coordinated aerial survey (ACAS) involves multiple unmanned air vehicles (UAVs), which must coordinate with external mission requirements to accomplish an information gathering task. For example, suppose a high-level mission planner assigns a set of reconnaissance tasks to a multi-UAV team. Each task includes a time window constraint, which facilitates coordination with external assets. The autonomy objective is to minimize total team travel-time.

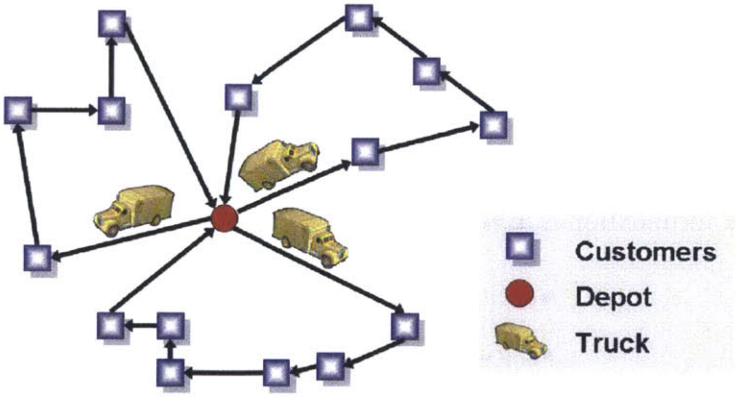


Figure 5-15: VRP Drawing

5.2.2 VRP-TW Problem Formulation

We formalize ACAS (in Table 5.1) as a vehicle routing problem with time windows (VRP-TW), following the notation presented in Fisher and Jörnsten [1997]. They state the prob-

Variable	Definition
m	number of vehicles
n	number of customers; index 0 denotes the depot
c_{ij}	travel time from location i to j
t_{ij}	time for travel from location i to j (equal to c_{ij} in our problem)
s_i	service time at location i (0 in our problem)
e_i	earliest time allowed for arriving at location i
u_i	latest time allowed for arriving at location i
N	$\{1\ldots$ n$\}$ the set of locations to visit
N_0	N$\cup\{0\}$ the location set including the depot
M	$\{1\ldots m\}$

Table 5.1: ACAS problem formulation

lem as a vehicle routing application where goods are transported from the depot and delivered to various locations by multiple vehicles, as shown in Figure 5-15. The delivery times are restricted by time windows and the distance between locations is computed as the Euclidean distance on a plane. The objective is to minimize the travel time such that locations are visited within the specified time windows. The time window constraints can make the optimal solution significantly more expensive.

5.2.3 Approximate Dynamic Programming

Approximate dynamic programming (ADP) combines function approximation and dynamic programming to learn an approximate value function for a Markov decision process (MDP). The basic MDP formulation is as follows:

- State: x
- Action: $u \in U(x)$
- Reward: $g(x, u)$
- Value Function: $J(x)$

The dynamic programming operator is subsequently applied to the value function at run-time to select the next action u:

$$u = \arg\max_u \mathrm{E}\left[g\left(x, u\right) + J\left(f\left(x, u\right)\right)\right]$$

Thus, learning a value function is tantamount to learning a feedback control policy.

5.2.4 Value Function Guided Search

A value function can also be used to guide a search. For example, the common A* algorithm prioritizes planning actions using the actual plan cost-so-far plus the approximate cost-to-go. A search in plan-space constructs a complete open-loop plan. Guided search requires a plan-space value function to help us search though a space of plans. The associated plan-space MDP is defined as follows:

- Plan-state: $x =$ the current plan
- Action: $u \in U(x)$ = plan augmentation
- Reward: $g(x, u)$ = immediate plan cost reduction
- Value Function: $J(x)$ = potential for plan improvement

5.2.5 Unified State-space and Plan-space MDP

The key observation is that our MDP definition is sufficiently general and can therefore describe both a state-space and plan-space framework. For example, we can view the state-space dynamic programming operator as constructing a one-step finite horizon plan. In this example, the current plan x is either empty or contains one action and action-space $U(x)$ includes all real-world actions. From this viewpoint, the state-space framework is just a special case of our plan-space MDP.

We extend this idea to include a planning strategy continuum from pure dynamic programming to guided search. We begin by describing Bertsekas' survey regarding open-loop feedback control and rollout (Bertsekas [2005a]). We then connect this idea to our general MDP planning model and the concept of natural planning action spaces.

5.2.6 Rollout

In this section, we review how ADP policies can be improved using rollout. A value based policy employs the control action that maximizes the dynamic programming operator. In effect, the dynamic programming operator simulates forward one step.

However, an approximate value function may yield a sub-optimal policy. Alternatively, rollout evaluates each initial control action; then simulates forward to the end of the planning horizon using the base policy. In other words, given an initial action, we simulate forward to see where the base policy leads. Essentially, rollout recomputes the Q-values (state-action values) at runtime guided by the approximate value function. Overall, rollout selects better control actions than the base policy.

The above paragraph describes a one-step rollout. We can generalize the notion of rollout to allow maximization at multiple decision points. For example, a two-step rollout examines each possible sequence of two actions, then simulates to the end of the planning horizon using the base policy. Ultimately, a t-step rollout, where t is the horizon length, is a complete forward search. Therefore, rollout defines a continuum between the one-step look-ahead dynamic programing operator and a complete search.

5.2.7 ADP and Guided Search

The above method is performed in action-space. In other words, actions are added to the plan in time order, the same order as they are executed in the real-world. On the other hand, plan-space search constructs an open-loop plan out of order. For example, a future action may be added to our plan even though our immediate action has not yet been determined. We can likewise guide plan-space search by learning a value function. A plan-space value function informs us of actions that should be added to our plan.

5.2.8 Open-loop Feedback Control

Search based planning extracts an action sequence for the entire planning horizon. Given a probabilistic setting, we would construct an open loop plan with the optimal *expected* outcome. During plan execution the system deviates from the expected state sequence, causing the remaining plan to be suboptimal. We respond to the system feedback by constructing a new open-loop plan. Specifically, we construct a complete open-loop plan; yet, we only employ the first action in the plan. This action selection approach (or policy) is known as open-loop feedback control.

5.2.9 Natural Planning Action Spaces

Our above description assumes temporally ordered rollout. In many cases, planning actions are more naturally performed in a different order. For example, when assigning tasks to UAVs, it is more natural to construct a complete route for a particular UAV. Specifically, greedy assignment of tasks to UAVs in temporal order results in a less optimal plan than holistic greedy route construction. This is an example of a natural planning action space. Our research employs natural planning action spaces combined with value based rollout. We experimentally discover natural planning action spaces, then learn a value function in plan-space using ADP. We then use the value function to perform OLFC using rollout in plan-space. In this way, we unify rollout-based OLFC (Bertsekas [2005a]) and plan-space ADP (Boyan [1998]).

5.2.10 Uncertainty

The VRP-TW formulation introduced in Section 5.2.2 implies a deterministic problem. However, ACAS involves uncertain system dynamics (e.g. sub-task completion time and high-level plan changes). Consequently, ACAS requires sequential decision making where actions are selected based on the current state. In the next section, we frame ACAS as a sequential decision making problem; however, we formulate each decision as a natural planning action space based dynamic program. Thus, we present our natural planning action space experiment results used to drive our approach. We then formalize our dynamic program and introduce rollout based policy execution.

5.2.11 Approximation Dynamic Programming Approach

Here we describe an ADP approach to the VRP-TW. To construct our ADP method, we first couch the planning problem as a sequential decision making problem, along a natural planning action space, where routes are constructed in stages. Next we formalize the problem in terms of states, actions, rewards, system dynamics and an optimization function. Finally, we reformulate the optimization problem as a dynamic program, and define the approximation architecture to represent the cost-to-go. Using this formulation, we learn an approximate solution using value iteration.

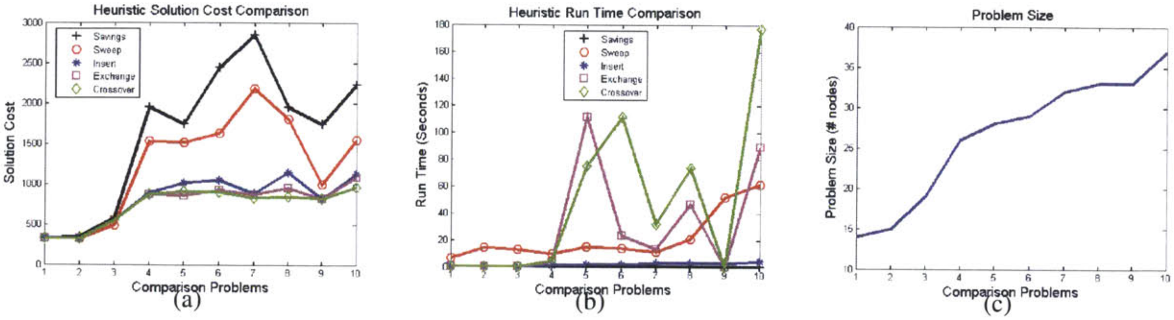
5.2.12 Sequential Decision Making Problem (Discovering a Natural Planning Action Space)

To formulate the VRP-TW planning process as a sequential decision making problem in a natural planning action space, we examined current VRP planning heuristic approaches. These heuristics are human-generated metrics to direct the planning process toward a reasonable solution. A good heuristic should be reasonably related to the cost gradient of the solution variable assignments. Alternatively, we view these heuristics as a natural planning action space. In particular, a heuristic merely restricts the set of planning actions we may take, given a particular planning state. For example, an unrestricted planning action space allows us to change any plan to any other plan. An unrestricted action space produces an optimal solution; however, it also leads to a computationally intractable search. Therefore, we seek a restrictive planning action space, which enables local improvements that are consistent with a globally optimal solution.

We compared the performance of the following methods on the VRP-TW: insertion, savings, sweep, crossover and exchange. A summary of the heuristics is presented in Appendix A. The results (Figure 5-16) of our comparison showed that the savings and sweep constructive heuristics perform poorly but the insert heuristic works quite well. In our experiments, we used the insertion route construction technique to provide a starting solution for the crossover and exchange improvement methods. The crossover and exchange methods garnered modest improvements over the starting solution. In other words, although crossover and exchange worked well, their results stem largely from the effectiveness of the insertion heuristic. Consequently, we designed our ADP algorithm around insertion.

5.2.13 Insertion Method & Problem Formulation:

The insertion method initially constructs n single node routes, where n is the number of locations to visit. The nodes in these single node routes are referred to as the "unallocated nodes." The insertion method starts building the first real route by allocating one node to it.



(a) (b) (c)

Figure 5-16: VRP-TW heuristic comparison (solution cost, run time, problem size)

Next we evaluate this *basic feasible plan*'s cost. The plan cost is the sum of the unallocated single node route costs plus the allocated route cost. We select the next node and route insertion location that reduces the current plan cost the most. The plan is then augmented by removing the respective node from the unallocated set and inserting it into the real route. The resulting cost reduction of the augmented plan is computed and the process is repeated. When the current route is full, we begin another one. The process repeats until all nodes are allocated.

To elaborate, the insertion method is comprised of a sequence of insertion decisions. At each iteration we acquire a plan cost reduction, represented by a current reward function $g(x, u)$, where x is the current state of the planning process and u is our insertion action. Action u consists of inserting an unallocated node at a particular position in the partial path, which we are currently building. Thus, action space U(where $u \in U$) is the Cartesian product of the unallocated node set and the partial path position set.

$$u \in U = \{\{unallocated\_nodes\} \times \{partial\_route\_positions\}\}$$

Similarly, the state also consists of the unallocated node set and the current partially constructed route. Our plan state does not include the fully constructed routes because they do not affect the future plan cost reduction.

$$x = \{\{unallocated\_nodes\}, \{partial\_route\}\}$$

Based on the above reward structure, the objective of our sequential planning problem is to maximize the total plan cost reduction over the entire n-stage planning process.

$$\max_{u_k} \quad \sum_{k=0}^{n-1} g\left(x_k, u_k\right)$$

Thus, a planning trajectory leading to the optimal solution maximizes the above function. However, the insertion method does not optimize this function because it makes greedy choices with respect to the insertion metric. Our approach is to reduce the greediness by learning an approximate cost-to-go function in plan-space.

The cost-to-go (J) is the maximum reduction in plan cost that can be attained starting from a given plan state. For example, at the beginning of the planning process, the cost-to-go is equal to $\max_{u_k} \sum_{k=0}^{n-1} g\left(x_k, u_k\right)$ where $x_0 = \left\{\left\{all \quad nodes\right\}, \{\phi\}\right\}$. Thus, the initial cost-to-go is the cost of n single node routes minus the optimal plan cost.

At subsequent planning stages, the cost-to-go may be less than the difference between the current plan cost and the optimal plan cost. Due to a suboptimal prior decision, the optimal plan may no longer be reachable from the current plan state. However, if we knew the actual cost-to-go, we could use it to direct planning decisions toward the optimal solution. Specifically, the current reward $g(x, u)$plus the cost-to-go $J\left(f\left(x, u\right)\right)$ is maximal on the trajectory leading to the optimal solution[2]. This property is known as the principle of optimality (Bellman and CORP [1952]). Using this principle and the assumption of a correct cost-to-go function$J\left(\bullet\right)$, we can state the following:

$$\max_{u_k} \quad \sum_{k=0}^{n-1} g\left(x_k, u_k\right) = \max_{u_0}\left\{g\left(x_0, u_0\right) + J\left(f\left(x_0, u_0\right)\right)\right\}$$

In other words, if we select the local planning decision that maximizes the current reward plus the cost-to-go, we will construct the optimal plan. We formalize these concepts as a dynamic program.

[2] Note that system dynamics function$f\left(x, u\right)$yields the next planning state given the planning action.

5.2.14 Dynamic Programming Problem

The dynamic programming problem and algorithm are defined as follows:

State	$x = \{\{unallocated\_nodes\}, \{partial\_route\}\}$ $T = \{\{\}, \{*\}\}$ (terminal state; with empty unallocated node set)
Actions	$U(x) = \{\{unallocated\_nodes\} \times \{partial\_route\_positions\}\}$
Dynamics	$x' = f\left(x, u\right)$ where$f$performs planning action u (node insertion) on state x, yielding state x'.
Rewards	$g(x, u) = \cos t\left(x\right) - \cos t\left(x'\right)$
DP Algorithm	$J\left(x\right) = \max_u \left\{g\left(x, u\right) + J\left(f\left(x, u\right)\right)\right\}$ $J(T) = 0$

5.2.15 Approximation Architecture and Learning

The dynamic programming (DP) formulation above assumes a correct cost-to-go function J. Using DP we could theoretically compute J. However, due to the extremely large state space, the computation is intractable. Instead, we represent the cost-to-go as a continuous mathematical function and learn an approximation using value iteration. The general approach is known as approximate dynamic programming (ADP).

We represent the cost-to-go function using non-parametric least squares regression with a polynomial kernel. Please see Appendix B for details. Our features consist of indicator variables of our state. For example, given the DP formulation above, we represent the current state as two concatenated vectors of binary variables indicating whether each node is part of the unallocated node set or the current partial route.

We compute our cost-to-go function using value iteration. Specifically, we initialize our approximate function (J_{approx}) arbitrarily, and compute improved estimates ($\hat{J}$) for a set of states samples (X_{samp}) by maximizing the DP equation with respect to the currentJ_{approx} for each x:

$$\hat{J}\left(x\right) = \max_{u_k} \left\{g\left(x, u\right) + J_{approx}\left(f\left(x, u\right)\right)\right\}$$

We then re-estimate function J_{approx} from value samples $\hat{J}\left(X_{samp}\right)$ using non-parametric least squares regression with a polynomial kernel. The process proceeds for a set number of iterations or until convergence. The resulting value function is subsequently used for select planning decisions such that π represents our base policy:

$$u = \pi\left(x\right) = \arg\max_{u}\left\{g\left(x,u\right) + J_{approx}\left(f\left(x,u\right)\right)\right\}$$

5.2.16 Rollout Policy

Our value function is approximate. Consequently, π is suboptimal. However, we can augment the value-based policy by using rollout to improve action-selection by judiciously searching forward in the plan space and evaluating the consequences of possible decisions. Specifically, we maximize over the initial action choices while following the base policy thereafter. For comparison, the optimal solution can be computed by breadth-first search over the entire plan space. However, the plan space *size* is exponential in the plan space *depth*, where the depth is the number of decisions required to produce a complete plan. Thus, the breadth-first search approach is intractable. Rollout can be thought of as a compromise between base policy π and breadth-first search.

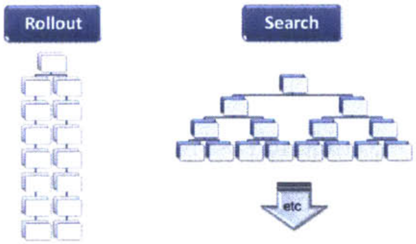
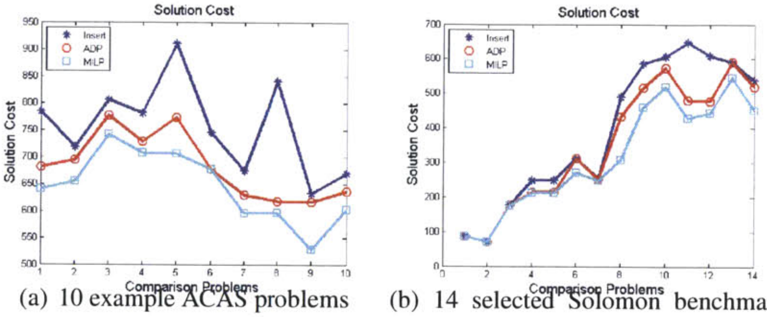


Figure 5-17: Rollout versus breadth first search

5.2.17 Experiments

Our approach uses ADP to find a better cost-to-go with respect to a natural planning action space. We then use rollout to extract an action in real-time. To assess our approach, we compared three approaches to the overall planning problem. Firstly, we used a standard approach to solving the optimization given by a Mixed Integer Linear Programming



(a) 10 example ACAS problems

(b) 14 selected Solomon benchmark problems

Figure 5-18: Performance results (Insert, ADP and MILP).

approach implemented using a commercial MILP solver (CPLEX). Second, we used a standard heuristic from the literature (specifically, an insertion heuristic). The MILP approach provided the optimal solution, but scaled extremely poorly. The heuristic approach scales well, but generally provides worse performance. Third, we examined our ADP algorithm.

The image on the left side of Figure 5-18 shows the comparison of the insertion heuristic, ADP (with rollout) and the MILP solver over a candidate set of 10 problems taken from the ACAS domain. The representative problems have 20 reconnaissance locations, time windows, and target-set coverage requiring multiple UAVs. In other words, it is impossible to cover the target set in the allotted time without using multiple UAVs (due to the number of targets, time windows and special dispersion). The targets are randomly dispersed over a 100 by 100 area. The base is located at the centroid. The time windows are randomly generated with an average width 10% of the mission duration. We restricted the analysis to problems small enough (20 targets) that the MILP solver could consistently find the optimal policy. Notice that the ADP solver generally finds low-cost policies in the same problems as the MILP solver, and always outperforms the heuristic.

To demonstrate the general applicability of this approach to resource allocation problems, we next tested the ADP algorithm against Solomon Benchmark Problems. The problems highlight several parameters affecting routing algorithms, specifically geographical dispersion, number of nodes serviceable by a single vehicle, percentage of time-constrained customers and the tightness / positioning of the time windows. In brief, the problems differ with respect to the width and density of our constraint of interest (time windows). To construct a useable problem set, we started with the Solomon VRP-TW type RC2 problems

(Soloman [1987]). The RC2 problems vary in both geographic dispersion (i.e. random and clustered placement) and time window tightness. We reduced the number of locations in each problem such that a MILP exact solution could be calculated. We then ran the MILP solver on each problem and kept each problem that could be solved in 10 minutes.

The right-hand side of Figure 5-18 again shows a comparison of the Insertion heuristic, ADP and MILP solvers. The difference between the insertion and optimal policies is less noticeable, but again the learned policy performs at least as well as the heuristic and usually better. It is worth noting that these were the only 14 problems in the data set for which CPLEX could compute a solution. Our ADP algorithm can easily scale up to much larger problems while, presumably, maintaining good quality results.

5.2.18 Problem Class Meta-Learning & Future Work

Above we learned a cost-to-go function which essentially estimates how much a plan change will improve our solution. The function estimate is based on problem specific features. The estimate is subsequently used to search and construct a plan. Given the above method, we must learn a cost-to-go function for each problem. However, we would like to learn a cost-to-go function for a whole class of problems, such as might be encountered by a small fleet of UAVs covering a particular area of responsibility from a forward operating base. We would learn one function over a range of missions, and use that function for planning. In that case, all of the expensive upfront learning can be performed offline, and only the final planning is performed in real time.

To accomplish this goal, we must be able to estimate the cost-to-go based on problem specific features, in addition to search state as demonstrated above. As a first step, we simply tried to construct a single function that would estimate the optimal solution cost over a class of problems. To do so, we constructed a set of 100 problems and solved them using MILP. We then extracted various problem specific features from each problem, based on time-window tightness, travel distances and target dispersion. We then constructed a function which mapped these features to the optimal solution cost. Our results show that we can accurately estimate problem cost features. Our results lend strong support for performing meta-learning over a problem class.

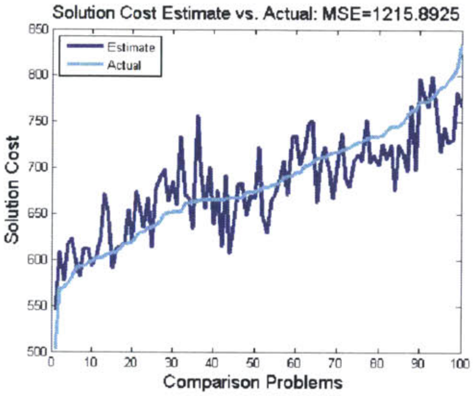


Figure 5-19: 100 problem comparison of estimated solution cost to MILP (open-loop optimal) solution cost.

5.2.19 Conclusion

In this section, we presented an idea that exploits the strengths of both value-based policy construction and search. We showed how to discover a natural planning action space experimentally, then described how to learn a value function based on that action space. Finally, our experimental results demonstrated that rollout policy extraction, with respect to our value function, selects good actions.

Our next step is to generalize our approach, such that a single value function can apply to a wider range of problem instances. A major challenge is to find representative basis functions for our linear architecture. Section 5.2.18 outlines our initial work in this area, and demonstrates that we can learn a function that approximates the optimal solution cost of a wide range of problems. While the idea is an unconventional approach to planning, it represents a viable approximate method to heretofore intractable problems.

5.3 Microscale Application (Disaster Relief)

Active detection of drivable surfaces in support of robotics disaster relief missions At the microscale level of the hierarchy, autonomous agents must make decisions based upon the information they collect while exploring the local area. In this section, we describe an application that employs an abstract Bayesian sensor that allows AUVs to optimally survey an area to find roads for rescue vehicles to follow to a rescue site. The system algorithmically determines which mission path will minimize risk to the ground vehicle while optimizing response time. Our test results showed that DUM methods were more effective than ADP plus rollout for risk-directed adaptive sampling problems (Bush *et al.* [2012]).

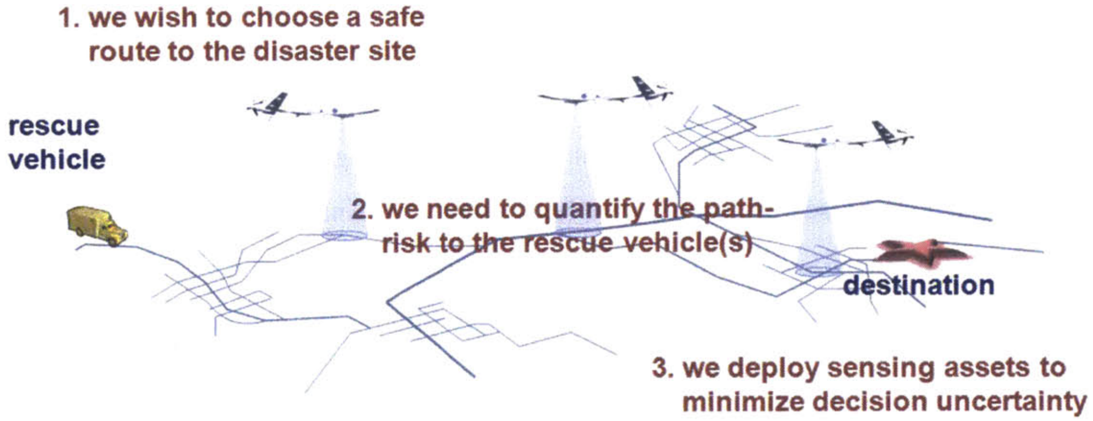

1. we wish to choose a safe route to the disaster site
rescue vehicle
2. we need to quantify the path-risk to the rescue vehicle(s)
destination
3. we deploy sensing assets to minimize decision uncertainty

Figure 5-20: Motivating Scenario: Natural Disaster Relief Scenario

While sensing and avionics technology has made great strides in the past decade, the autonomous capabilities of UAVs lag behind. Autonomous systems lack humans' creative reasoning abilities to maintain safe and robust behavior in uncertain environments. Thus, most unmanned systems still rely on human tele-operation, which is manpower intensive and requires reliable communication with a ground base. In this section, we propose a new approach to adjustable autonomy for reducing operator workload while retaining trust that the mission will proceed safely. Previous work in adjustable autonomy usually involves humans manually assessing the utility and setting the level of autonomy. This approach requires them to maintain complete situational awareness throughout the mission. However, applications that demand large-scale distributed efforts, such as disaster relief, environmental surveys, and convoy protection render this approach impractical. Our goal is to design a system that assesses its own abilities and requests human assistance in the amount needed. The challenge is to build the trust of the human operator that this system truly knows what assistance it needs and can operate safely otherwise.

Our approach to adjustable autonomy gains operator trust by making and explaining decisions based on risk. Our work is developed in the context of a disaster relief scenario, where a rescue vehicle must traverse an uncertain landscape to administer aid. The mission planner must tradeoff between risk reduction and decision deadlines. We consider two aspects to risk: the probability of mission failure and decision uncertainty in the mission planning process. Due to the inherent uncertainty in our belief, the probability of mission failure is not known exactly, so the mission planner evaluates risk as a distribution over this probability. In particular, the metric used to evaluate risk is the confidence that the probability lies above a threshold. The metric is calculated over the entire rescue vehicle mission path and it is calculated through a combination of Dijkstra's algorithm and value iteration. These risk distributions give rise to uncertainty when choosing between mission plans for lowest probability of failure.

To mitigate this, aerial scouts are sent ahead of the rescue vehicle to reduce belief uncertainty where it is needed most. The scouts are tasked with surveying areas that yield the highest expected uncertainty reduction within the time before the mission vehicle makes its next decision. Path planning for the scouts is achieved via approximate dynamic programming using a stochastic value function represented by a Gaussian process. For each scout, we use a unique representation of the state-space which is centered and rotated around the vehicle. Based on the updated belief provided by the scouts and the timing constraints of the scenario, the mission planner determines whether mission plans exist that have low enough risk associated with them.

Based on a series of risk thresholds, different amounts of human assistance are requested. Thus, we can engage the operator specifically when needed and at the proper amount, as well as provide reasoning and context for why and where the operator's help is needed. Simulation experimental results are presented which verify that the scout planner effectively reduces belief and decision uncertainty given any area to survey. The results demonstrate the potential of this new approach to achieve consistent safety and robustness throughout a mission by requesting operator assistance at the appropriate times.

5.3.1 Innovations

Adjustable autonomy has long been appealing for its vision to utilize the best abilities of humans and autonomous agents. However, the difficulties in creating a truly synergistic, dynamic relationship between humans and agents make adjustable autonomy a loosely defined concept. In this section, we introduce the risk-based innovations of our approach to adjustable autonomy, describe the system architecture, and explain the major algorithms driving our framework.

An adjustable autonomy system makes two types of decisions: The first type determines what vehicle actions to take in the future and the second governs when and how to engage the human operator. *Our first innovation is that we ground both these decision types on the risk to the mission goals.* Given a logistical plan for the mission, we probabilistically quantify the risk to each component of the plan. The component risks imply a risk configuration over each mission goal, which informs how we engage the human. Furthermore, the notion of risk is built into the planning process. Our adjustable autonomy architecture takes advantage of this to provide situational awareness, keeping the human involved at the appropriate level of detail for each mission component.

Our second innovation is how we improve our knowledge of the risks to the mission through the use of scout aerial vehicles. Using only the initial low-resolution knowledge of map risk severely limits the quality of decisions we can make, so we dynamically deploy scouts to collect more detailed information. Our algorithms lead the scouts toward the most valuable data that will help us identify the least risky paths for future components of the mission. The scouts are tasked to survey areas least well known about relative to the logistics plan. Flying over an area that our logistics vehicles will never cross on ground is useless unless it lies on the quickest path from one area of interest to the next. In summary, our contribution to adjustable autonomy is to encode risk at each decision-making process.

5.3.2 Architecture

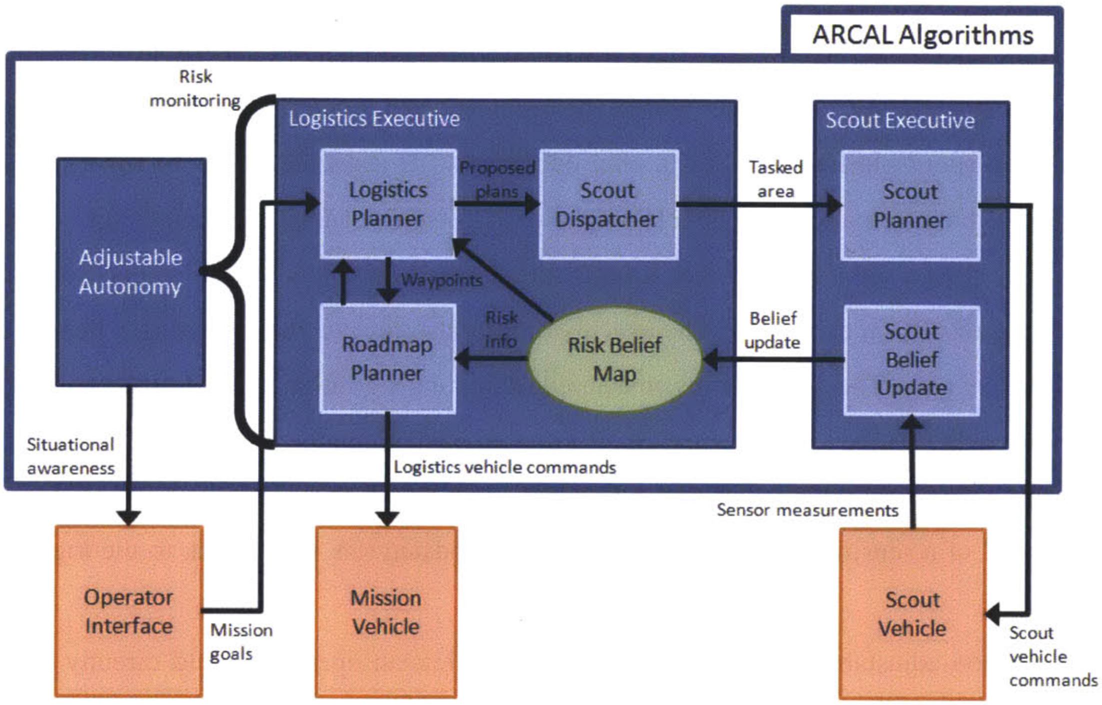


Figure 5-21: System Architecture

The algorithmic modules within our artificial intelligence architecture enable the incorporation of risk information and involvement of the human operator. The modules include the logistics executive, the scout executive, and the adjustable autonomy module. These components interact with the logistics vehicle, the scout vehicle, and the human operator, respectively, depicted in Figure 5-21. The logistics executive contains several sub-modules. Two of them are the high-level logistics planner and the low-level roadmap planner, each containing a risk assessment functionality that operates on the risk belief map. Together, these items determine the course of action for the logistics vehicle. The logistics planner accepts mission goals from the operator and generates sequences of waypoints, producing a high-level roadmap which will achieve the mission goals. Then, finding the actual path taken between waypoints is performed by the roadmap planner.

It is the job of the logistics planner to choose the actual sequence of waypoints in such a way that it balances and reduces the risk among each component of the mission. However, to make well-informed decisions, it will need the scouts to gather additional data in areas the logistics vehicle may cross in the future. The scout dispatcher determines where to send the scouts, given the plans currently considered by the logistics planner. Each plan has some uncertainty in how truly risky it is to execute. This plan risk uncertainty is mapped into map uncertainty, or, in other words, the scout dispatcher determines the map locations that are responsible for most of the plan uncertainty. It tasks the scouts to survey these areas, for the information they collect will help disambiguate candidate plans. Each scout's executive runs a scout planner that accepts these areas as input as well as a time limit for reporting results on each area. The executive must also receive the current risk belief for the relevant area. The planner runs an adaptive sampling algorithm that is trained to fly the path that achieves the highest expected information gain within the time allotted. As sensor measurements arrive, the belief update module incorporates them into the risk belief, and at the end of a sensing task, the scout reports the updated risk belief back to the logistics executive.

In a non-adjustable autonomy architecture, the human operator would directly interface with the logistics executive, but here, the adjustable autonomy module mediates their interaction. This module continuously monitors the risk associated with each mission component according to the entire state of the logistics executive. It tracks the possibility that each component's risk might exceed user-specified thresholds. As these risks evolve due to additional planning and updated risk beliefs, adjustable autonomy gradually requests human attention or intervention for certain mission components. Thus, while the operator still specifies mission goals to the logistics planner, she now has an interface to override the different components of the logistics executive at varying levels of control. Together, all these modules provide a rational, risk-based framework to help direct the operator's attention to the most pressing issues.

5.3.3 Disaster Relief Algorithms

Risk Assessment

A key capability of our system is to assess the risk to the mission goals. Risk is defined as the likelihood that the logistical plans we produce will achieve each goal. In other words, if a plan is to succeed, then every part of the plan must succeed. The risk assessment problem is then stated as follows: Given a path plan that nominally achieves the mission goals and a belief map of the environment, we wish to compute a distribution over the path success probability. While we would like to know the true path success probability, this is impossible since we do not have the true map of the environment. Rather, we possess a belief map which models not just where we think certain features and obstacles are, but also how well we know them. This reflects the intuition that although we may know a certain type of obstacle exists at a general vicinity, without extremely fine sensing, we have only a general idea of its precise location and threat level. Thus, we must compute and our algorithms must operate on a probability distribution over the success probability, i.e. a *risk distribution*.

Given this definition of risk, we describe how to represent risk in a belief map. We can then build paths over the map and operate on the risks it encounters to devise a risk distribution for the path. Finally, we explain how the scout measures and updates the risk belief map.

Our belief map is represented by a grid of square cells. Each cell contains a distribution over the probability of success if we traverse that cell in any direction, independently of all other cells. We use this interpretation because it allows us to use the Markov assumption later when composing cells into paths. The distribution in each cell takes the form of a Beta distribution, which represents a distribution over the probability parameter p of a binomial distribution. It is parameterized by (α, β), where α and β are the effective number of observed successes and failures, respectively. The mean and variance of a Beta distribution are given by

$$\mu = \frac{\alpha}{\alpha + \beta}$$

$$\sigma^2 = \frac{\alpha\beta}{(\alpha + \beta)^2(\alpha + \beta + 1)}$$

We may compute the inverse relationships as well:

$$\begin{aligned}\alpha &= \frac{\mu^2(1-\mu)}{\sigma^2} - \mu \\ \beta^2 &= \frac{\mu(1-\mu)^2}{\sigma^2)} - \mu + 1\end{aligned}$$

In our belief map, we parameterize each cell with a mean and variance to represent a Beta distribution. Not only does the Beta admit an intuitive interpretation, its parameterization is also appealing for real-time calculation.

The form of our belief map makes it convenient to compose cells into paths, although we will need to approximate the distribution for the resulting path. We make the Markov assumption that the probability of successfully traversing a certain cell is independent of the probabilities for other cells. Then, given a path of length n cells with random variable probabilities $p_1, \ldots, p_n$ of successfully traversing each one, the success probability for the entire path is

$$p = \prod_{i=1}^{n} p_i \tag{5.24}$$

Unfortunately, the true distribution for the entire path is not a Beta distribution and is hard to compute, so we approximate it as a Gaussian. Gaussian distributions are also parameterized by a mean and variance. Thus, by applying the independence property, we obtain the following expressions for mean and variance of the path's risk distribution.

$$\begin{aligned}\mu = E[p] &= E\left[\prod_{i=1}^{n} p_i\right] \\ &= \prod_{i=1}^{n} E[p_i] = \prod_{i=1}^{n} \mu_i \\ \sigma &= E[p^2] - E[p]^2 \\ &= E\left[\prod_{i=1}^{n} p_i^2\right] - \mu^2 \\ &= \prod_{i=1}^{n} E\left[p_i^2\right] - \mu^2 \end{aligned} \tag{5.25}$$

$$= \prod_{i=1}^{n} (\sigma_i^2 + \mu_i^2) - \mu^2 \tag{5.26}$$

Equation 5.25 is straightforwardly interpreted as the estimated risk. Equation 5.26 specifies that due to the multiplicative nature of Equation 5.24, a cell will amplify the variance effects of other cells if both its mean probability of success and its variance are large, and vice versa.

The Gaussian introduces the issue that it extends to $\pm\infty$. Thus, we introduce another approximation by truncating the distribution at 0 and 1 and scaling the resulting curve so that the area beneath it integrates to 1.

There still remains the question of how we measure and model obstacles from the environment into our belief map. We assume our sensor has algorithms for detecting and characterizing features of the environment. For example, suppose the scout's camera detects a pothole and computes a "measurement" of p for the success probability. If the camera's resolution is characterized by a variance σ_z^2, then the pothole's risk distribution is characterized by $\mu = p$ and $\sigma^2 = \mu_z^2$. Now, we must encode this information into the grid cells c_i that the pothole occupies. Assuming each grid cell has the same distribution, we take the characteristic length d of the pothole (such as diameter or side length), and invert

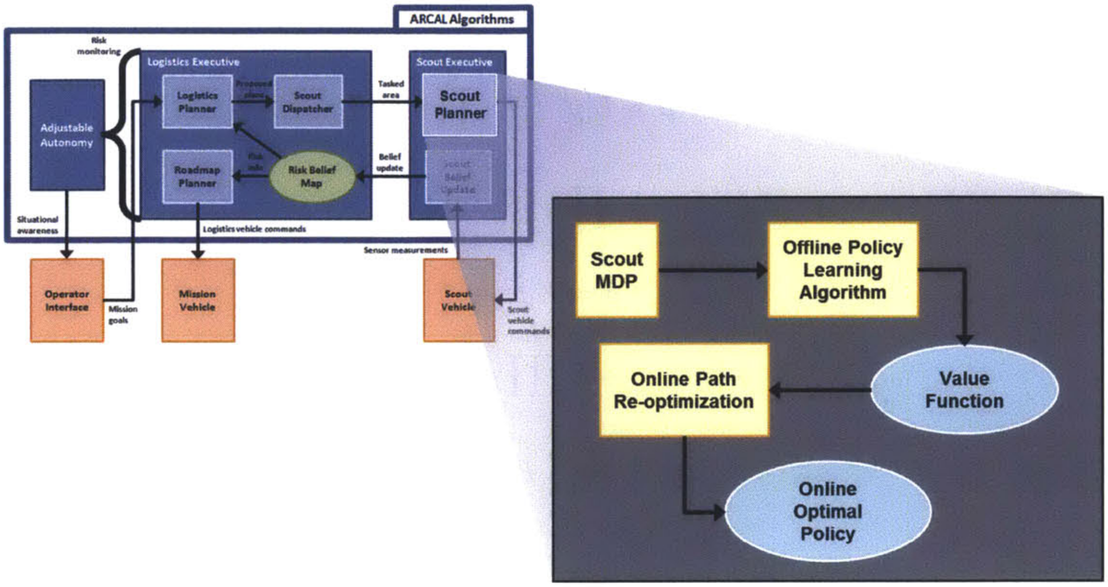


Figure 5-22: Scout Architecture

the relationships in Equations 5.25 and 5.26.

$$\begin{aligned} \mu_i &= \mu^{(i/d)} \\ \sigma_i^2 &= (\sigma^2 = \mu^2)^{(1/d)} - \mu_i^2 \end{aligned}$$

Thus, we are effectively spreading the obstacle's risk distribution over the set of cells it occupies.

However, the Beta distribution parameterized by (μ_i, σ_i^2) is a measurement and not what we insert into the belief map. At each time step t, the belief map will already have a prior distribution $(\mu_t, \sigma/^2_t)$. We use a Kalman filter to integrate the measurement into the belief, which reduces to the Equations 5.27 and 5.28.

$$\mu_t + 1 = \frac{\sigma_t^2 \mu_i + \sigma_i^2 \mu_t}{\sigma_t^2 + \sigma_i^2} \tag{5.27}$$

$$\sigma_{t+1}^2 = \frac{\sigma_t \sigma_i}{\sigma_t = \sigma_i} \tag{5.28}$$

To summarize, we formulate risk assessment in terms of finding the risk distribution over a path, given a risk belief map. The map is gridded into cells, each of which contains a Beta distribution. Paths are sequences of adjacent cells, and we represent their risk distributions as truncated and scaled Gaussians. To update the belief map with scout measurements of an obstacle, we spread the obstacle's distribution over the grid cells it encompasses, and we apply the Kalman filter to update our belief.

Scout Planner

This section describes the algorithm for the Scout Planner component of the ARCAL architecture (Autonomous Robot Control via Autonomy Levels). As described in Sections 5.3 and 5.3.1, the scout's purpose is to obtain more detailed scans of certain areas that could yield safe routes for the logistics vehicle. The scenario is depicted in Figure 5-20. While the logistics executive tasks the scout to examine certain areas, it would be inefficient for the scout to traverse every square mile of its assigned area. For example, a human operator would pilot the scout immediately to the areas that we are most uncertain about and thus stand to gain the most from detailed surveillance. Furthermore, the scout only has limited time to complete its scans and report back to the logistics executive. Our scout planner algorithm addresses these observations and directs the scouts to collect data that optimally reduces risk uncertainty for the logistics vehicle.

Figure 5-22 zooms into the scout planner within the ARCAL architecture and illustrates various components of the scout planner algorithm. The following subsections motivate and walk through these components. Section 5.3.3 begins by defining the scout planning problem in the framework of the Markov decision process (MDP). Using this formalism, we can use well-developed value iteration techniques described in Appendix A to solve for the optimal policy that dictates what path the scout should take. The policy is typically encoded as a value function. However, solving this MDP exactly for a typical scout scenario would require intractable computation, and the value function could only be represented in unreasonable amounts of storage space. Thus, the value iteration technique augments the value iteration process with approximations to yield non-optimal but reasonable solutions. These calculations are performed offline, and the approximate solution is stored in an approximate value function. When the time comes for the scout to execute actions online, it further re-optimizes the value function to its particular situation within the available computation time. This online process is described in Section 5.3.3.

Scout Planning Problem as a Markov Decision Process

The scout planning problem in the context of ARCAL is formulated as follows. The scout dispatcher tells the scout which subset of the full map needs to be examined to have the uncertainty in the risk belief reduced. This subset is represented as a set of grid cells. Each grid cell has a prior risk distribution associated with it. The scout's goal is to fly a path over the area in the allotted time exactly such that it maximizes the total reduction in variance over these grid cells. (The total variance reduction is the sum of all variance reductions in each grid cell.) To solve this problem, we cast it in the general framework of the MDP, which is as follows.

For our scout planning problem, we define the following components of an MDP:

- The state $\mathcal{S}$ includes the vehicle location and pose as well as the belief map (i.e. the risk distributions over the relevant grid cells).
- The action set $\mathcal{A}$ defines how the scout vehicle can move. In our problem, the available actions are *left*, *right*, and *straight* at any grid cell.
- Our problem is deterministic. Thus, the transition probabilities $\mathcal{P}_{ss'}^{a}$ are 1 if and only if the new location, pose, and belief in state s' match those according to the dynamics and Kalman filtering resulting from taking action a in state s. In effect, P is the specification of the problem dynamics.
- The reward $\mathcal{R}_{ss'}^{a}$ is defined to be the total reduction in uncertainty in the relevant grid cells going from state s to state s'. It is calculated by taking the sum over all the reductions in variance resulting from the Kalman filter updating the state from the observations.

It is interesting to note that the state space includes the belief map in addition to the location and pose. This information is a necessary part of the state because the reward in going between states is solely defined by the reduction of variance. It is more valuable to move between cells and see a great decrease in variance because of high initial uncertainty in the area swept over by the sensor than to move between the same cells and see a small decrease because the uncertainty was already low to begin with. Also, including the belief map makes our state space continuous. This will prompt our approximation architecture described in Section 5.3.3.

This section reviews methods for value function representation and approximation. A more thorough treatment was included in Chapter 3 and Appendix C.

We use ADP to find an approximate solution, with the understanding that this exact value iteration is only practical for a small, discrete state space. Because our scenario deals with a large, continuous state space, we will use Algorithm 13 as a building block to the more sophisticated techniques.

Our problem formulation from above assumes a discrete state set S. If the state space is discrete, V and Q can be represented as a table of values, one record for each discrete state. The table of values is initialized arbitrarily and improved iteratively. The problem with this approach is that many real world state spaces are continuous and an acceptable discrete representation is intractably large. In many cases we cannot even store the huge table representation in memory. Additionally, to learn the value function, we must perform a value iteration backup for every single state, which would take far too long.

We address these complexities by representing Q using approximation architecture $(\hat{Q})$. This architecture stores the state-action value function in a compressed form, such as a linear function over problem variables rather than an explicit combination of each possible variable assignment. When inserted in the value iteration process, we get an approximate value iteration procedure which alleviates the storage problem and shares information across state variables, thus decreasing learning time. The approximation architecture, therefore, solves both problems inherent to exact value iteration.

Algorithm 12 outlines the approximate value iteration algorithm where $(\hat{Q})$ is the approximate value function represented by an estimation architecture. Algorithm 12 takes as input the MDP tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where $|\mathcal{S}|$ is very large and possibly infinite. Algorithm 12 begins by randomly initializing state-action value table $\bar{Q}(s, a)$ over state subset

Algorithm 12 Approximate state-action value iteration algorithm where $\hat{Q}$ stands for an approximation architecture representation of the state-action value function, $\bar{Q}$ stands for a lookup table of state action values of a subset $\bar{S}$ of the full statespace $\mathcal{S}$. The key changes from the exact algorithm are highlighted in green.

```
**ApproxStateActionValueIteration**($\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$) $\Leftarrow$ {
for all $s \in \bar{S} \subset S$ do
   for all $a \in A$ do
      Initialize $(\bar{Q}(s, a))$
   end for
end for
InitializeApproximationArchitecture($\bar{Q}_t . \hat{Q}_t$)
$t \Leftarrow 0$
repeat
   $t \Leftarrow t + 1$
   for all $s \in \bar{S} \subset S$ do
      for all $a \in A$ do
         $\bar{Q}(s, a) \Leftarrow \sum_{s' \in S} \mathcal{R}^a_{ss'} \left[ \mathcal{R}^a_{ss'} + \gamma \max_a \hat{Q}(s', a) \right]$
      end for
   end for
   $\hat{Q} \Leftarrow$ UpdateApproximationArchitecture($\bar{Q}, \hat{Q}$)
   for all $s \in \bar{S}$ do
      $\pi_s \Leftarrow \arg\max_a \hat{Q}(s, a)$
      $V_t(s) \Leftarrow \bar{Q}(s, \pi(s))$
   end for
until $\max_{s \in S} \left| \bar{V}_t(s) - \bar{V}_{t-1}(s) \right| < \epsilon, \forall s \in \bar{S}$
return $\hat{Q}_t$
}
```

$\bar{S}$. We then perform a Bellman backup over state subset $\bar{S}$, using $\hat{Q}$ for future state-action value estimates. The newly computed state-action values are stored in table $\bar{Q}$. The policy, approximation architecture and state value table are then updated. This process repeats until a convergence threshold is met. Algorithm 12 returns approximation architecture $\hat{Q}_t$.

Although this approximation yields better storage space and learning time, it may suffer from an inability to adequately represent the state-action value function. We acknowledge this by augmenting the state-action value function so that for every state-action pair, we return a distribution over the value rather than just a single number. Assuming normal distributions, we can write this in terms of a mean and variance:

$$\mathcal{N}\left(\mu_{s,a}, \sigma_{s,a}^2\right) \Leftarrow \bar{Q}(s, a) \tag{5.29}$$

Thus, we incorporate representational uncertainty into the approximate state-action value iteration process so that we learn a distribution estimate over the value. Our offline value function returns these distributions, not just an expectation function. This enables the online algorithm, when deciding between possible actions to execute, to specifically investigate the values of state-action pairs which are not well known *and* are likely to be viable alternatives. Stated another way, we can use an online search algorithm that is guided by the state-action value function. We discuss this next in Section 5.3.3.

Before moving on, we first summarize the offline scout algorithm, which is illustrated in Figure 5-23. The scouts use approximate dynamic programming to create a policy for acting in the world. A policy is a mapping from states to actions, which tells the scouts what to do in any situation. Their state-space includes not only location, pose and risk, but it also encompasses uncertainty in the risk map belief.

Computing a value function can be computationally expensive, so we compute it offline approximately through value iteration before the mission starts. We do so by simulating the scout flying to explore where the greatest reward lies, and we save snapshots of this simulation as data points for our $\bar{Q}$ table. We then generate an approximation architecture on each iteration by regressing over these data points, taking into account the representational uncertainty.

Online Planning and Re-Optimization Algorithms

In this section we highlight the weaknesses of the previous algorithm and explain how it can be augmented with a search algorithm. A more thorough treatment of online search-based policies for MDPs was included in Chapter 2. This review is centered around the problem at hand: disaster relief.

The previous algorithm computes a policy offline. Another approach is to compute an action online, just for the current situation. We actually propose both. We suggest using

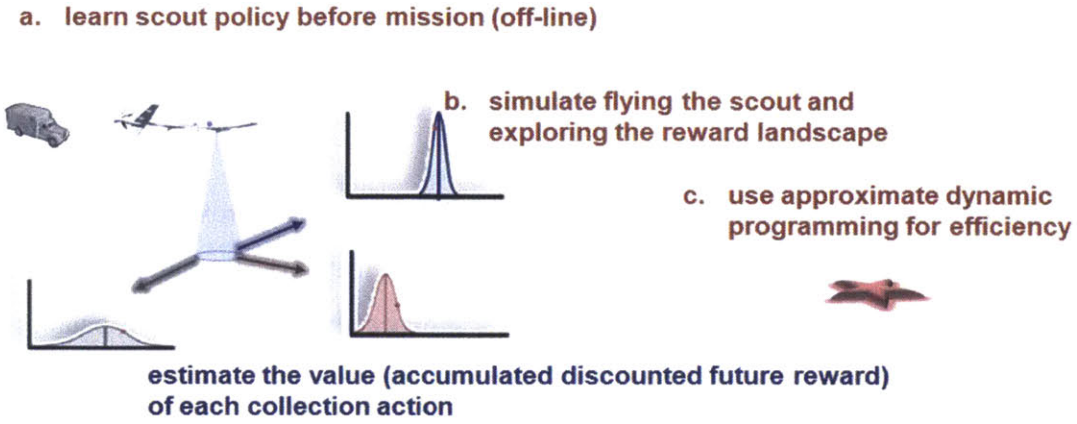

a. learn scout policy before mission (off-line)
b. simulate flying the scout and exploring the reward landscape
c. use approximate dynamic programming for efficiency
estimate the value (accumulated discounted future reward) of each collection action

Figure 5-23: Visualization of Q-function in the Approximation Architecture with Uncertainty

the offline policy as a starting point and then improving it online, tailored to the current situation.

In this section we summarize why we need online planning and re-optimization. In short, re-evaluating actions over a pre-determined horizon helps to reduce the approximation error, which helps us select the best action to ultimately execute. This re-evaluation turns out to be more accurate than the off-line state-action value, because the off-line function is necessarily compressed and therefore not expressive enough to capture every detail. The discussion includes how we use the state-action value function uncertainty to select actions for re-evaluation. In particular, we will describe a family of tree-search algorithms which use the state-action value function to determine which branches of the tree to search.

Section 5.3.3 outlines a method for computing the state-action value function. Computing the state-action value function is tantamount to computing a policy because it tells you how good each currently available action is given your current state or situation. In other words, it tells you how good each action is and you simply have to select and execute the best one. However, the resulting state-action value estimates are not as accurate as they could be. The off-line generated function is necessarily compressed and therefore not expressive enough to capture every detail. Therefore, instead of using it raw, we improve upon the estimates by performing some additional simulations online.

Recall that the state-action value function covers the entire state-space. In other words, it will provide an estimate of the action values emanating from any state or situation we may encounter. This is a challenging request. In contrast, our situation is a far less daunting task. However, since we do it in realtime, we do not have as much time to complete that task. Our approach attempts to make the best trade-off between online and offline computation by computing a state-action value function offline and using it as a starting point for our online algorithm.

The process of online reevaluation involves generating a tree of candidate scout vehicle paths to test (Figure 5-24). We generate the tree of paths with the help of the offline generated state-action value function. Starting from the current state, the algorithm selects b actions to simulate, where b is called the branching factor. After simulating the actions, the algorithm will reach b new states. From each of those new states, the algorithm again selects b actions to simulate. After two steps, the algorithm will reach b^2 new states. This process proceeds to a predetermined planning horizon h. Collectively, the algorithm simulates b^h paths. The algorithm's task is to re-estimate how much each scout collection path will improve our knowledge about the risks to the mission vehicle. We think of this increase in knowledge as a reward in accordance with the MDP framework. Thus, the algorithm adds up the cumulative reward garnered from each path. If the planning horizon is long enough to reach the end of the mission, then the simulated accumulated reward is used as the value of the initial action. If the planning horizon is not long enough to reach the end of the mission, then the remaining value is estimated using the offline state-action value function.

The above algorithm uses branching factor b and horizon$h$. We select b and h such that we have enough time to re-evaluate b^h paths. To do this, we first determine how many computations can be performed in the allotted time between decisions. We then select a planning horizon and calculate a branching factor which will result in that number of calculations.

A central question in this process is how to choose the actions to reevaluate. As stated above, we generate the tree of paths with the help of the offline generated state-action value function and we select b and h such that we have enough time to re-evaluate b^h paths. One assumption in this process is that we do not have enough time to reevaluate every action over the planning horizon. Therefore, we elect to evaluate actions that have a promising outcome, with consideration for how sure we are about our estimates of those outcomes.

At each step in the algorithm described above, we select b actions to simulate. This selection uses the offline state-action value function. The function provides an estimate of the value (future cumulative reward) of each action, if taken from the current state. The estimate actually includes a distribution described by a mean and variance. The distribution captures how well we know a given value. A high variance distribution means that we do not know that value very well. Likewise, a low variance distribution means that we know that value rather precisely.

We select a sample from these distributions, one for each action, and then choose the action with the highest sample value. If one of the action distributions consistently produces a high sample value, this tells us that we have little reason to evaluate other options.

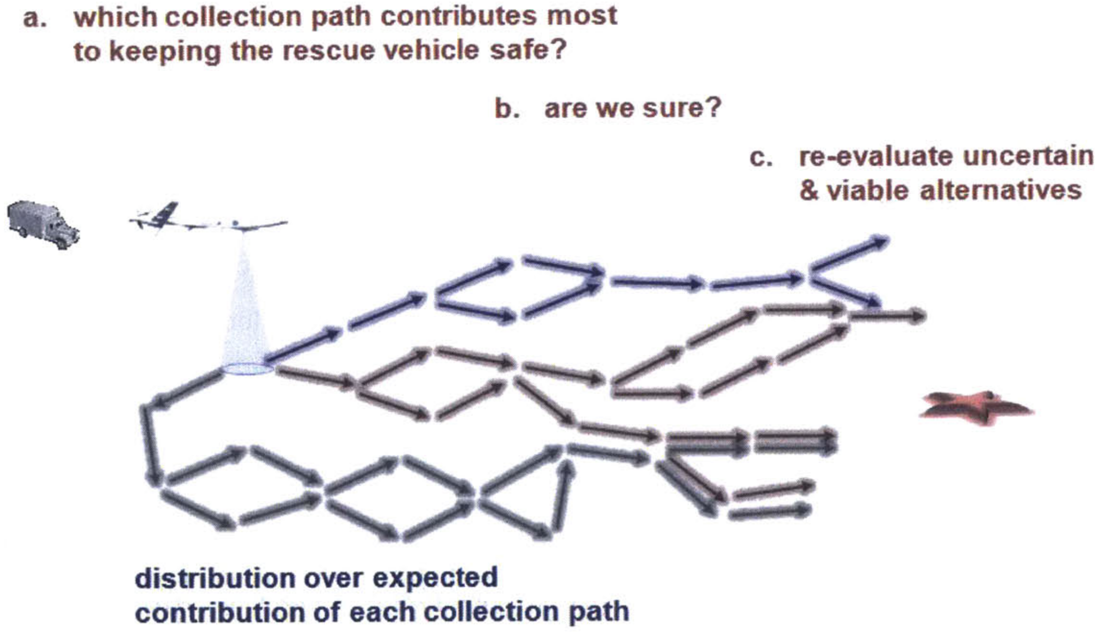

a. which collection path contributes most to keeping the rescue vehicle safe?
b. are we sure?
c. re-evaluate uncertain & viable alternatives
distribution over expected contribution of each collection path

Figure 5-24: Process of online reevaluation involves generating a tree of candidate scout vehicle paths to test.

However, if there is a state-action value distribution with an especially high variance, it will sometimes produce a sample with the highest value even though its mean is lower. This phenomenon exactly mirrors the probability that said action is the best, given what we know. In other words, we explore the actions in proportion to how good they are and how certain we are about that. In summary, we determine which actions to re-evaluate by representing the uncertainty about their true value. This distribution is used to select actions (Figure 5-24) for re-evaluation when they appear to be good but the truth is uncertain. We may also evaluate poorer-looking plans when the true value has such high variance that it warrants a second look.

To summarize, our method allows us to use offline knowledge and processing to guide our scouts online. The way we construct the offline policy informs the additional online processing. In this way, we can exploit both on and offline control processing in a complimentary way. We can compare this to how people deal with a planning task. Suppose you are navigating home from work. You have a policy based on experience, which tells you which road to take from a given intersection. However, given the time of day you may wish to consider alternatives. For example, given the traffic, you may be unsure if your standard policy is optimal. Therefore, you can simulate in your head the consequences of some alternative routes. The result of the simulation tells you which way to go (what action to take) right now. Your mental simulations will surely be biased toward perceived viable alternatives. In other words, you will consider some alternatives and others you will not. In this way, your offline knowledge informs your online situational reevaluation.

The above process actually describes a family of algorithms, because changing our search horizon and branching factor fundamentally changes the algorithm. For example, if we use a very small branching factor with a long horizon, the algorithm closely resembles the rollout algorithm. Rollout is a longstanding method of evaluating moves in the game of backgammon (Tesauro [1995]). On the other hand, using a short horizon with a large branching factor closely resembles model predictive control. Different configurations will work better for different applications,as well as at different points in the mission. For example, toward the end of a mission, it could be helpful to use a wider search (larger branching factor) to make sure that we rightfully consider the end goal. Further extensions to our algorithm may include using different branching factors at different levels of the search tree. For example, it is easier for a function to capture long term objectives then short term details. Therefore, this small extension would allow the algorithm to rely more on offline knowledge in the middle of the mission and rely more on simulation for starting and ending the mission. All in all, our method describes a broad family of algorithms via a set of configuration parameters. Describing the algorithm in this way enables us to tune it, making it broadly applicable to many applications.

5.3.4 Algorithm Demonstration Results

To show how our scout finds efficient sensing paths, we center our discussion on four example scenarios. Each scenario tasks the scout to reduce variance within a 10×10 area, but the areas of high variance differ across scenarios. Scenario A has high variance in the southwest quarter patch and zero variance (i.e. perfect knowledge) everywhere else. This corresponds to a very large feature in one corner of the map which we have no information about, and everywhere else we have perfect information already. In scenario B, we split the uncertain feature into two smaller features, one in the southwest corner and the other in the northeast. Scenario C involves a feature in the middle that we know a great deal about (i.e. low variance) but we grow more uncertain the farther we go toward the edges. Finally, scenario D randomly smears high variance across the entire map, simulating a realistic setting where we may have poor prior information of a certain region.

Each scenario was run with a mission length of 25 time steps. Within each scenario's context, we first illustrate the evolution and convergence of the approximate value function during value iteration. We use 100 samples to represent the state subset. Note that there are 100 unique grid cells the scout can be in, four possible orientations in each, and an infinite number of map beliefs possible. Thus, our value function representation is extremely sparse relative to the actual state space. We then display the paths constructed during online execution to show how the scout chooses to survey areas with higher uncertainty within the time allotted. The tree search algorithm is limited to 20 node traversals of computation, but searches down to a depth of seven nodes.

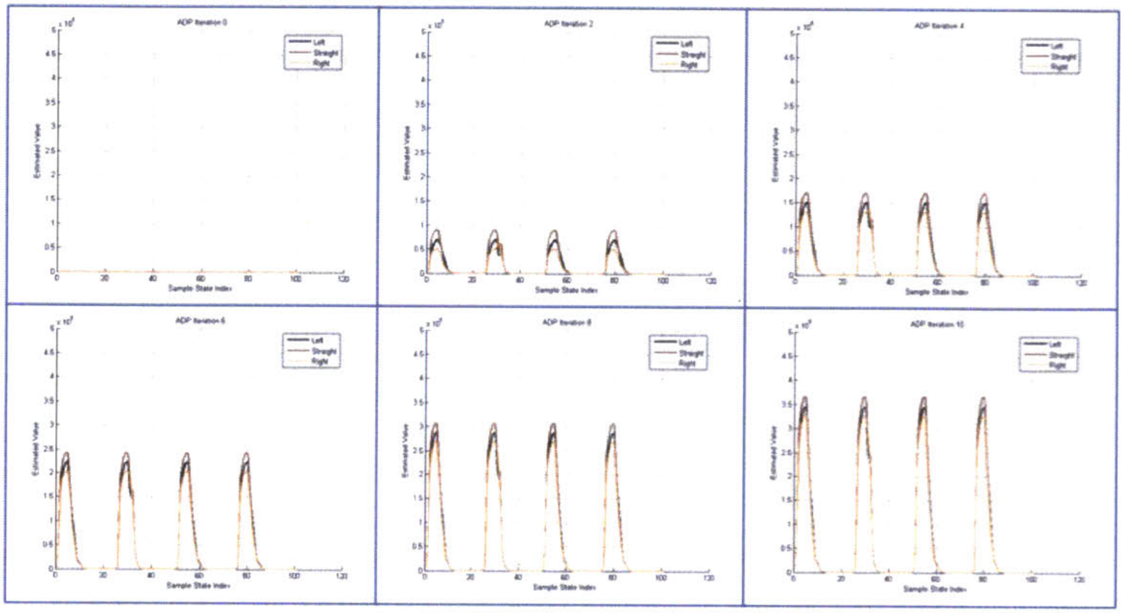


Figure 5-25: Value Function Representation for Scenario A

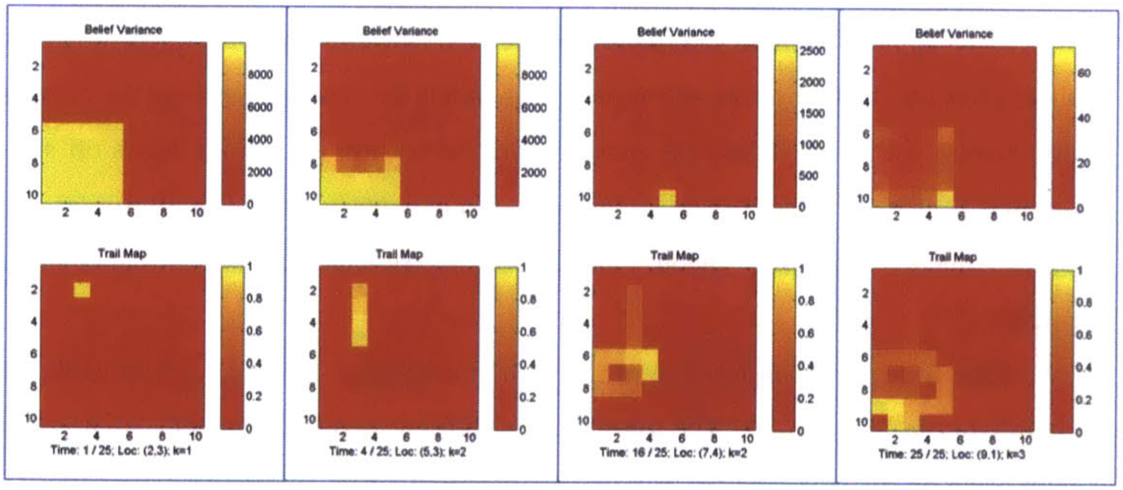


Figure 5-26: Belief Variance and Scout Path for Scenario A

Scenario A

Figure 5-25 depicts the state-action value table $\bar{Q}$ evolving over 10 sets of value iteration. The x-axis represents different sample states in our lookup table, and the y-axis shows the value associated with that state. When querying the value of a state, we are not querying $\bar{Q}$, but rather $\hat{Q}$ representing the estimation architecture which interpolates over the sample states in the $\bar{Q}$ table. However, to aid conceptual convenience and transparency, we will refer to the $\bar{Q}$ plots as the value function plots. The sample states were constructed by initializing a simulated scenario four times and letting the scout fly a pre-determined lawnmower pattern which sweeps across the area for the length of the mission, 25 time steps. To avoid gathering the exact same data each time, stochasticity was introduced into the path, more with each subsequent pass.

The resulting representation of $\bar{Q}$ on the even iterations are shown in sequence on value plots in Figure 5-25. At first, the values for each sample state are initialized with low random noise, which is not visible at the scale shown. In subsequent iterations, the values accrue at each step, since each state "looks ahead" to the next best state and adds that state's value to its own reward (i.e. variance reduction) for taking the action leading into that state. The values gradually converge (i.e. the increase at each step gets smaller) since the accruement is increasingly discounted over later iterations. This is consistent with contraction mapping Equation C.3 in the Appendix A. However, the most interesting parts of these plots are the four peaks, corresponding to when the scout passes over the patch of high variance and hence gathers the most reward. Afterward, the reward tapers off since the scout's sensor coverage is overlapping areas just previously seen. This illustrates how the value function effectively encodes and exploits the structure of belief variance in the scenario.

We have shown the construction of the path at time-steps 1, 4, 16, and 25 of the algorithm's run in Figure 5-26. The top plot in each frame shows the belief variance with a color scale on the side, and the bottom plot the scout's path so far. As can been seen, the scout takes the reasonable action to plunge south into the area of highest uncertainty and then loops through it until the end of the mission. Note how the scout "carves away"

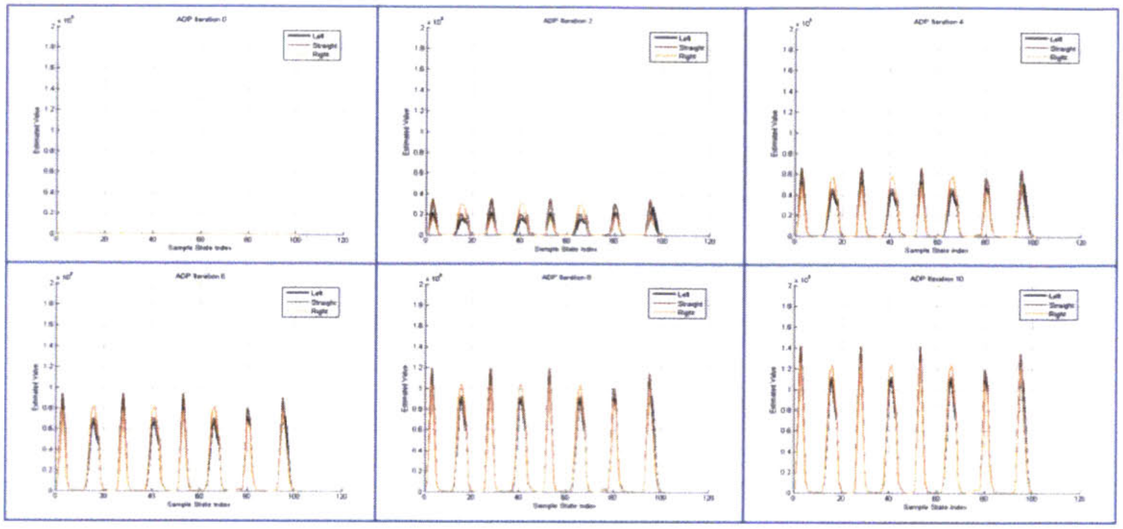


Figure 5-27: Value Function Representation for Scenario B

at the belief variance in the top plots. We rescale the colors so that areas with the highest remaining variance always appear yellow, and hence one can see how they guide and attract the scout. However, one should note how much the scale has changed by the end of the mission, thus showing the extent of variance reduction.

Scenario B

Figure 5-27 shows the value function for two uncertain features in this scenario. For each of the four runs, the value function peaks in two places, corresponding to the two features. However, the first peak in each of the first three runs is slightly taller than the second peak. This is due to the width of the sensor footprint. When it is primarily sensing the first feature, it also captures a little bit of the second, given their proximity. Thus, when it actually flies over the second, it will not generate as much variance reduction as the first. Since there are only these two features, the effects of stochasticity in generating the sample states are not very pronounced. However, as the stochasticity is turned up with each pass, we notice the peaks for the last pass look substantially different than the first three. This indicates we have allowed ourselves to learn about the features from a slightly different perspective, which gives us a richer representation.

Now we turn to the scout's path, shown in Figure 5-28. The scout begins by heading

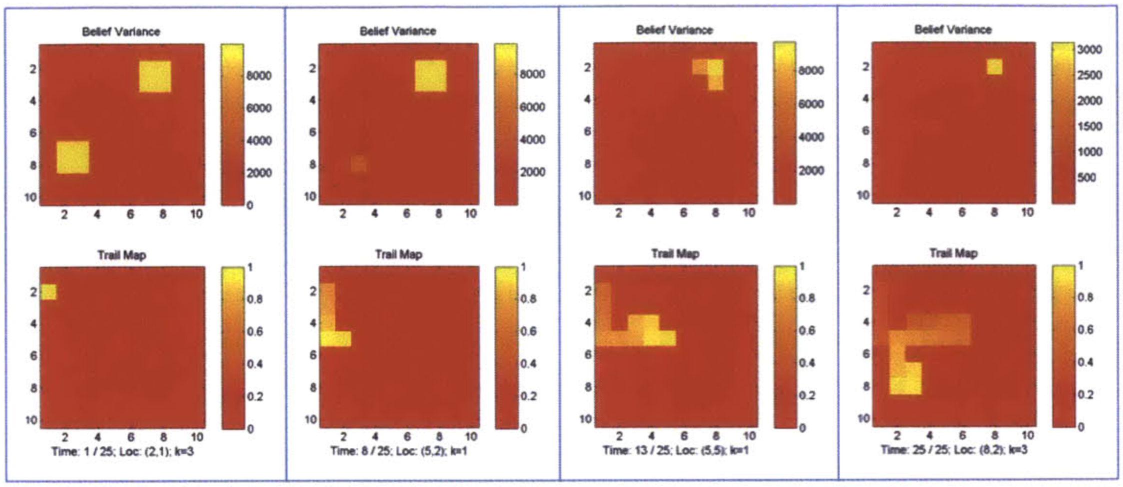


Figure 5-28: Belief Variance and Scout Path for Scenario B

to the feature in the southwest corner. Not only is it slightly closer to the scout's initial position, but the value function also suggests it is of higher value than the other features. This is an artifact of our approximation architecture, but it is a small price to pay and easily mitigated on average with stochasticity introduced during the online algorithm. Yet, the scout does not go all the way to the bottom feature because as it approaches, its sensor footprint already provides good coverage of the feature. Thus, two steps away, it turns toward the northeast feature.

Similarly, it does not need to head all the way east. About halfway through the mission, it turns back again to better understand the first feature, rather than spend all its remaining time around the second feature and not reducing overall variance as much. This kind of behavior illustrates the value of the online search algorithm, for knowing when to turn back cannot be encoded in the value function itself. The look-ahead feature of search allows the scout to plan based on how far ahead its mission end is. Here we used a search depth of seven. Had we searched even deeper, we may have arrived at the more efficient solution of spending half the time at each feature and not have to turn back. However, searching deeper costs time. If we are to maintain the same online computational time, then searching too deep would prune the branching factor and thus degrade the quality of search.

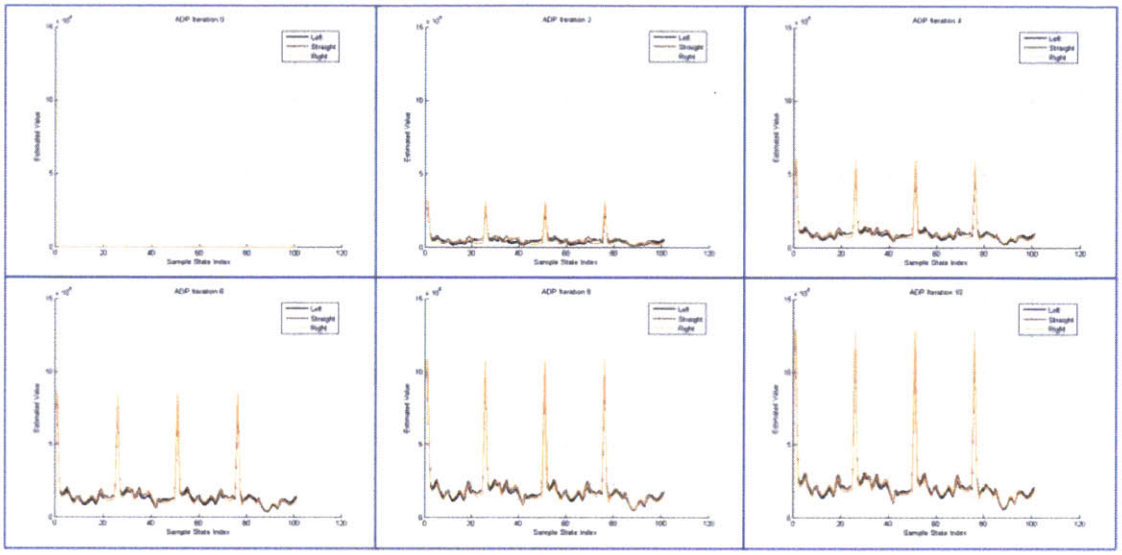


Figure 5-29: Value Function Representation for Scenario C

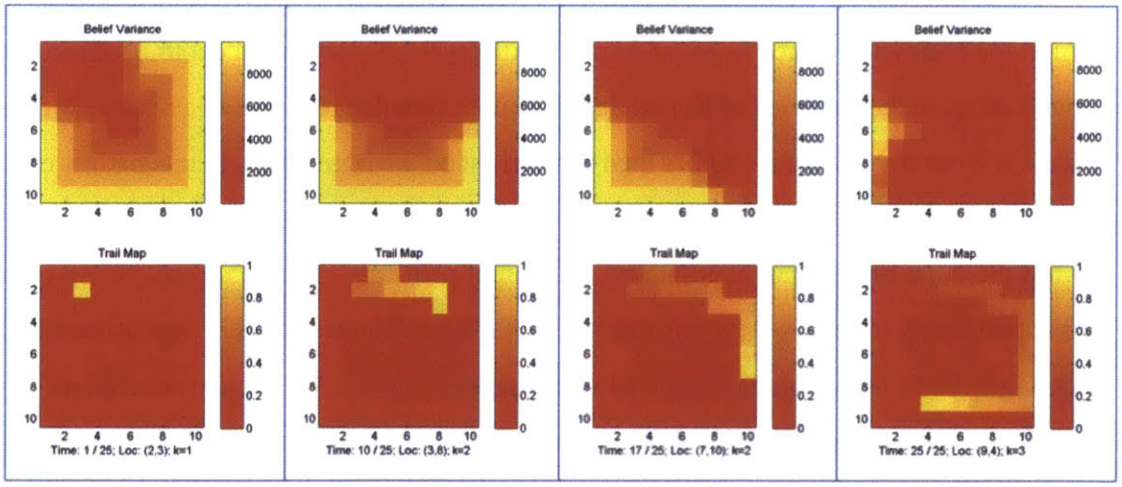


Figure 5-30: Belief Variance and Scout Path for Scenario C

Scenario C

In this scenario, we have the highest variance on the boundaries, and it decreases as we approach the center. Since we are flying a lawnmower pattern over the area, we notice periodic peaks as we approach the edges in the value function plots shown in Figure 5-29. As we make sweeps over the central area where variance is lower, we notice a decreasing trend in the values, but it picks back up as we start to swing closer to the opposite edge. The high initial peak at each run is due to the scout sensing completely new terrain; it corresponds to states where the scout is in the top-left corner and the belief map variance is high throughout the region covered by the sensor footprint. Since there is much more total variance spread across the region in this scenario than in A and B, the effects of stochasticity are much more clearly seen. While the general trend for the values across each run is similar, the details are quite different. This captures a much richer and more informative representation of the value function.

The reasonable thing for the scout in this scenario is to hug the edges where the variance is highest. This is exactly what Figure 5-30 shows. However, it does not just hug the edges statically, but it dynamically tries to go inside a bit, too, where valuable information also exists. Given the time it has, the scout tries to balance how much time it spends closer to the outside and inside. In 25 time-steps, it is infeasible for the scout to make a full circuit around the perimeter, which is 36 grid cells. With an online look-head depth of seven, the scout realizes in the south leg of its path that it is more valuable to maintain a distance of one cell away from the south edge rather than travel right on the edge. Also, in this path, we can notice stochasticity in the online process. On the third step of its path, the scout turns north toward the edge rather than continue on its present path east. This is clearly suboptimal, and it could only have arisen by "mistake." The "mistake" arises from the representational uncertainty in the value function approximation architecture $\hat{Q}$. We encode uncertainty into the value function because we know it is not perfect. However, this means each query to the value function is a probabilistic draw from a distribution characterized by this uncertainty. Thus, we have to submit to the possibility of a clearly suboptimal decision every once in a while. As the path shows, though, these mistakes are uncommon and quickly recovered

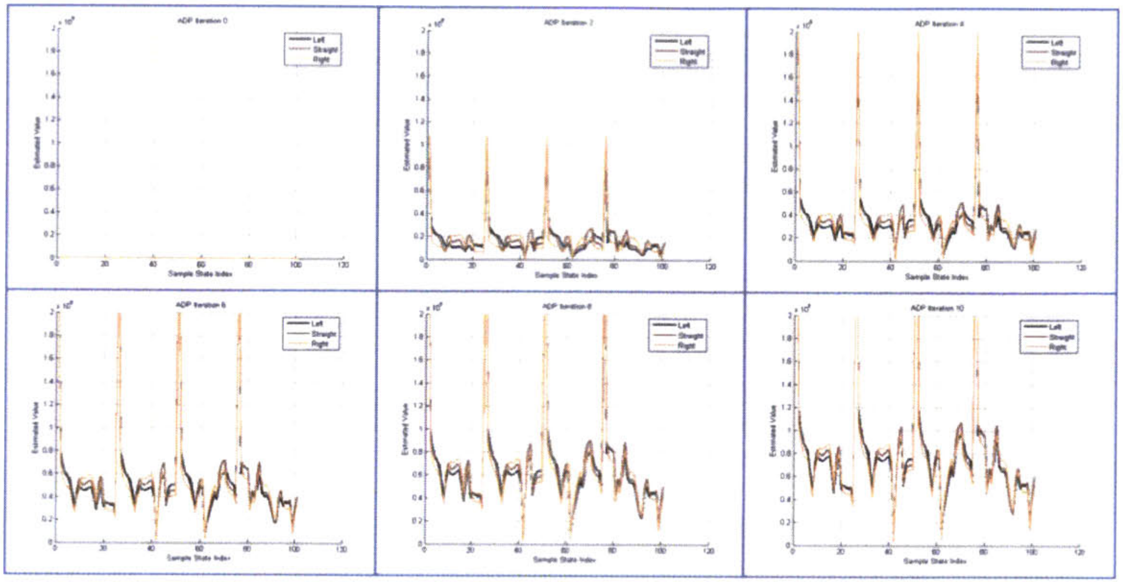


Figure 5-31: Value Function Representation for Scenario D

from.

5.4 Scenario D

When we have random high variance everywhere in the graph, the corresponding value function shown in Figure 5-31 is messy and difficult to interpret by hand. It bears some resemblance to Scenario C's value function, since there is value everywhere in the region. However, it is substantially less structured. Nevertheless, the value function still encodes the random variations in this map as well as the larger structure, and this will be apparent in the path the scout chooses.

Similar to the plots for Scenario C, Figure 5-32 shows that the scout chooses to fly a circuit around the region to provide greatest coverage. As before, the scout does not have enough time to fly cover every piece of high uncertainty, and thus the scale at the end is much larger than the scale for Scenarios A and B. However, the path leaves less uncertainty uncovered than in Scenario C because the scout can afford to hug the outer edges less and thus has less distance to travel to make a circuit. Note that the scout does not even have to complete the circuit or else it would overlap with its initial sensor footprint. It is worth noting that in running multiple trials for each of these scenarios, Scenario D showed the greatest variation in the planned path. That is, while the paths for the other

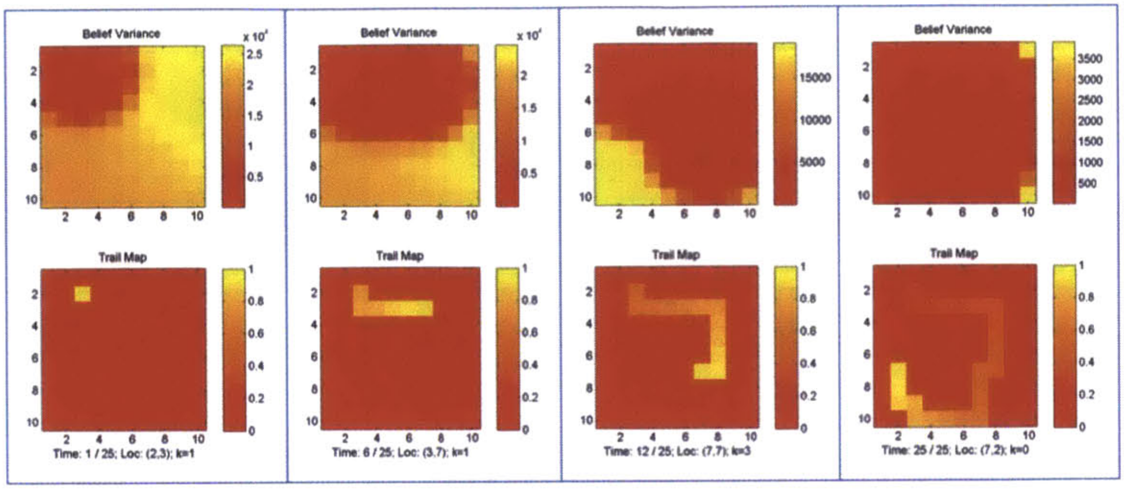


Figure 5-32: Belief Variance and Scout Path for Scenario D

scenarios followed predictable and expected patterns, sans the occasional "mistake," the paths for Scenario D may choose to wander in toward the center or wander out at any given point. This suggests there is larger representational uncertainty in the value function, which is to be expected in a less structured environment. It means multiple good solutions may exist, and since our representation models this aspect, we do not limit ourselves to a single deterministic path in any scenario.

Algorithm Summary

These examples show that our scout planning algorithm is capable of finding a path through an area which purposefully surveys the most uncertain features, thus generating the most valuable data for the logistics planner. This is accomplished through a combination of the offline value iteration and online search procedures discussed in Section 5.3.3. The flexibility of our algorithm arises from the Markov Decision Process framework, which easily adapts to any given scenario. An important detail of our approach is that since we cannot exactly represent the value function, we acknowledge it by introducing stochasticity into our decision-making. Thus, our non-deterministic solutions, while rarely optimal, are more robust to this uncertainty. When integrated with the entire ARCAL system, the scouts shall be valuable additions to the logistics vehicle.

5.3.5 Impact

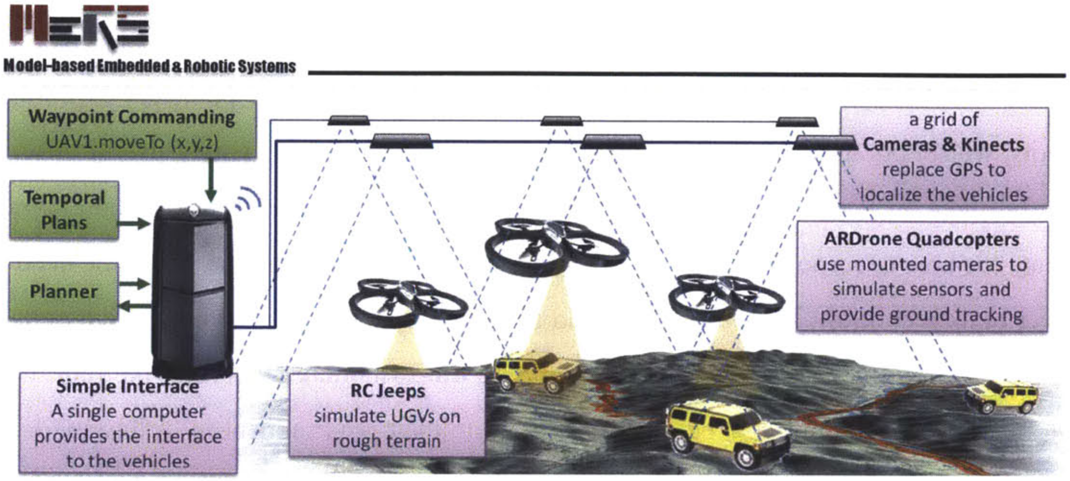

MERS
Model-based Embedded & Robotic Systems
Waypoint Commanding
UAV1.moveTo (x,y,z)
Temporal Plans
Planner
Simple Interface
A single computer provides the interface to the vehicles
RC Jeeps
simulate UGVs on rough terrain
a grid of
Cameras & Kinects
replace GPS to localize the vehicles
ARDrone Quadcopters
use mounted cameras to simulate sensors and provide ground tracking

Figure 5-33: Heterogeneous Robotics Test Bed

The ARCAL project proposes two complimentary areas of research which, when combined, can increase operator confidence in autonomous system behaviors. The first incorporates concepts of risk-based adjustable autonomy with risk verification within system functions and task-directed adaptive search techniques. The second involves new methods to effectively test and evaluate collaborative autonomous team behaviors prior to field deployment.

Task-directed search algorithms for UAV scout path-planning, used to improve knowledge of the risks to the mission, are developed and implemented. To develop and validate the concepts and technologies used in this project a natural disaster recovery scenario has been employed as an example. A team of UAVs is dispatched to determine the safest and fastest path for a disaster recovery convoy to deliver relief supplies. An algorithm test battery was developed and used to run tests on the scout path-planning algorithms. Test results from a number of runs with the algorithm applied to terrain examples with various belief uncertainty concentrations are shown and described. Initial results show that the algorithm performs quite effectively in these situations. Although a natural disaster recovery scenario is used as an example, the autonomous algorithms, concepts and technologies being developed can certainly be used for other scenarios with the UAV Simulation Environment.

6 Summary

Autonomous Science Motivated by the vision of autonomous interplanetary exploration robots, we set out to make exploration systems that can go out and gather useful information on their own, at a global or planetary scale. This concept require that autonomous agents are able to respond in real time to situations and opportunities, without the need for direct human control. These agents should have the capacity to determine which data are most relevant to their missions, such as mapping life-sustaining resources like fuel, water and metal. They should also be able to respond to their environment by following leads, rechecking indeterminate results that are potentially interesting, managing the discerning power of their sensors, balancing the costs and benefits of detecting or negating the presence of a phenomena, and balancing the benefit of a potential find with the opportunity cost of locating it.

In Chapter 2, we developed an architecture that allows us to perform information gathering at a global scale through a 3-level hierarchy of information gathering agents, each of which is performing adaptive sampling online. This hierarchy allows us to perform information gathering effectively at each level, framing active sensing as an info-MDP and solving it using approximate dynamic programming. In Chapter 3, we showed that we can solve an info-MDP using ADP, producing great science return, by using decision uncertainty minimization planning. Representation uncertainty minimization, which was introduced in Chapter 4, supports DUM by building an architecture for instantiating the approximate state-action value function. In Chapter 5, we demonstrated each level of the hierarchical architecture on an application.

Effective sensing plans computed in real-time via blending on- and offline computation

Having a method for blending on- and offline computation means that we can compute effective sensing plans in real time. As a result, we can also re-compute those plans, thereby adapting to uncertain outcomes in the real world. Thus, our Monte Carlo information guides-based algorithm enables active sensing. Our experimental evidence showed that DUM is more efficient than other algorithms used in active sensing, and delivers greater science returns.

This algorithm is also important to the broader adaptive sampling community. Specifically, we have demonstrated that information guides can be used to optimize adaptive sampling. Yet, a Monte Carlo information guide planner is an adaptive sampling algorithm. Monte Carlo planning with information guides allows us to choose selectively where to simulate, in order to optimize our decisions. As the selective simulation is an evolving information gathering task—i.e. adaptive sampling—this thesis uses an adaptive sampling based search technique to optimize adaptive sampling problems.

While these ideas are powerful and useful to a wide community, they also have limitations. As currently implemented, the methods use a Gaussian model to predict or describe the uncertainty in outcome. However, the true shape of this uncertainty may not be Gaussian. In some cases, it appears to be very non-Gaussian. Therefore, exploring other ways of modeling the outcome could improve the performance of the algorithm. One option would be to use a mixture of Gaussians to model the outcome.

An architecture for active sensing

In this thesis, we developed and used an architecture for active sensing that has three levels. This architecture closely mirrors many real world information gathering problems. These problems range from scientific sensing of the oceans to logistical support for disaster relief. They all use multi-level, multi-resolution platforms, each of which has a different concomitant field of view. These earthly applications closely mirror their planetary exploration counterparts. For example, planetary exploration requires directed and optimized data collection for scientific inquiry, as well as data collection for logistical support.

The architecture is useful to a larger community as well. A variety of science and logistical support missions could also benefit from such an architecture. It is also useful for subsystem cueing within a self-contained system. For example, a wide field camera could look for movement, while a narrow field telescope focused in on movement or other interest indicators.

DUM may have broader benefit to symbolic, hierarchical and factored Markov decision problems. One possible future algorithm would use this architecture to optimize a hierarchy of variable duration actions. Consider a UAV being deployed to several mapping locations. In the current framework, the lower level mapping tasks are all of the same specific duration and a value function estimates the value of these actions. However, it is quite reasonable to want to spend more time at some mapping locations than others. A value distribution could be learned, which estimates the informational value of collecting data at a mapping location over a range of time intervals. By doing so, the agent would have available the decision making information needed to trade-off how much time to spend at each location, rather than only being able to select tasks of the same duration. Likewise, if the remaining value of a task has petered out prematurely, the task may be aborted in lieu of starting a different task. This added flexibility may greatly increase what the overall mission can accomplish.

Take aways: The two main take aways from this thesis are a method for blending on- and offline computation and an architecture for active sensing, both of which improve the ability of autonomous agents to collect high-quality information in applications ranging from ocean sensing to disaster relief to planetary exploration.

Appendices

A Marine bathymetric mapping

We collaborated with the Monterey Bay aquarium research institute to develop a depth control algorithm for the Dorado class autonomous underwater vehicle, which conducts scientific underwater missions such as mapping the sea floor (shown in Figure A-1). Our spliced linear programming depth control algorithm has been successfully deployed at a topographically difficult region three miles off shore in Monterey Bay (see Figures A-7 and A-8). The algorithm improves the quality of scientific measurements by allowing the underwater vehicle to approach closer to the sea floor as shown in Figure A-9. Figure shows that our algorithm is more successful than the previously existing algorithm at maintaining the proper altitude above the sea floor.

A.1 Opportunities

Autonomous bathymetric mapping (Monterey Bay Aquarium Research Institute [2006]), demonstrated the feasibility of and opportunities for enhancing planning activities. This project utilized a Dorado class AUV operated by the Monterey Bay Aquarium Research Institute (MBARI) that conducts bathymetric mapping of Monterey Bay (Fig. A-1). The AUV collects bathymetric data using a multibeam sidescan sonar sensor and navigates using a gyro-based inertial navigation system integrated with a Doppler velocity sonar. The AUV deploys from a ship, descends in a spiral path and navigates along the ocean floor using GPS.

As it approaches the ocean floor, the AUV tracks its position via multiple navigation sonars, one of which monitors the relative location of the ocean floor for accurate positioning. This sensor, the Doppler velocity sonar, must maintain ground-lock in order to provide reliable velocity estimates. When ground-lock is lost, the AUV defaults to a less accurate inertial navigation system. Because data quality is strongly influenced by localization, accurate bathymetric maps depend on maintaining ground-lock. The AUV attempts to remain at a target depth of 50 meters above the ocean floor, where ground-lock can be reliably maintained.

Aquadynamic perturbations often prevent the AUV controller from accurately navigating to its target depth. The AUV must also pull up prematurely when approaching a steep incline. The current depth planner projects 100 yards ahead and plots a course straight toward the highest point on the horizon. This method often loses ground lock in steep terrain such as the Davidson Seamount. We collaborated with the Monterey Bay aquarium research Institute to develop a better depth control algorithm for the Dorado AUVs dynamics controller. The algorithm, referred to as a spliced linear programming algorithm, helped the vehicle maintain ground-lock in variable terrain and thus improved its mapping capabilities.

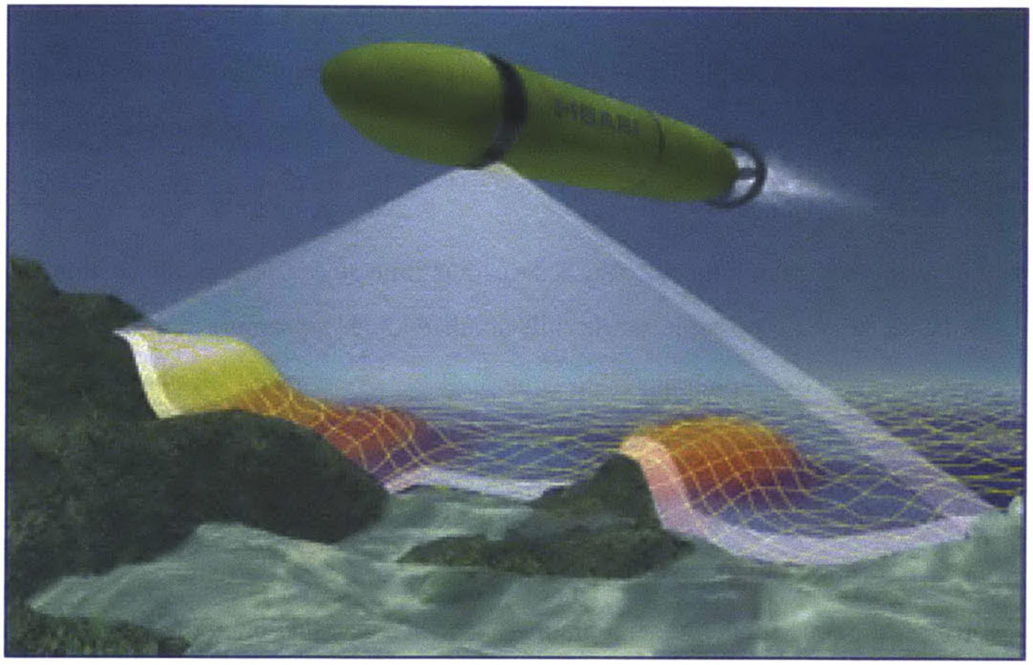


Figure A-1: We collaborated with the Monterey Bay aquarium research institute to develop a depth control algorithm for the Dorado class autonomous underwater vehicle, which conducts scientific underwater missions such as mapping the sea floor.

A.2 Mission planning algorithm

We created a prototype dynamics-based optimized depth waypoint planner for the MBARI AUV. This planner takes: 1) a linearized model of the AUV dynamics, 2) constraints on the AUVs performance, such as pitch angle and elevator angle constraints, 3) a profile of the sea floor along a transect, 4) and minimum, maximum and optimal depth specifications relative to the sea floor. The planner provides a series of depth waypoints along the transect that minimize the average positional deviation from the AUVs optimal depth, while adhering to hard constraints such as pitch angle. In addition to penalizing optimal depth deviation, we also penalize average pitch rate magnitude, which dampens the control inputs. Without the pitch rate penalty, the optimal planner will use quickly changing (chattering) control inputs to minimize depth error. The pitch rate penalty is a design parameter, which depends on the relative importance of a smooth control plan.

Planning is carried out with respect to discrete-time AUV dynamics. A full plan takes place over several hours, whereas the time constants of the AUV dynamics are on the order of seconds. Generating a full depth waypoint plan therefore involves optimization over a very large number of decision variables. The following specifications make this problem tractable:

1. For waypoint planning, we use a 5-second time interval. We chose this interval to be as large as possible while still small enough to capture the AUV dynamics adequately. After a plan has been generated, a more detailed simulation with a finer time resolution is carried out to validate the plan.

2. We plan depth waypoints for 1,000-second segments and splice the solutions together to give a full plan. The splicing ensures a smooth transition between segments, and ensures that one segment is not planned in such a way that planning the next segment is infeasible.

A.3 Simulated planning results

Figures A-2 through A-6 show simulated test results from the Davidson 2007 Survey. We developed, simulated and tested our algorithm using the Davidson Survey data. Figure B-2 shows the location of the survey, off the coast of California near Big Sur.

We ran our spliced linear programming algorithm on the Davidson Seamount data. The data was first resampled at 5 second intervals. We then ran our linear programming

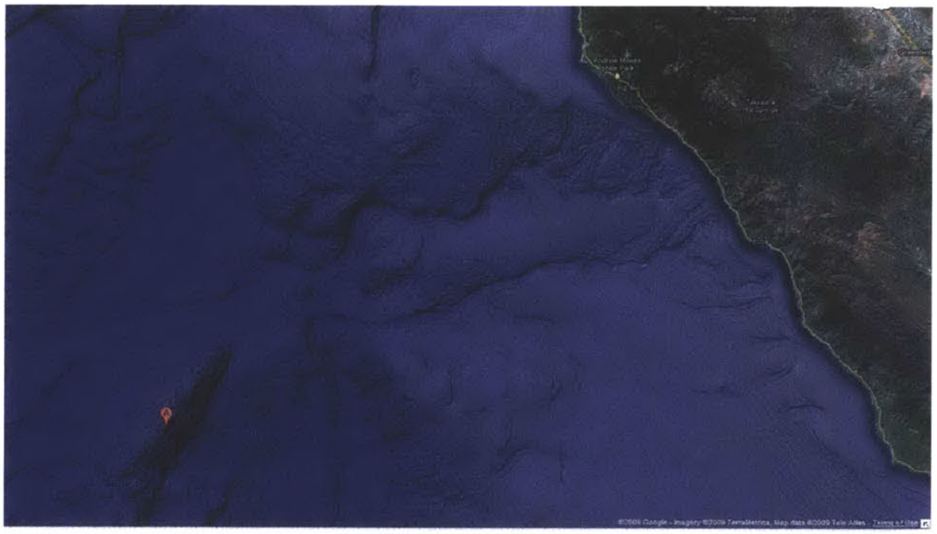


Figure A-2: The red marker shows the location of the 2007 Davidson Seamount survey. Davidson Seamount lies west off the coast of California near Big Sur. We used the survey data to develop and test our algorithm in simulation.

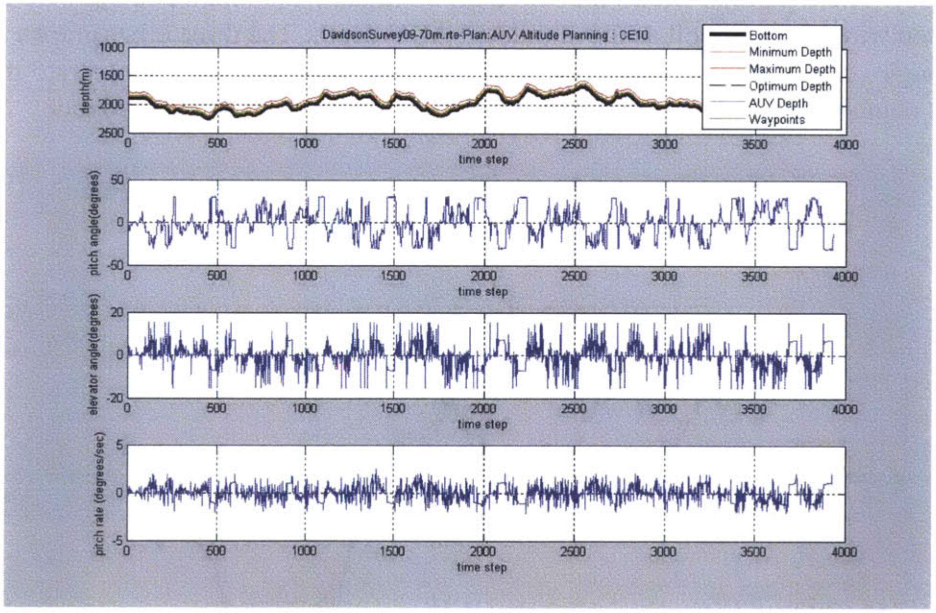


Figure A-3: We ran our spliced linear programming algorithm on Davidson Seamount data, resampled at 5-second intervals. The solver produces a set of target waypoints for the controller to follow. The top graph shows target waypoints, sea floor depth, minimum, maximum and optimal depth as well as simulated AUV depth. The three lower graphs show the pitch angle, elevator angle and pitch rate respectively.

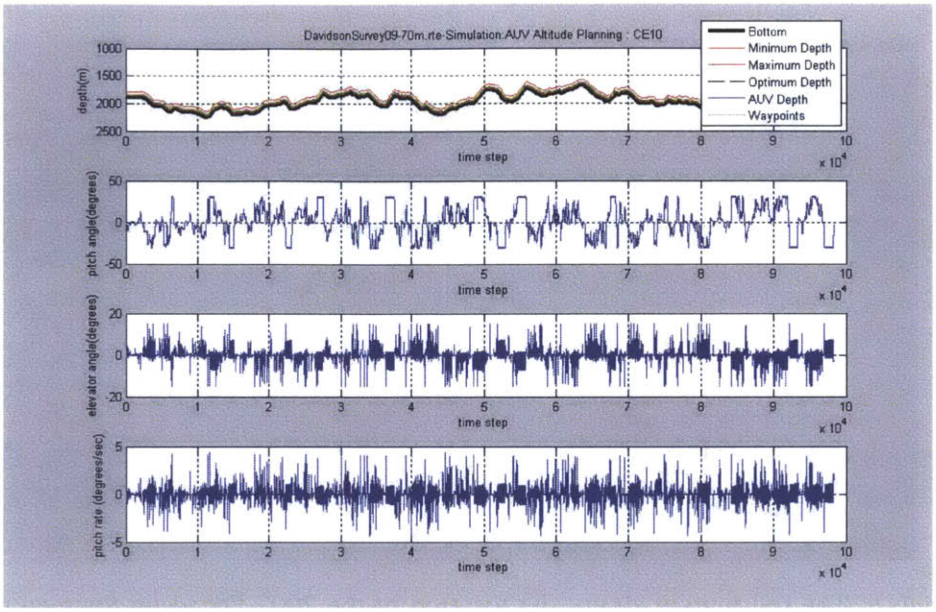


Figure A-4: Simulation results of the entire transect using the AUVs actual, shorter (.2 second) dynamics controller. The dynamics controller provides input to the articulated ring-wing and ducted thruster actuators. The simulation accepts the 5 second interval target waypoints for the controller to follow. The inputs are duplicated to accommodate the higher frequency. The top graph shows the sea floor, minimum, maximum and optimal depth, the outputted waypoints as well as the simulated AUV depth. The three other graphs show the pitch angle, elevator angle and pitch rate. The simulation results demonstrate that our 5 second planning interval is sufficiently short to produce good vehicle guidance.

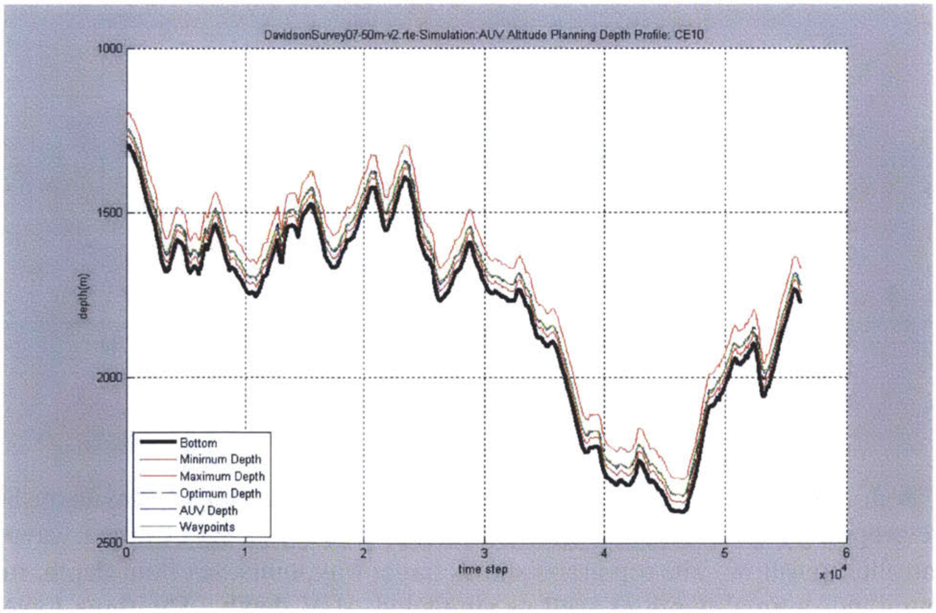


Figure A-5: Larger-scale simulation results. The graph shows the sea floor, minimum, maximum and optimal depth, as well as the simulated waypoints and AUV depth.

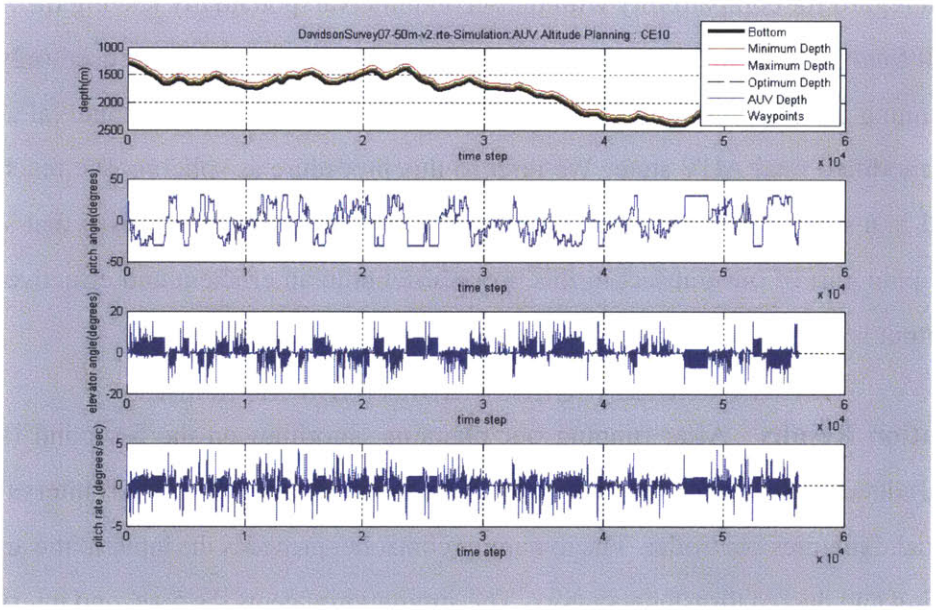


Figure A-6: The graph zooms in on a small portion of the simulation results in order to show algorithm performance along a difficult topographic inflection. The red lines show the minimum and maxium depths. The dotted line shows the target depth. The blue line shows the simulated AUV depth. At the difficult inflection point, our algorithm induces the AUV to change altitude in anticipation of the topography, so that it does not grossly overshoot the target depth while still observing the mini-mum depth constraint.

solver on the first 1,000-second interval. The solver takes as input the initial AUV state, the desired depth, depth constraints, linearized vehicle dynamics and the respective vehicle dynamics controller. The solver produces a set of target waypoints for the controller to follow.

Splicing Method Once we solve for the first interval, we used the final AUV state as the initial AUV state for the next segment. Unfortunately, the first interval was optimized without regard for compatibility with the second interval, potentially leaving the AUV in a difficult final state. In order to accommodate the subsequent interval we used only 75% of the planning interval, discarding the latter 25% and beginning the next interval according to this modified final AUV state. We refer to this procedure as splicing. We reach end the interval in a safe state because we have planned through the potential risks that state. We also re-plan part of our transect in this procedure but in an efficient and effective manner. The output is shown in Figure A-3.

Simulation Results After running our planning algorithm on the 5-second (sampling interval) data, we simulated the entire transect using the shorter (.2 second) interval used by the actual dynamics controller. The dynamics controller provides the input to the articulated ring-wing and ducted thruster actuators. The simulation accepts the 5-second interval target waypoints for the controller to follow. The inputs are duplicated to accommodate the higher frequency. The simulated results are shown in Figure A-4. The AUV initially follows a path between the minimum and maximum altitude, but then navigates along the optimal depth, except when pitch angle constraints make this impossible. Overall, the simulation results demonstrate that our 5-second planning planning interval is sufficiently short to produce good vehicle guidance and the algorithm generates a path that makes full use of the pitch angle allowance in order to minimize the depth error. Figure B-5 shows the main simulation results in a separate larger format, with the sea floor, minimum, maximum and optimal depth, the outputted waypoints, and the simulated AUV depth. Figure A-6 zooms enlarges a segment of the simulation results at a steep topographic inflection. At this feature, our algorithm guides the AUV to change its depth in anticipation of the topography, so as to avoid grossly overshooting the target depth while maintaining its minimum depth constraint.

A.4 Field Test Results

After performing the simulation experiments, we then conducted a field test using our algorithm to plan the waypoint based control inputs for the AUV mission and comparing it to results from the cruder algorithm for the same area. The test was conducted at a topographically difficult region three miles off shore in Monterey Bay (see Figures A-7 and A-8). The blue region indicates deeper conditions whereas orange indicates shallower conditions. The UAV made three forays into deep water along a relatively challenging path for the algorithm and AUV to navigate.

Figure A-9 shows the results of our algorithm for the AUV mission. The black line shows the ocean floor. The red line shows the AUV depth under the control of the former depth planning algorithm. The green line shows the actual AUV depth while being controlled by our sliced linear programming algorithm. During steep inclines, the old algorithm anticipated the depth change and turned up too early. Our algorithm induces the AUV truncate these responses allowing the AUV to track the target depth more closely (see Fig. B-10).

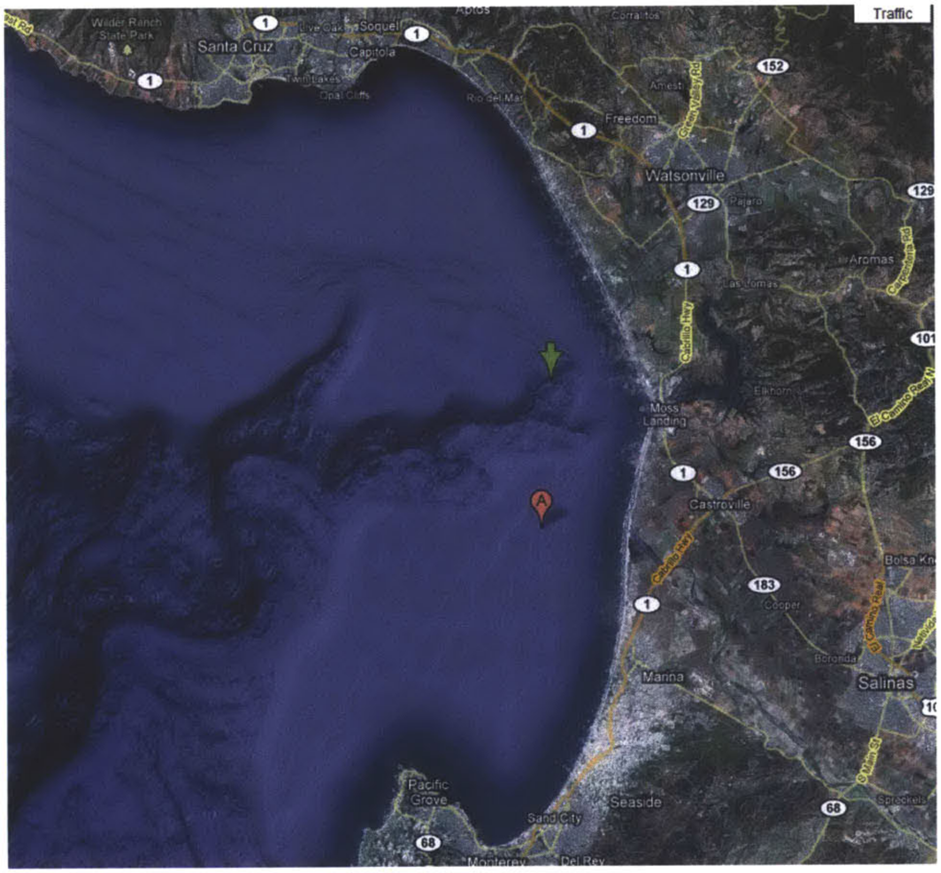


Figure A-7: We conducted a field test using our algorithm. Our spliced linear programming depth control algorithm has been successfully deployed at a topographically difficult region three miles off shore in Monterey Bay. This figure shows the geographic location of our field test.

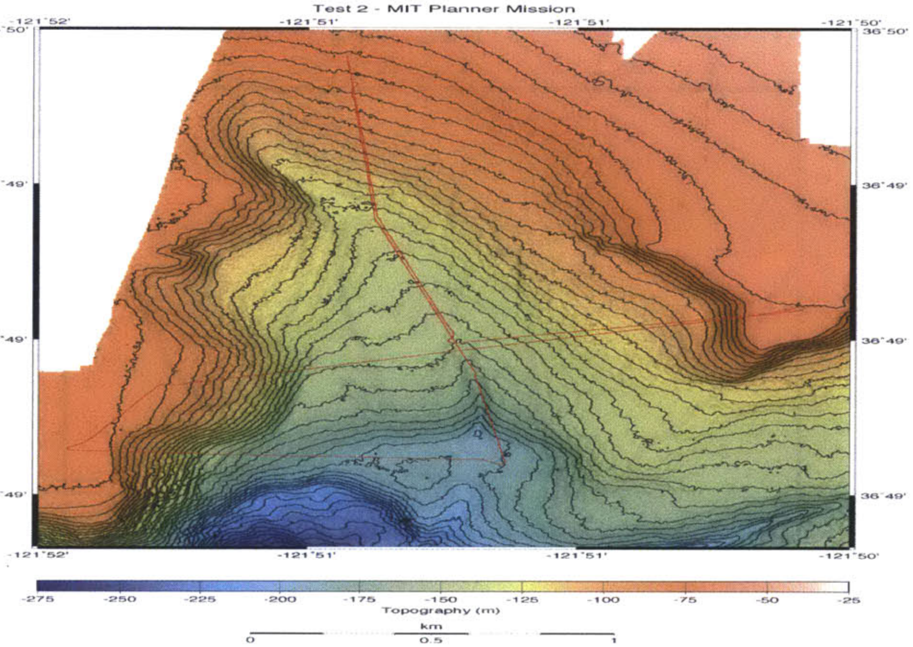


Figure A-8: The path of the mission is shown. The transect was designed to be challenging for the algorithm and AUV to navigate. The blue region indicates deep water and the orange region indicates shallow water. The mission starts in shallow water, at the top of the figure, then goes down, left, right, left and back up. The result is three forays into deep water.

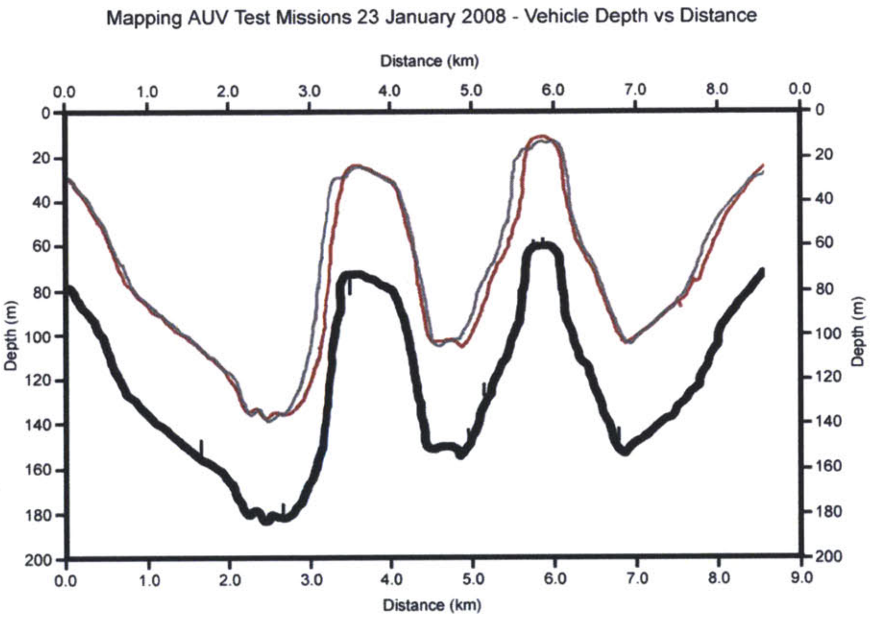


Figure A-9: Our algorithm improves the quality of scientific measurements by allowing the underwater vehicle to approach closer to the sea floor. The graph shows the field test results comparing our algorithm to the previously used algorithm. The black line shows the ocean floor. The red line shows the actual path followed when controlled by our new algorithm. The green line shows the actual altitude under the old algorithm. The results show that the AUV tracked the target depth better when controlled by our new algorithm.

Figure A-10: Our algorithm was more successful than the previously existing algorithm at maintaining the proper altitude above the sea floor. This graph shows the deviation from the target altitude for our new algorithm (in red) and for the old algorithm (in green). This graph makes it easy to see the deviation from the target altitude. The results show that the AUV tracked the target depth better when controlled by our new algorithm.

B Simulation Demonstrating Adaptive Mission Planning for Mars Exploration

Experimental Design Our system takes a set of nodes, constructs a plan to visit a subset of those nodes, and simulates the flight path to test the validity of the plan. Plan construction starts in the adaptable mission planner. The adaptable mission planner takes a set of node locations and utilities and calculates the optimal path to get the most utility given a limited amount of fuel.

The plan is then fed into the kino-dynamic path planner, which creates waypoints to guide the UAV to the next node. These waypoints are built on a one-second time scale, and thus represent a finely discretized plan. These waypoints are followed using a simulation of the environment and an autopilot to direct the aircraft. The simulation returns the actual fuel usage, which may cause changes in the plan. The block diagram of the complete system is shown in Figure B-1. Each part of the system is explained in greater detail in the following sections.

Coordinating Agile Systems Through the Model-based Execution of Temporal Plans Unlike previous solutions, our integration would enable continuous high-level planning. The integration enables the simulation framework to accept plan updates which considers recent observations when creating the plan. Continuous high-level planning would allow the UAV to enact high-level goals according to the observed environment. Specifically, it assimilates information that it learns about its environment (i.e., a more accurate map and/or its ramifications on the utility of future tasks). This integration would therefore provide a system which observes, learns and updates its higher level planning goals.

Adaptable Mission Planner The high-level mission planner has been designed to solve the finite-horizon path planning as a Selective Traveling Salesman Problem (STSP). This is also known as the Orienteering Problem. The adaptable mission planner accepts a given set of utilities respective coordinates representing the sites of potential scientific value.

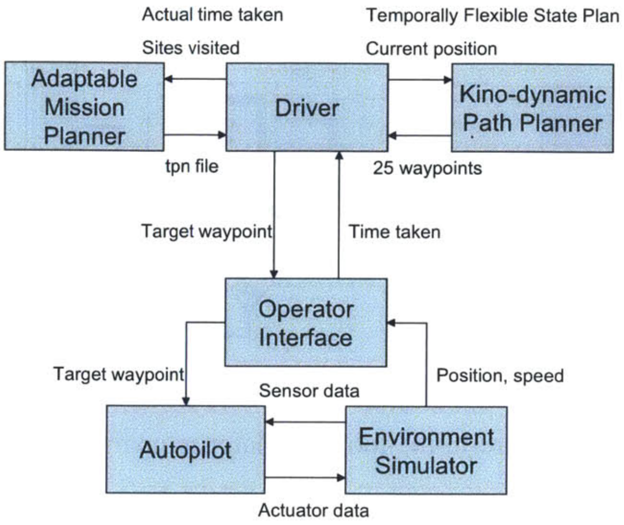


Figure B-1: Block diagram of the complete system.

These are the targeted positions on the map that represent potential exploration sites in the mission planner's model. The mission planner generates a traceable path of nodes that will maximize its utility within a given distance constraint.

The following steps outline the major parts of the adaptable mission planner:

1. Create Adjacency Matrix from node coordinates
2. Order list of utilities to match ordering of nodes. Starting node indexed at 0.
3. Create traceable path through nodes using STSP Solver
4. Remove the next node to be traveled to from the list of considered nodes
5. Reduce the remaining distance by the distance to the next node
6. Repeat from step 1 with an updated set of utilities from the UAV's sensors

.

We use a kino-dynamic path planner to determine the best route to the next science site. The kino-dynamic path planner produces closely spaced waypoints for the autopilot to follow and avoids features such as mountains.

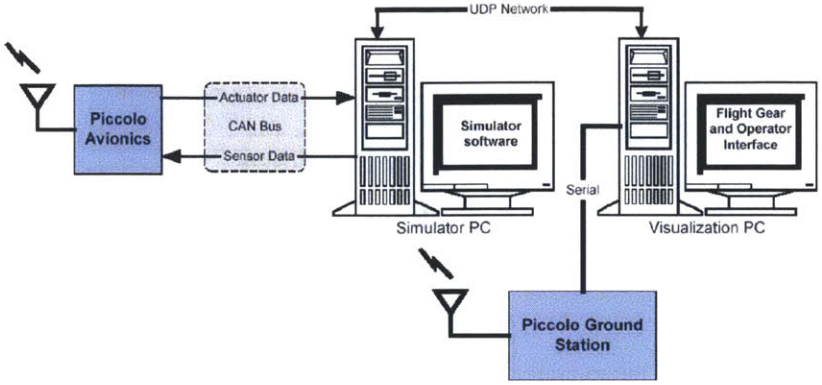


Figure B-2: Piccolo hardware-in-the-loop simulator setup made by Cloud Cap Technology

Kino-Dynamic Path Planner We modified a kino-dynamic path planner by Thomas Leaute Léauté and Williams [2005a] to take in a new plan at each high-level planning node. Each of these nodes corresponds to a science site. The system essentially combines a driver, navigator and adaptable mission planner with a hardware-in-the-loop simulator. As illustrated in Figure B-2, the simulator includes an operator interface and environment simulator with a Piccolo UAV Autopilot. The UAV autopilot handles all of the control dynamics. It takes as inputs the aircraft pose, velocity, parameters from the environment simulator and target waypoints from the operator interface. The autopilot then solves the routing problem of how to transit to the next waypoint, and sends actuator instructions to the environment simulator. The environment simulator calculates motion and sensor data given the actuator data and sends this data to the operator interface and to the autopilot as feedback. The operator interface then returns the time taken to the driver so the driver will know how long it took to reach the waypoint.

The kino-dynamic path planner generates a guided plan by creating a set of 25 waypoints that the plane traverses at a rate of one waypoint per second. The kino-dynamic path planner re-plans every 18 seconds. It uses a 25-second planning horizon, which ensures aircraft stability within that horizon. Because the planner does not look further ahead, it may leave the plane in a difficult position at the edge of this time horizon. With seven seconds of safe travel before it, the plane should be able to generate a new plan that avoids collisions or navigational destabilization.

When the plane reaches the end of its plan (the 18th waypoint), we generate a new set of waypoints. The framework is coded so as to accept a new plan at this point. Since the adaptable mission planner only plans for high-level goals (features of scientific interest), we first check to see if the plane has reached one of these sites. If not, then the kino-dynamic path planner re-plans as usual. If it has reached a science site, then we pass information back to the adaptable mission planner, which enables it to generate a new plan. The adaptable mission planner needs to calculate fuel usage from transit to the new science site and exactly which science sites have been visited so that it can generate a new plan. Once a new high-level plan has been created, the kino-dynamic path planner adopts it. In order to integrate the new plan into the system, we must remove only the relevant constraints from the constraint set, and then re-introduce the new constraints based on the new high-level plan.

When this process has been completed, we then use that plan to create a new set of waypoints via the kino-dynamic path planner. These waypoints are fed into the autopilot. This process continues looping until the plan has been completed or a time constraint has been violated. Normally, the plan will finish as planned. However, though the high-level planner estimates the travel time to each science site, the plane may encounter obstacles which cause it to take longer than expected to get to a given science site. If the simulated travel time takes too much longer than the estimate, then the plane will run out of fuel and will not be able to complete the plan. In this case a constraint violation (too much total time taken) will halt the process.

B.1 Empirical Testing of Different Adaptive Planning Approaches

Planning strategies differ in terms of factors such as risk aversion and function differently according to the rate of change in the environment. This appendix describes empirical evaluation of different planning strategies in different environments. We prepared a set of experiments in order to compare the performances of a) passive or static planning, b) intermediate planning and c) aggressive or greedy planning under different rates of environmental change.

The first plan type is the static finite-horizon plan. This strategy creates a plan to travel a certain distance (finite-horizon) at the beginning of a simulation, and then carries out the plan without making any changes (static). In the Mars exploration example, the UAV flies until it runs out of fuel. The finite-horizon is made to match the amount of fuel, so that the UAV will complete the plan at the same time it runs out of fuel. This means that the finite-horizon plan actually covers the entire mission. In practice this plan uses our STSP solver once and feeds the result into the simulator, which then reports how much of the plan was successfully completed.

The second planning strategy uses an adaptable finite-horizon planner. This is similar to the static finite-horizon planner, in that it creates a plan that covers the lifetime of the UAV. It differs however in how it re-plans after reaching each science objective, so that it can react to re-appraisal of the utility of science sites. In practice this plan has the simulator call the STSP solver after each waypoint is reached. With our modifications, the simulator is then able to switch to a new temporally flexible state plan.

The third planning strategy, the greedy approach, operates as a one step receding-horizon planner. At each step it chooses the best node to travel to and then goes there. The best node is defined using a combination of travel cost and science utility. Our greedy planner tried to maximize utility/travel cost at each step. Different weightings would be possible for a greedy planner. For example, it could only consider distance and always fly to the closest point. It could also only consider utility and always fly to the highest utility site. Different situations would favor each of these strategies, but we chose an intermediate design that favors utility divided by travel cost.

Example 1 In Figure B-3 we show a scenario with three nodes, A, B, and C, where the UAV starts at A. The distance between A and B is 2, the distance between A and C is 4, and the distance between B and C is 6. The utility of B is 3 and the utility of C is 6. If the UAV is allowed to fly a total distance of 4, then the utility-maximizing greedy planner will perform better. It will fly to C and get a utility of 6, while the cost-minimizing greedy planner will fly to B and get a utility of 3. If the UAV is allowed to fly a total distance of

8, then the cost-minimizing greedy planner will perform better. It will fly to B, then C to gain a utility of 9, while the utility-maximizing greedy planner will fly to C for a utility of 6. It is not difficult to construct similar situations for different sets of greedy planners, with different weightings between cost and distance.

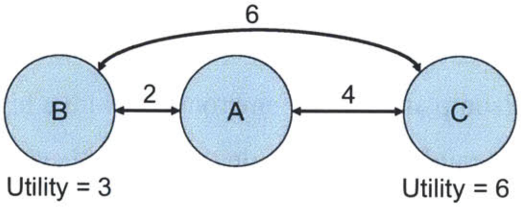


Figure B-3: Simple example scenario

Experiment Design Our experiments highlight differences between strategies. In an unchanging environment, we would expect the adaptable finite-horizon planner to make exactly the same plan as the static finite-horizon planner. This is because they would have the same knowledge at the beginning and create the same plan, and if the environment remained unchanged then the adaptable planner would have no reason to update its plan. In an unchanging environment we would also expect the finite-horizon planners to outperform the greedy planner, since they create an optimal lifetime plan.

When the assigned utilities start changing, this no longer applies. The finite-horizon planners are expected to suffer, since they no longer have perfect knowledge of the utilities. In particular, the static finite-horizon planner will be unable to react to changes in the environment, and the more severe the changes, the more random its performance will be. If the changes happen infrequently, we would expect the adaptable finite-horizon planner to perform fairly well, since it will be able to incorporate the changes into its plan. In the extreme case, however, the utilities will be completely random at each time step, and there will be no benefit from planning. In this case the greedy planner would perform well, since it would prioritize high utility points as they appear.

Example 2: Imagine a scenario with three nodes, A, B, and C, where the UAV starts at A. The distance between A and B is 2, the distance between A and C is 4, and the distance between B and C is 6. The utility of B is 3 and the utility of C is 6. The UAV is allowed to fly a total distance of 8. The finite-horizon planner will travel to B, then C, to maximize its expected reward at 9. The greedy planner will travel to C and get a reward of 6, then run out of fuel. However, if after the first time step the situation changes so that C is only worth 1, then the finite-horizon planner will only gain a utlity of 4, while the greedy planner will have already received its reward of 6. The static finite-horizon planner performs better than the adaptable finite-horizon planner in an environment with a high rate of change wherein the static planner realizes random utility while the adaptable planner does not.

Hypothesis In general, we would expect that the more frequent and significant the changes in utility, the poorer the finite-horizon planners performance. We would expect the greedy planner to not be especially affected by the changes, so its performance relative to the finite-horizon planners would improve. By running each strategy under different levels of change, we can identify the crossover point where greedy planning starts to outperform finite- horizon planning, and see where adaptable planning outperforms static planning. If different strategies perform better at different rates of change, it becomes useful to predict the amount of change. Identifying expected change allows us to adopt the most appropriate strategy for that environment.

We therefore tested each strategy under different levels of change. We chose to implement change as a probability each node at each step. When change occurs, the utility of that node is swapped with another node at random. We chose to swap utilities instead of assigning new ones so as to keep the magnitudes of the utilities the same. This also scale utilities within a given range so as to avoid the need to normalize them.

The probability or percent change is a continuous and quantitative variable that permits us to create relationships and plots between the change parameter and the performances of the planning strategies. The change parameter only affects frequency of change and not magnitude. In order to consider magnitude, we would need to change the initial assignment of utilities. Set of utilities consisting of several high utilities and several low utilities would lead to significant changes when a high utility node gets swapped with a low utility node. We therefore examined several different utility distributions in order to analyze the effects of change on a strategys performance.

Experimental Parameters The simulation was run with a roughly Gaussian distributed set of utilities, such that there are several medium values and some high and some low values. There are 11 nodes, and the values 1, 5, 5, 10, 10, 10, 10, 10, 15, 15, and 20 were distributed among them. This fairly even distribution is expected to be similar to many real scenarios. The different planning strategies were run for different frequencies of change. We implemented rates of change of 0%, 5%, 10%, 15%, ... 100%, for a total of 21 different change frequencies. These levels of change represent the chance of each node to change its utility at each time step. Each of the three planning strategies was run 25 times for each change level, and the accrued utilities were averaged over the 25 runs. In order to be fair to each of the three planning strategies, they were each run using the same random numbers. This means that the changes were calculated for each step of a run, and then all three strategies were run using that sequence of changes. This was done to keep one algorithm from getting lucky while the other algorithms got bad runs. In other words, minimize random noise in the analysis.

B.2 Results

This section describes the results of a simulation run and demonstrates the benefits of adaptable mission planning. The benefit specifically arises from re-planning at each step in the high level plan. A static planner constructs a plan at the beginning of the mission and adheres to that plan throughout the mission regardless of changing conditions. An adaptable mission planner on the other hand can periodically generate new plans that take time constraints and informational rewards into account. In this simulation, informational rewards are formulated as updated utilities for science sites.

We interfaced the adaptable mission planner with a Mars aircraft simulator. The adaptable mission planner constructs a plan that is executed in the simulation environment. The operational interface of the simulation system (Fig. B-4) displays the location of the UAV on a (representative) map of an area of Mars. The aircraft begins near a starting location where the vehicle entered the Martian troposphere. The map depicts mountainous areas which the aircraft should avoid, as well as craters, rock outcroppings and cliffs which have scientific utility. The operator interface shows the progress of the UAV as it navigates through this environment.

In constructing a plan, the adaptable mission planner takes in a set of science sites (labeled B through J), the starting location (labeled A), a set of utilities for each science site and a maximum travel distance (5 nautical miles). The utilities reflect the science value of a given site. In this example, we start out with a set of utilities shown in Figure B-4b. The initial utilities reflect 6 low value sites (C, E, G, H, I, J) and 3 sites of high interest (B, D and F). The adaptable mission planner constructs the first plan which is reflected in Figure B-4c. This plan shows that the plane is able to hit all of the high value science sites. The red arrows indicate the path that the UAV will take.

The execution of the plan starts when the adaptable mission planner sends a plan to the kino-dynamic path planner. The kino-dynamic path planner takes that plan and constructs a set of waypoints (a low-level plan), which guide the UAV along the route designated by the plan. The set of waypoints are approximately one-second apart and respect the kino-dynamics of the plane. These waypoints are sent to the auto-pilot which essentially flies the airplane. The plane flies to each waypoint in succession. To do this, the auto-pilot sends actuator instructions to the simulator which simulates both the UAV and the environment.

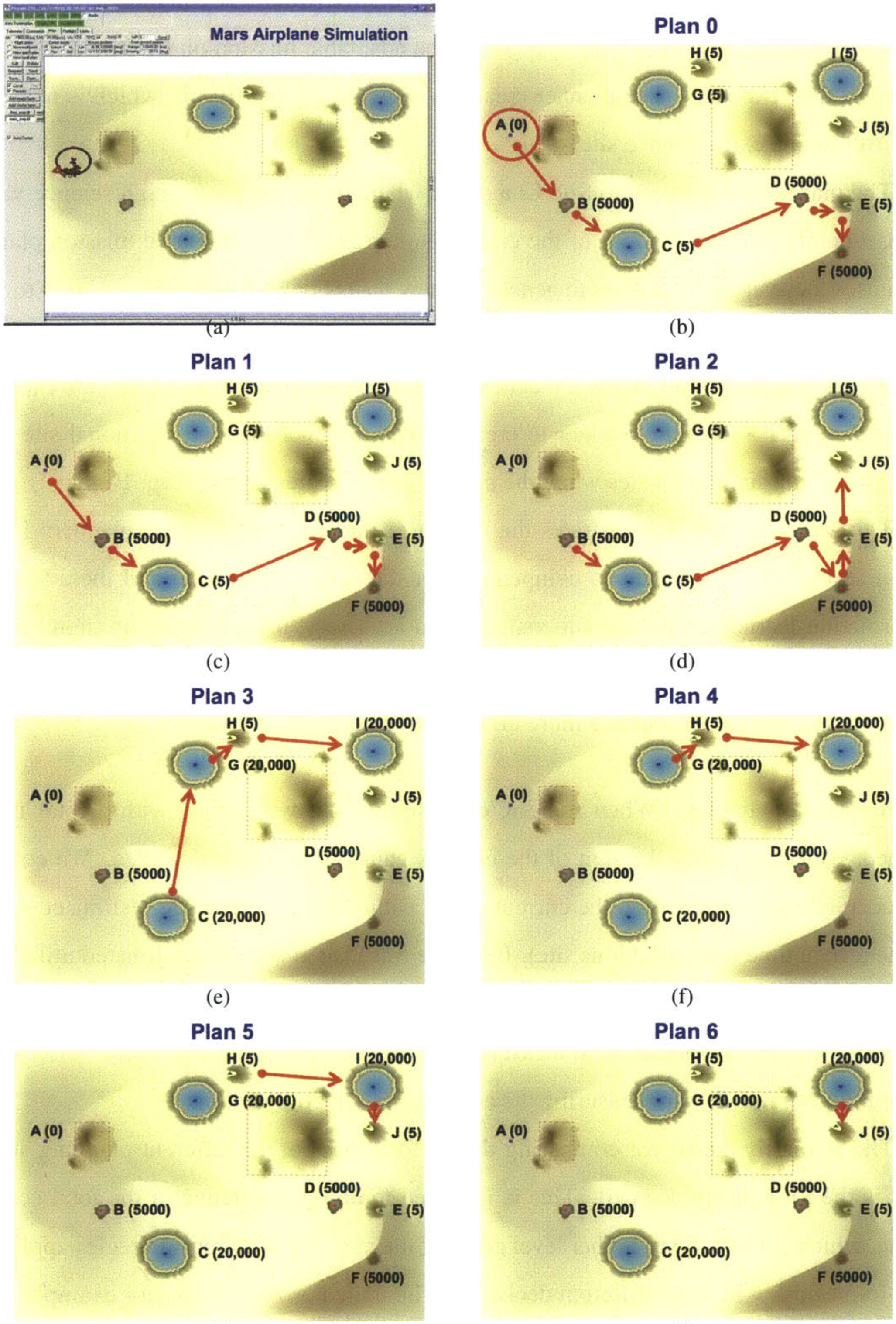
The plane initially proceeds from the Mars environment entry point (Site A) to the first science site (Site B). When the plane reaches a waypoint, it notifies the kino-dynamic path planner. The kino-dynamic path planner waits for the plane to reach the 18th node. When the 18th node is reached, it sends a new set of waypoints. Prior to this transmission however, the kino-dynamic path planner must first check to see if a science site has been reached. If one has, it sends the status information to the adaptable mission planner which generates a new plan. When the plane arrives at Site B, status information is then sent back to the adaptable mission planner, which uses this new information to generate a new plan. The planner receives the actual travel time to the current site as well as the updated utilities. The travel time to a given node is estimated in the planner during plan construction but the environment simulator determines the actual time, which result from other factors such as the kino-dynamics, the initial pose or direction and environment conditions. The plane arrives at Site B faster than expected giving the adaptable mission planner more mission-time left than expected. The planner creates a new plan (Figure B-4d), which is now able to travel to one extra science site.

The plane then proceeds to Site C, a crater where it accrues a very high science value. Consequently, the utility of all of the craters is increased. The adaptable mission planner then uses these updated utilities to generate a new plan (Figure B-4e), which is able to visit all of the remaining craters. The plane then proceeds to Site G, which results in no change in the plan (Figure B-4f). When it reaches Site H however, it returns a lower than expected travel time that allows the planner to create a plan that includes one additional site (Fig. B-4g). The mission concludes when the plane travels to Sites I and J (Fig. B-4h).

This walk-through demonstrates the benefit of the adaptable mission planner. In this example the plan is adapted to compensate for the actual travel time and the additional information learned with each site visit. When necessary, the adaptable mission planner generated a new plan which included actual travel time, as well as the knowledge gained from the visit thus resulting in a more effective planner.

Performance Metrics When running each simulation, we end up with a total utility gained, which is simply the sum of the utilities gained at each point visited. We counted the accrued utilities based on the current utility estimate for a given site (estimated values at the arrival time to the previous site). If a node was visited when the estimated utility was 5, then it would accrue as a value of 5 regardless of how the utility changes afterwards. The objective of this method is to measure how well a planning strategy works based upon what it knows rather than measuring the accuracy of the utility estimator.

The primarily metric for comparing the performances of the different planning strategies is the total utility accrued. This is intended to be representative of the total science value, which correspond to higher level goals of the mission. Different strategies applied to this framework will make different decisions. The finite horizon planner for example often forgoes a near-term reward in exchange for a higher reward later. A static planner chooses to ignore new information in favor of following its initial plan to fruition. Our experiments



(g) (h)

Figure B-4: (a) The operator interface displays the UAV's position on the map with the science sites. (b)-(h) Solution walk through.

show that the adaptive finite horizon strategy is most effective over a variety of conditions.

Analysis Figure B-5 shows the results for the Gaussian distribution of utilities. The x-axis shows the rate of change and the y-axis shows the average total utility gained. The blue line is the static finite-horizon plan, the green line is the adaptable finite-horizon plan and the red line is the greedy plan (adaptable one-step receding-horizon plan). In an unchanging environment (zero rate of change), the static planner performs as well as the adaptable planner because they operate the same. The adaptable planner considers evolving utility values but when they do not change, its performance remains the same. Both finite-horizon plans outperform the greedy plan at zero rate of change because they are optimizing over the entire mission, trading off near-term rewards in exchange for a higher overall gain.

Greater perceived environmental change diminishes the performance of both finite-horizon planners. The static planner's performance drops most quickly demonstrating the deficiencies of the static planning strategy. The adaptive finite-horizon planner performs the best during a moderate rate of change, highlighting the strength of this strategy. After about 50% change, the greedy planner outperforms the static planner and performs equitably with the adaptable finite-horizon planner. The greedy planner performs at its best under these conditions, but it does not outperform the adaptable finite horizon planner.

In summary, the static planner outperforms the greedy planner under stable conditions and the greedy planner outperforms the static planner under unstable conditions. The adaptable planning strategy performs well in all situations. It matches the static planner under stable conditions, outperforms both planners under moderately changing conditions, and matches the greedy planner under very unstable conditions. Noise in the accrued utility values (in spite of being averaged over 25 runs) a chaotic system in which utility is essentially a random variable, especially for the static planner. The two finite-horizon planners also highly similar inflection points. This behavior occurred because they were run using the same set of random conditions as described above.

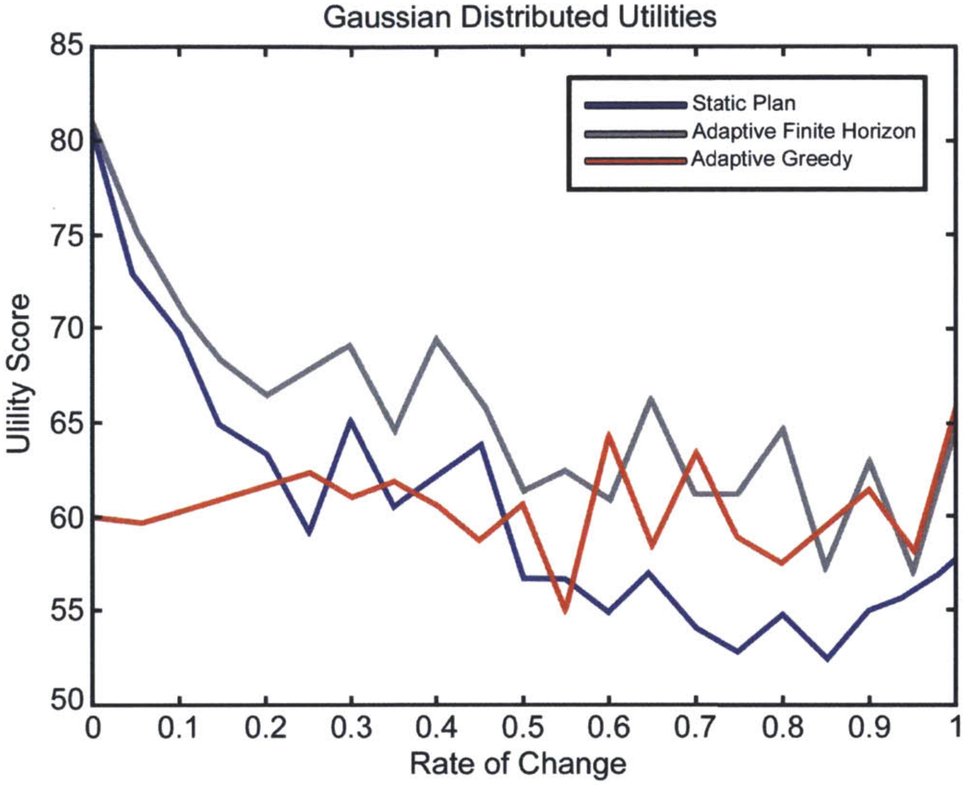


Figure B-5: Total Utilities for a Gaussian Distributed Scenario: The graph shows the results for the Gaussian distribution of utilities. The rate of change is along the x-axis and the average total utility gained is on the y-axis. The blue line is the static finite-horizon plan, the green line is the adaptable finite-horizon plan and the red line is the greedy plan (adaptable one-step receding-horizon plan). These results are very interesting. As expected, the static planner outperforms the greedy planner under stable conditions and the greedy planner outperforms the static planner under unstable conditions. However, the most interesting thing that these results show is that the adaptable planning strategy is good in all situations. It is as good as the static planner under stable conditions, it is better than either planner under moderately changing conditions, and it is approximately equal in performance to the greedy planner under very unstable conditions.

C Learning MDP policies offline (DUM Background)

In this section we define the Markov decision process, our problem framework. We outline an optimal but computationally intractable policy-learning algorithm followed by a tractable but suboptimal policy-learning algorithm. We then explain that the suboptimality of the second algorithm can be modeled in terms of policy-representational uncertainty and the probability of error, and how this uncertainty can then be used in an online reoptimization algorithm.

C.1 Markov decision process defined

An MDP is defined by its state set S, action set A, state transition probability distribution P, and reward function R. When executing action a, in state s, the probability of transitioning to state s' is denoted as $\mathcal{P}^a_{ss'}$ and the expected reward associated with that transition is denoted $\mathcal{R}^a_{ss'}$. A policy assigns an action to each state of the MDP. The value of a state under a policy ($V_\pi(s)$) is the expected value of the discounted sum of rewards obtained when policy π is followed, starting in state s. The objective is to find an optimal policy π^*, which maximizes the value of every state ($V_{\pi^*}(s) = V^*(s)\ \forall s$).

The optimal value function (V^*) is the solution to the Bellman optimality equations (e.g. Bertsekas and Tsitsiklis [1996a]): $\forall s \in S$

$$V(s) = \max_{a \in A} \sum_{s' \in S} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')] \tag{C.1}$$

where the discount factor $0 \leq \gamma < 1$ makes future payoffs less valuable than more immediate payoffs. The Bellman equation (Eq. C.1) is also referred to as the dynamic programming equation, and describes a one-step look-ahead with respect to the value function. It is known that the optimal policy π^* can be determined from V^* as follows: $\pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^*(s')]$. Therefore, the optimal value function is virtually the MDP solution.

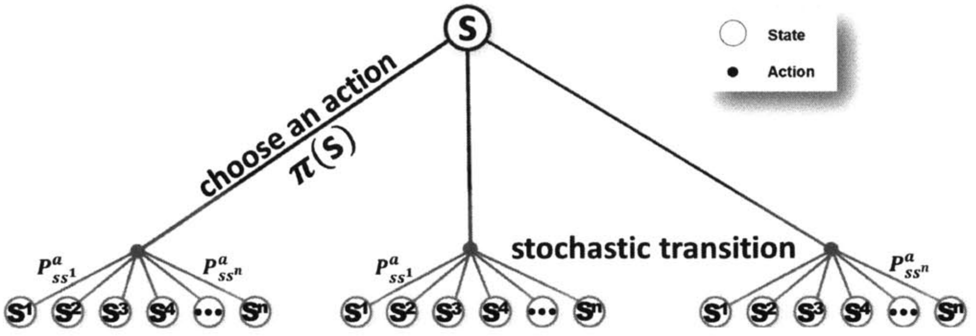


Figure C-1: MDP mechanics: In the MDP framework, we find ourselves in state s where policy π chooses an action to execute. Although the action is chosen, we do not know for sure which state we will transition into. When executing action a, in state s, the probability of transitioning to state s' is denoted as $\mathcal{P}^a_{ss'}$. The objective of the MDP optimization problem is to find the best policy, that is the policy that maximizes the value of every state.

C.2 An optimal offline policy for a small discrete MDP can be found via value iteration

Given our MDP model ($\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$), value iteration can be used to determine the optimal value function. Starting with an initial guess, V_0, we perform the following iteration $\forall s$:

$$V_{k+1}(s) = \max_{a \in A} \sum_{s' \in S} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V_k(s')]. \tag{C.2}$$

The contraction mapping

$$\max_{s \in S} |V_{k+1}(s) - V^*(s)| \leq \gamma \max_{s \in S} |V_k(s) - V^*(s)| \tag{C.3}$$

implies that V_k converges to V^* as $k \to \infty$. Algorithm 13 (see inset that follows) shows the value iteration algorithm in its entirety. The algorithm takes as input the MDP tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$. Each state value is initialized to zero and updated via the Bellman equation. After the convergence threshold is met, the entire table of state values is returned. That table of values (V) can be used to choose the optimal action in real time using the following equation:

$$a^* = \arg\max_{a \in A} \sum_{s' \in S} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^*(s')] \tag{C.4}$$

Algorithm 13 Exact value iteration algorithm for solving MDPs.

```
ValueIteration($\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$) $\Leftarrow$ {
for all $s \in S$ do
  $V_0(s) \Leftarrow 0$
end for
$t \Leftarrow 0$
repeat
  $t \Leftarrow t + 1$
  for all $s \in S$ do
    $V_t(s) \Leftarrow \max_a \gamma \sum_{s' \in S} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + V_{t-1}(s')]$
  end for
until $\max_s |V_t(s) - V_{t-1}(s)| < \epsilon$
return $V_t$
}
```

Relationship between value function $V(s)$ and state-action value function $Q(s, a)$

While the state-action value function $Q(s, a)$ provides the value in state s of any action a, the value function $V(s)$ provides the value in state s of the best action selection as determined by policy $\pi(s)$ as follows;

$$\pi(s) = \arg\max_{a \in A} Q(s, a) = \arg\max_{a \in A} \sum_{s' \in S} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')] \tag{C.5}$$

Thus, the value in state s is equal to the maximum Q-value $\forall a \in A$

$$V(s) = \max_{a \in A} Q(s, a) \tag{C.6}$$

or simply the Q-value of the action selected by policy $\pi(s)$.

$$V(s) = Q(s, \pi(s)) \tag{C.7}$$

Likewise, the state-action value function can be defined as the value obtained by acting greedily with respect to the value function.

$$Q(s,a) = \sum_{s' \in S} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]. \tag{C.8}$$

While the previous Algorithm 13 assumes the end result of value iteration to be the value function, we actually care most about finding policy π_t where

$$\pi_t(s) = \arg\max_a \sum_{s' \in S} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V_{t-1}(s')] \tag{C.9}$$

These new concepts are used in Algorithm 14 shown below. Algorithm 14 differs from Algorithm 13 because it returns policy π_t and uses the state-action value function as an intermediary between the value function and the policy, which explicitly shows how the three relate. Algorithm 14 takes as input the MDP tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$. Each state value is initialized to zero. Using the state values, a state-action value is then calculated for each state-action pair. The policy and state values are then recalculated from the state-action values. This process repeats until a convergence threshold is met. Algorithm 14 returns policy π_ϵ.

C.3 Policy Iteration

Now that we have defined an MDP, value iteration, and the relationship between the value function and the state-action value function, we can bring these together into the more general framework of policy iteration. As we know, a policy is the mechanism that maps each state to an action, thereby deciding for us how to act in each situation. Policy iteration (PI) is an algorithm that calculates a sequentially improving series of policies. Thus the PI algorithm is premised upon having an existing policy and improving it. PI has three steps:

- Policy Iteration Steps:
  - Initialization
  - Evaluation

Algorithm 14 Exact value iteration algorithm using Q-values.

```
ValueIterationQ($(\langle \mathcal{S}, \mathcal{A}, \mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'}, \gamma \rangle)$) $\Leftarrow$ {
for all $s \in S$ do
  $V_0(s) \Leftarrow 0$
end for
$t \Leftarrow 0$
repeat
  $t \Leftarrow t + 1$
  for all $s \in S$ do
    for all $a \in A$ do
      $Q_t(s,a) \Leftarrow \sum_{s' \in S} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V_{t-1}(s')]$
    end for
    $\pi_t(s) \Leftarrow \arg\max_a Q_t(s,a)$
    $V_t(s) \Leftarrow Q_t(s, \pi_t(s))$
  end for
until $\max_s |V_t(s) - V_{t-1}(s)| < \epsilon$
return $\pi_t$
}
```

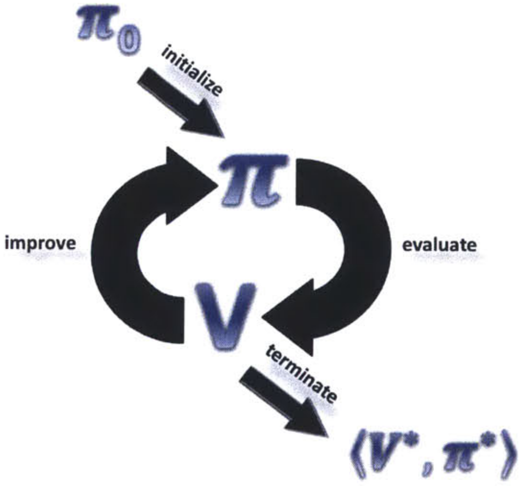


Figure C-2: A policy is initialized in some way, often arbitrarily. That policy is evaluated, to find $V(x)$. That is, we find the value of that policy in each state, where the value is the discounted future reward of executing that policy. Policy evaluation can be done using Monte-Carlo simulation as discussed in Section D.3. Equation C.9 is used to extract an improved policy from the existing value function.

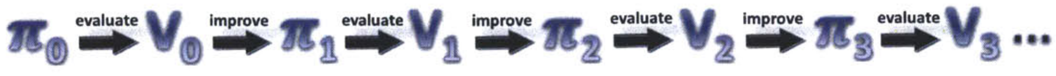


Figure C-3: MDP mechanics: In the MDP framework, we find ourseleves in state s where policy π chooses an action to execute. Although the action is chosen, we do not know for sure which state we will transition into. When executing action a, in state s, the probability of transitioning to state s' is denoted as $\mathcal{P}^{a}_{ss'}$. The objective of the MDP optimization problem is to find the best policy, that is the policy that maximizes the value of every state.

- Improvement

The above steps are depicted in Figure C-2. A policy is initialized in some way, often arbitrarily. That policy is evaluated, to find $V(x)$. That is, we find the value of that policy in each state, where the value is the discounted future reward of executing that policy. Policy evaluation can be done using Monte-Carlo simulation as discussed in Section D.3. Equation C.9 is used to extract an improved policy from the existing value function.

Policy Improvement Shorthand

Policy evaluation is the process of figuring out the value of executing a given policy, for each and every state, given a policy, which is performed after we improve the policy. The value function and the policy are closely related. The policy improvement algorithm obtains a new policy π^{k+1} from policy π^k. The policy improvement step can simply be minimizing the DP equation or it can be something more complicated. Likewise, the DP equation corresponds to a one-step look-ahead policy over V. TV is shorthand for the DP equation applied to the entire state-space such that:

$$V^{k+1} = TV^k \tag{C.10}$$

The DP equation shorthand for an individual state is:

$$(TV)(x) = min_{u \in U(x)}[g(x,u) + \alpha \sum_{j=1}^{n} p_{ij}(u)V(j)],\ for\ i = 1, 2, 3, ..., n \tag{C.11}$$

The DP equation may be applied multiple times, the result of which is:

$$V^{k+b} = T^b V^k \tag{C.12}$$

In regular value iteration, the improvement step is the DP algorithm, which is a one-step look-ahead. Likewise, the policy extraction algorithm is the DP algorithm. But the point of this section, is that the policy extraction and improvement algorithms can be anything. For example, they can be a two-step look-ahead algorithm, a five-step look-ahead algorithm, or they can even be a twenty-step search algorithm. The only restriction is that they are based on the current value function and improve upon the current value function. This idea is discussed and expanded in Section 3.7 when we devise an online search algorithm that exploits the value function. In Section 3.7, the value function is used like a heuristic to guide online search. Arguably, T uses V as a heuristic regarding how good a future state is and consequently, how good a given action is. Given that the cumulative reward is what we value and given the short sightedness of an OSLA policy improvement step, it stands to reason that a multi-step look-ahead is better (albeit more computationally involved). That online search algorithm presented in Section 3.7 is designed for policy extraction, and improves upon the policy learned offline. A similar method could be used with policy iteration.

Alternate Policy Improvement Methods

There are any number of ways that you can improve upon an existing policy. They all involve simulating or looking ahead.

It is clearest to consider this idea under a deterministic problem as we consider any old deterministic policy. In other words, the policy always chooses the same action from a given state and that action always leads to a particular next state. We have evaluated that policy to get $V(x)$. We are in state x and we wish to consider alternatives to $\pi(x)$. Suppose action $a = \pi(x)$ and action a takes us into state x^a. Likewise actions b and c take us into states x^b and x^c. We therefore do a one step look ahead simulation to see if $V(x^b) > V(x^a)$ or $V(x^c) > V(x^a)$. If either $V(x^b)$ or $V(x^c)$ is greater than $V(x^a)$, then integrating that new action into π would result in a better policy. Likewise, we can do this for each and every state in the problem's state-space.

In exact value iteration there is no reason to do anything but one-step look-ahead policy improvement. It is guaranteed to converge to the optimal answer and it is quicker than doing a more complicated policy improvement method. However, when the value function is merely approximated and interpolations are made for unexplored states, then the guarantee that the Bellman Equation's one-step look-ahead always improves your policy no longer holds.

C.4 A suboptimal offline policy for a realistically large MDP can be found via approximate value iteration (review)

The above formulation and algorithms assume a discrete state set S. If the state space is discrete, V and Q can be represented as a table of values, one record for each discrete state; the table of values is initialized arbitrarily and improved iteratively. The problem with this approach is that many real world state spaces are continuous and an acceptable discrete representation is intractably large, exceeding memory capacity. Additionally, we must perform a time-consuming value iteration backup for each state.

We address these complexities by representing Q using an estimation architecture ($\widehat{Q}$). The estimation architecture stores the state action-value function in a compressed form, such as a linear function over problem variables rather than an explicit combination of each possible variable assignment. When used in approximate value iteration (Bertsekas and Tsitsiklis [1996a]), the estimation architecture alleviates the storage problem and shares information across state variables, decreasing learning time. The estimation architecture, therefore, solves both problems inherent to exact value iteration.

Algorithm 15 outlines the approximate value iteration algorithm where $\widehat{Q}$ is the approximate value function represented by an estimation architecture. Algorithm 15 takes as input the MDP tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where $|\mathcal{S}|$ is very large, possibly infinite. Algorithm

Algorithm 15 Approximate state-action value iteration algorithm where $\widehat{Q}$ stands for an approximation architecture representation of the state-action value function, $\bar{Q}$ stands for a lookup table of state action values over a subset $\bar{S}$ of the full statespace S.

```
ApproxStateActionValueIteration($\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$) $\Leftarrow$ {
for all $s \in \bar{S} \subset S$ do
  for all $a \in A$ do
    Initialize ($\bar{Q}(s,a)$)
  end for
end for
InitializeApproximationArchitecture($\bar{Q}_t, \widehat{Q}_t$)
$t \Leftarrow 0$
repeat
  $t \Leftarrow t + 1$
  for all $s \in \bar{S} \subset S$ do
    for all $a \in A$ do
      $\bar{Q}(s,a) \Leftarrow \sum_{s' \in S} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma \max_a \widehat{Q}(s',a)]$
    end for
  end for
  $\widehat{Q} \Leftarrow$ UpdateApproximationArchitecture($\bar{Q}, \widehat{Q}$)
  for all $s \in \bar{S}$ do
    $\pi(s) \Leftarrow \arg\max_a \widehat{Q}(s,a)$
    $\bar{V}_t(s) \Leftarrow \bar{Q}(s, \pi(s))$
  end for
until $\max_{s \in \bar{S}} |\bar{V}_t(s) - \bar{V}_{t-1}(s)| < \epsilon$ , $\forall s \in \bar{S}$
return $\widehat{Q}_t$
}
```

15 begins by randomly initializing state-action value table $\bar{Q}(s, a)$ over state subset $\bar{S}$. The approximation architecture $\widehat{Q}$ is then initialized based on the $\bar{Q}$ table. We then perform a Bellman backup over state subset $\bar{S}$, using $\widehat{Q}$ for future state-action value estimates. The newly computed state-action values are stored in table $\bar{Q}$ and the policy, approximation architecture and state value table are then updated; this process repeats until a convergence threshold is met. Algorithm 15 returns approximation architecture $\widehat{Q}_t$.

D Online simulation based control algorithms (DUM Background)

D.1 Online simulation based control

In this section we introduce simulation as a useful tool for building control policies. We introduce the concept of a policy tree and describe how every other algorithm, such as the well-known Monte Carlo sampling algorithm, is a method for sampling from and more efficiently approximating that policy tree. We contrast existing full feedback and open-loop algorithms. We discuss existing guided and unguided algorithms. This naturally motivates and leads into our uncertainty guided algorithm. Online search algorithms that are guided by a policy that was learned offline inevitably improve upon that policy. DUM takes that idea a step further. The motivation of DUM is to guide online search in an optimal way with respect to what is known at the time.

Why is simulation useful? We use a computer to imitate a real-world process in order to make predictions and decisions regarding how to act within that system. Simulation as a method can be employed to enlighten us on how even complex systems work. Thus, we do simulation when analytical solutions and in-depth analysis is impossible; we do this for complex systems that we do not fully understand.

Traffic flow example: For example, we can simulate traffic patterns and bottle-necks around a city. Even a city of thousands of traffic lights can be simulated, because we understand the individual constituents of the system. So, even though we do not understand the whole system, we can still build it intersection by intersection and thereby simulate an entire city. We can then use this computer simulation model to test design changes that will get rid of bottlenecks. Specifically, traffic lights control traffic flow through the city. Thus, we can simulate the city under different traffic control policies. The simulation provides a way to evaluate and compare proposed bottleneck reducing policies.

Hyper-parameters: We know that simulation is used to analyze complex systems. But specifically, simulation is used to estimate certain statistics of distributions within that system such as the mean of the distribution of interest. For example, we could estimate the mean travel time between two points within a city under a specific traffic control policy. Most importantly, we can estimate how precise that estimate is. We may estimate that the mean travel time between point A and B under policy π is 10 minutes. Yet, we may maintain a very large variance over that measure because we are not sure about its true value. Quantifying that measure's uncertainty helps us to better compare it to the same measure under a different policy.

It is important to make the distinction between the uncertainty over a measure such as expected travel time, and the intrinsic variation of travel time in the real world. We know that the travel time from point A to B may actually vary. That is to say, it is not the same for every trip. So, we can also estimate that real-world dynamical variation, measured as a variance over time-traveled. Likewise, we will never truly know what the value of that variation measure is. But we can determine how well we know that value, represented as uncertainty over the true real-world variation in travel time. Confusingly, this may very well be a variance about or with respect to a variance. These parameters about parameters are called *hyper-parameters* and they are very important to decision uncertainty minimization (see DeGroot and Schervish [2001]).

Monte Carlo analysis: To summarize, we may simulate a system even without a theory regarding how it works. We can determine certain features of the system such as the mean and variance of some important measure. Knowing how well we know something is very important when making decisions, and is a key property of DUM. This type of analysis involves chance and randomness. Consequently, we refer to it as Monte Carlo analysis, named after the European gambling mecca.

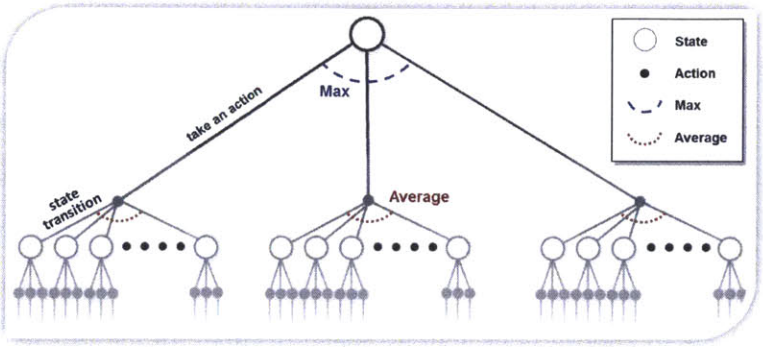


Figure D-1: Starting from the current state, the policy tree is a complete elicitation of every possible action emanating from the current state and every possible future state resulting from each action as depicted in Figure D-1. A policy graph considers all possible actions that can be taken and all possible subsequent future states stemming from that action. This procedure continues recursively, such that the policy graph captures and characterizes all possible action-state sequences. At each level, starting from the bottom, we average over outcomes and maximize over actions, resulting in the optimal full-feedback policy.

D.2 Policy Tree

In this section we define a complete policy tree (Figure D-1), which provides the optimal solution to the control problem. We will also walk through the algorithm for computing the policy tree. This algorithm is very general and optimal but computationally expensive. Computing the policy tree online or otherwise is completely unrealistic for any of the real problems we face. Although this algorithm is prohibitively expensive computationally, it is instructive in order to elucidate what that optimal full-feedback policy looks like.

Starting from the current state, the policy tree is a complete elicitation of every possible action emanating from the current state and every possible future state resulting from each action as depicted in Figure D-1. A policy graph considers all possible actions that can be taken and all possible subsequent future states stemming from that action. This procedure continues recursively, such that the policy graph captures and characterizes all possible action-state sequences. At each level, starting from the bottom, we average over outcomes and maximize over actions, resulting in the optimal full-feedback policy.

Full feedback: A full feedback policy considers the consequences of future actions, which are contingent upon the results of the current action and any intermediate actions.

This is in contrast to an open loop policy, which is sometimes called a plan. Many autonomous controllers also allow some feedback through replanning, but are not full feedback. Full feedback is important in some situations. For example, suppose you have the choice of opening one of two doors. Behind one door is a dragon and behind the other door is your friend Bob bringing you birthday presents. In the first step you choose which door to open. In the second step you choose to attack with your sword or not. In the open loop plan you have to choose your actions ahead of time. In the full feedback plan you can select options along the way depending on your observations of the situation. If you were to evaluate the plan of $\langle \text{door one}, \text{attack} \rangle$, you would average over the cases where you encountered the dragon and did the right thing and encountered Bob and did the wrong thing. Likewise, if you evaluate the plan $\langle \text{door one}, \text{invite inside} \rangle$ you would average over the cases where you encountered the dragon and did the wrong thing and encountered Bob and did the right thing. A better policy considers what the agent observes when opening the door. Preferably, you could option to attack when you see the dragon and invite inside when you see Bob. That way, you would be averaging over the outcomes of the best action taken in each situation. See Figure D-2. The resulting policy of a policy tree is equivalent to using exact dynamic programming for time-dependent MDPs.

Policy tree walk-through: A policy tree steps forward in time, simulating each possible action. For each action, each possible eventuality (future state) is simulated. Part of evaluating a given action entails averaging the rewards garnered. the rewards $\mathcal{R}^a_{ss'}$ are averaged, weighted by their probability of occurring $\mathcal{P}^a_{ss'}$ However, we also want to average in the future rewards that stem from this initial action. Therefore, we simulate forward again. We do this for each action, and take the value from the best action. This can be done many steps forward, recursively, all the way to the end of the control problem. We then maximize over control actions on the way back. **`PolicyTree`** returns the following value along with the associated action:

$$\max_{a \in A} \sum_{s' \in S} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma \texttt{PolicyTree}(s')] \tag{D.1}$$

Algorithm 16 Policy Tree: A policy tree is the most general and optimal planning algorithm. Given enough samples (N), this algorithm chooses the best action $a \in \bar{\mathcal{A}}$ where $\bar{\mathcal{A}} \subseteq \mathcal{A}$ given all subsequent actions are also limited to $\bar{\mathcal{A}}$. This brute-force approach to full-feedback control, makes no assumptions regarding certainty-equivalence, does not require a base policy, and does not sacrifice optimality by ignoring the effects of future contingencies. $\bar{Q}$ denotes the state-action value lookup-table, which we are building. However, $Q(s', a')$ refers to the future value of taking action a' from state s', as estimated by the recursive Monte Carlo planning algorithm. d = search depth. The computational complexity is $O((|A|N)^d)$.

```
 1: PolicyTree($s \in S, \mathcal{M} \equiv \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$) $\Leftarrow$ {
 2: if $s \in T$ then
 3:     return 0
 4: end if
 5: for all $a \in \mathcal{A}$ do
 6:     $\bar{Q}(s, a) \Leftarrow 0$
 7:     for all $s' \in \mathcal{S}$ do
 8:         $\bar{Q}(s, a) \Leftarrow \bar{Q}(s, a) + \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma$ PolicyTree($s', \ldots$)]
 9:     end for
10: end for
11: return $\max_{a \in \mathcal{A}} \bar{Q}(s, a), \arg\max_{a \in \mathcal{A}} \bar{Q}(s, a)$
12: }
```

We will walk through the pseudocode of Algorithm 16. In Line 1 we pass in the current state and the problem model into **PolicyTree**. Line 2 checks to see if the problem has reached the ending state T. If so, we pass back the value of 0 to the previous recursive call. If there is no terminal state, then **PolicyTree** never ends. In that case, a maximum forward search depth can be used in practice. Line 5 loops through all actions. In the case of continuous actions, such as how hard to push the accelerator or selecting the best roll-rate for an aircraft maneuver, action discretization may be employed. Line 6 initializes the state-action value so that we can average over all possibilities. Line 7 loops through all possible future states s'. Line 8 computes the weighted current reward plus discounted future reward for the future state instance s' and adds it to the weighted average. After all is said and done, we return the action with the maximum value.

PolicyTree is very complex. The complexity of a policy tree stems from its recursive nature. The complexity of **PolicyTree** is $(|A||S|)^h$ where $|A|$ is the number of possible actions, $|S|$ is the number of system states and h is the number of steps required to get to the end. We can reduce the complexity by sampling the states instead of eliciting them all, testing fewer actions or limiting the lookahead. The next Subsection is a standard way to reduce the complexity by sampling fewer states.

D.3 Monte Carlo sampling based planning

First we present the very general optimal full-feedback-sampling-based algorithm (number 17). This algorithm makes no assumptions regarding certainty-equivalence, does not require a base policy, and takes into account the effect of future contingencies. The algorithm begins with the current state $s \in S$ and model $\mathcal{M}$. It then tries all actions within set $\bar{\mathcal{A}}$. For each action, the algorithm collects and averages N samples, by simulating the state-action, then recursively calling itself with the new state. The algorithm eventually returns the action with the highest value.

Algorithm 17 Generic Monte Carlo Planning: This is the most general optimal Monte Carlo sampling-based planning algorithm. Given enough samples (N), this algorithm chooses the best action $a \in \bar{\mathcal{A}}$ where $\bar{\mathcal{A}} \subseteq \mathcal{A}$ given all subsequent actions are also limited to $\bar{\mathcal{A}}$. This brute-force approach to full-feedback control, makes no assumptions regarding certainty-equivalence, does not require a base policy, and does not sacrifice optimality by ignoring the effects of future contingencies. $\bar{Q}$ denotes the state-action value lookup-table, which we are building. However, $Q(s', a')$ refers to the future value of taking action a' from state s', as estimated by the recursive Monte Carlo planning algorithm. $d =$ search depth. The computational complexity is $O((|A|N)^d)$.

```
MonteCarloPlanning($s \in S, \mathcal{M} \equiv \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle, d \in \mathbb{N}$) $\Leftarrow$ {
if $d \equiv 0$ then
  return 0
end if
for all $a \in \bar{\mathcal{A}} \subseteq \mathcal{A}$ do
  $\bar{Q}(s,a) \Leftarrow 0$
  for $n = 1 : N$ do
    $s' \Leftarrow$ MonteCarloSimulation($s, a, \mathcal{M}$)
    $\{Q(s',a'), a'\} \Leftarrow$ MonteCarloPlanning($s', \mathcal{M}, d-1$)
    $\bar{Q}(s,a) \Leftarrow \bar{Q}(s,a) + (\mathcal{R}^a_{ss'} + Q(s',a'))/N$
  end for
end for
return $\max_{a \in \mathcal{A}} \bar{Q}(s,a), \arg\max_{a \in \mathcal{A}} \bar{Q}(s,a)$
}
```

Although this algorithm is optimal, it is also computationally intractable for most real-

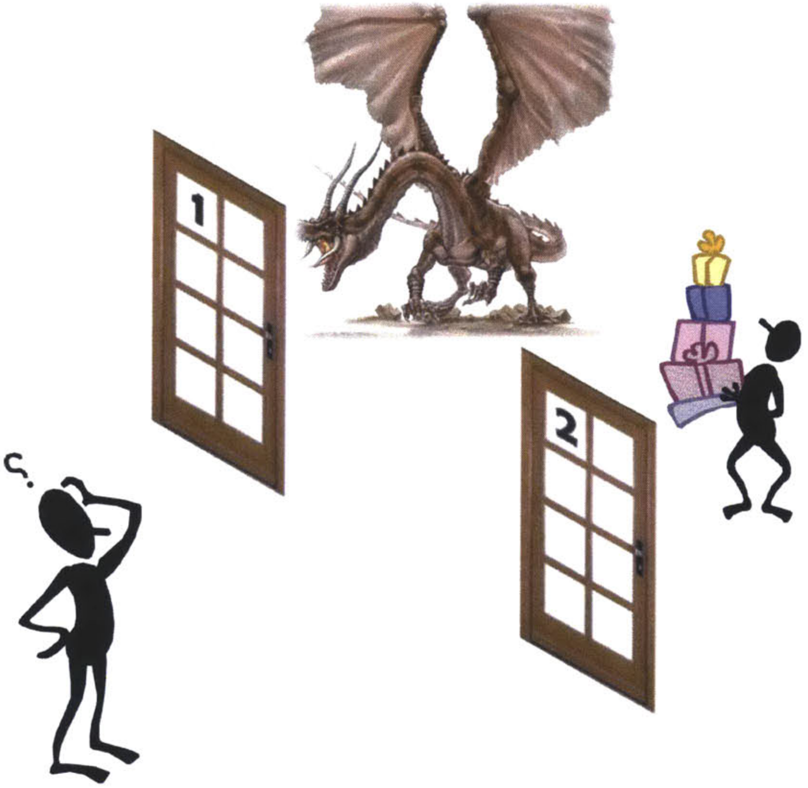


Figure D-2: A full feedback policy considers the consequences of future actions, which are contingent upon the results of the current action. Full feedback is important in some situations. For example, suppose you have the choice of opening one of two doors. Behind one door is a dragon and behind the other door is your friend Bob bringing you birthday presents. In the first step you choose which door to open. In the second step you choose to attack with your sword or not. In the open loop plan you have to choose your actions ahead of time. In the full feedback plan you can select options along the way depending on your observations of the situation. If you were to evaluate the plan of $\langle \text{door one}, \text{attack} \rangle$, you would average over the cases where you encountered the dragon and did the right thing and encountered Bob and did the wrong thing. Likewise, if you evaluate the plan $\langle \text{door one}, \text{invite inside} \rangle$ you would average over the cases where you encountered the dragon and did the wrong thing and encountered Bob and did the right thing. A better policy considers what the agent observes when opening the door. Preferably, you could option to attack when you see the dragon and invite inside when you see Bob. That way, you would be averaging over the outcomes of the best action taken in each situation.

world problems. There are several ways to address this intractability. The first option is to ignore the stochastic aspect of the problem, and simply model a nominal or expected trajectory. This method is called certainty-equivalence and will be explained next. A second option, called open-loop feedback control, ignores new information when making future control decisions but does stochastically evaluate each plan. The third option is to re-evaluate $\langle state, action, policy \rangle$ tuples online via simulation. This method, rollout, considers both problem stochasticity and future contingencies, but it must be boot-strapped with a base policy.

Algorithm 18 Monte Carlo simulation s function entails simulating one step of the system, starting at state s and executing action a

```
MonteCarloSimulation ($s \in s, a \in A, M \equiv \langle S, A, P, R, V \rangle$) $\Leftarrow$ {
$\mathcal{P}^a_{ss'}$
Select action $s'|s, a$ proportional to $\mathcal{P}^a_{ss'}$
$s' \in S$
where $P(s'|s, a) = \mathcal{P}^a_{ss'}$
return $s'$
}
```

D.4 Certainty-equivalence

Our next algorithm makes a certainty-equivalence assumption to reduce the complexity of its analysis. The assumption is that the utility of the expected outcome is equal to the expected utility, which is the average utility over all possible outcomes.

The expected outcome, $E[s'|s, a]$, is calculated as

$$E[s'|s, a] = \sum_{s' \in S} s' \mathcal{P}^a_{ss'} \tag{D.2}$$

and denoted by

$$\bar{s'} = E[s'|s, a] \tag{D.3}$$

given that such an average is even meaningful. For a numerical state space, $\sum_{s' \in S} s' \mathcal{P}^a_{ss'}$ would represent the average next state. The $\bar{s'}$ for a discrete non-numeric unordered state space can be interpreted as being the most likely state. For a Gaussian distributed state transition outcome, $\bar{s'}$ is both the average and most likely next state. The point is that certainty equivalence means there is always some deterministic representative next state $\bar{s'}$ such that $\mathcal{R}^a_{s\bar{s'}} + \gamma V(\bar{s'}) = \sum_{s' \in S} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]$. To break that down, the value of s given the expected outcome $\bar{s'}$ certainty equivalence assumption is simply calculated as:

$$V(s) = \mathcal{R}^a_{s\bar{s'}} + \gamma V(\bar{s'}) \tag{D.4}$$

rather than

$$V(s) = \sum_{s' \in S} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')] \tag{D.5}$$

where $a = \pi(s)$. Therefore, if true, we could correctly and more easily calculate our policies using this substitute deterministic problem as shown in Algorithms 21 and 22.

An example of where this assumption is actually applied is the Riccati equation, when calculating the settings for a linear-quadratic regulator. Indeed the certainty equivalence property does hold for that very specialized linear-quadratic-Gaussian problem definition (see Athans [1971] and Bertsekas [2005b]). However, the real world fails to comply to such a neat and tidy model.

To review, certainty-equivalence means that a policy found to be optimal with respect to the nominal system is also optimal in the fully stochastic system. By nominal system we are referring to the expected state-evolution sequence. For example, in a system with additive state-transition noise, we simply ignore the noise. Therefore, when planning in such a system we can assume a nominal system disturbance and only simulate that nominal path, for a given policy or plan. Turning the analysis into a deterministic problem makes it much simpler. In fact, when certainty equivalence is assumed, and we assume that the system evolves deterministically along the nominal trajectory, a policy and a plan are the same in that imagined environment. Therefore, the certainty equivalent approximate solution is even more efficient to compute than the open loop feedback control solution.

Although certainty-equivalence only holds for a minority of systems, the planning strategy can be applied to any system with varying quality results. You may have seen this idea used in other algorithms, without even recognizing it. Any algorithm that uses nominal or expected system disturbances is implicitly making the certainty-equivalence assumption. However, the certainty equivalent approximate solution can work poorly, especially if the system disturbances are chaotic, and the appropriate reaction to these disturbances drastically deviates from the nominal reaction. This is common in information centric processes such as tracking, where the underlying process evolves chaotically and the uncertainty about that process follows a similar path.

An algorithm using this simplifying assumption is shown in Algorithm 19. The algorithm begins with the current state $s \in S$ and model $\mathcal{M}$ (Line 1). It then tries all actions within subset $\bar{\mathcal{A}}$ (Line 5). For each action, the algorithm simulates one nominal sample representing the expected outcome (Line 7). After simulating the state-action, the algorithm then recursively calls itself with the new state (Line 8). The recursion terminates when the simulation reaches the end of the mission or a prescribed forward search-depth. The algorithm eventually returns the action with the highest value. This deterministic algorithm is much more efficient because it does not need to optimize over numerous samples and their subsequent outcomes.

In summary, the biggest drawback of the certainty-equivalence method is that, by simply modeling the nominal or expected trajectory, it ignores the stochastic aspect of the problem. Therefore, systems with complex-transition probability distributions will be poorly optimized when using certainty-equivalence.

D.5 Open loop feedback control

Algorithm 19 Certainty-equivalence Planning: This algorithm assumes certainty-equivalence to make the computations tractable. As opposed to Monte Carlo sampling, this algorithm bases its decision upon only simulating the single, expected state-transition. Therefore, it is faster than algorithm 17. certainty-equivalence planning is only optimal in special circumstances. Therefore, the faster speed comes at the price of optimality. $\bar{Q}$ denotes the state-action value-lookup table, which we are building. However, $Q(s', a')$ refers to the future value of taking action a' from state s', as estimated by the recursive certainty-equivalence planning algorithm. d = search depth. The computational complexity is $O(|A|^d)$.

```
CertaintyEquivalencePlanning$(s \in S, \mathcal{M} \equiv \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle, d \in \mathbb{N}) \Leftarrow \{$
if $d \equiv 0$ then
    return $0$
end if
for all $a \in \bar{\mathcal{A}} \subseteq \mathcal{A}$ do
    $\bar{Q}(s, a) \Leftarrow 0$
    $s' \Leftarrow$ NominalSimulation$(s, a, \mathcal{M})$
    $\{Q(s', a'), a'\} \Leftarrow$ CertaintyEquivalencePlanning$(s', \mathcal{M}, d - 1)$
    $\bar{Q}(s, a) \Leftarrow \bar{Q}(s, a) + (\mathcal{R}^a_{ss'} + \gamma Q(s', a'))/N$
end for
return $\max_{a \in \mathcal{A}} \bar{Q}(s, a), \arg\max_{a \in \mathcal{A}} \bar{Q}(s, a)$
$\}$
```

Open-loop Feed-back Control (OLFC)

OLFC simplifies search by limiting it to specific plans. An open loop plan consists of a sequence of actions which do not adapt to feedback from the environment. OLFC makes decisions by comparing open loop plans. However, OLFC only runs these plans for a period of time. It then replans using the same open loop plan comparison methodology. Thus, OLFC reacts to the environment stimuli, but it is not making plans that are optimized to the maximization of future contingencies. Rather, OLFC is optimized to specific hard future commitments.

One non-myopic approach to map exploration is open-loop feedback control (OLFC). To select a sensing action, OLFC generates an open-loop plan (via search) for the mission duration, which minimizes uncertainty. Many plans are evaluated this way and compared by estimated value; the best plan is chosen, and the first action in the plan is executed. After the first action is executed a new open-loop plan is generated for the remaining mission,

Algorithm 20 Open loop feedback control algorithm: To select an action, OLFC generates an open-loop plan via search, and evaluates it via N simulations. Many plans are evaluated this way and compared based upon their estimated value. The best plan is chosen, and the first action in the plan is executed. After the first action is executed a new open-loop plan is generated for the remaining mission. The process continues for the mission duration. The computational complexity of OLFC is $O(|A|^d \times N)$, where $|A|$ is the number of actions to choose from and d is the search depth; the number of time-steps into the future over which we search. $\bar{Q}$ denotes the state-action value lookup table, which we are building. However, $Q(s', a')$ refers to the future value of taking action a' from state s', as estimated by the recursive Monte Carlo planning algorithm. $d =$ search depth. The computational complexity is $O(|A|^d \times N)$.

```
OLFC($s \in S, \mathcal{M} \equiv \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle, d \in \mathbb{N}$) $\Leftarrow$ {
if $d \equiv 0$ then
  return $0$
end if
for all $a \in \bar{\mathcal{A}} \subseteq \mathcal{A}$ do
  $\bar{Q}(s, a) \Leftarrow 0$
  for $n = 1 : N \in \mathbb{N}$ do
    $s' \Leftarrow$ OLFC($s, a, \mathcal{M}$)
    $\{Q(s', a'), a'\} \Leftarrow$ MonteCarloPlanning($s', \mathcal{M}, d - 1$)
    $\bar{Q}(s, a) \Leftarrow \bar{Q}(s, a) + (\mathcal{R}^a_{ss'} + Q(s', a'))/N$
  end for
end for
return $\{\max_{a \in \mathcal{A}} \bar{Q}(s, a), \arg\max_{a \in \mathcal{A}} \bar{Q}(s, a)\}$
}
```

with the process continuing for the mission duration. Pseudocode outlining this process is shown in algorithm 20.

While OLFC is effective, it is still intractable for large real-world problems. The computational complexity of OLFC is $O(|A|^d \times N)$, where $|A|$ is the number of actions to choose from and d is the search-depth; the number of time-steps into the future over which we search. For example, suppose we have 10 actions to choose from and a mission length of 100. In this case, there would be 10^{100} possible plans to evaluate, which is intractable even with the fastest of today's computational resources. The plan outcomes are probabilistic, which makes evaluating them more expensive. As OLFC entails re-planning in real-time, these costs present an obstacle.

The final important problem with open-loop feedback control is that it compares plans, which do not react to new information, rather than policies, which do. In the next section we will discuss rollout; a method that deals with this shortcoming. In particular, when the value of the best contingent response varies widely then ignoring the new state information during the planning process is unacceptable. The policy must consider not only the system stochasticity, but also the response contingencies. One way around this problem is rollout, which we will discuss next.

Algorithm 21 Nominal Simulation - expected next state

```
NominalSimulation ($s \in S, a \in A, \mathcal{M} \equiv \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$) $\Leftarrow$ {
return $E[s'|s,a]$
}
```

Algorithm 22 Nominal Simulation - most likely next state

```
NominalSimulation ($s \in S, a \in A, \mathcal{M} \equiv \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$) $\Leftarrow$ {
return $\arg\max_{s' \in S} \mathcal{P}^a_{ss'}|s, a$
}
```

D.6 Rollout

Section C.4 explains how suboptimal policies for real-world MDPs can be found via approximate value-iteration and represented using an approximation architecture. The suboptimality of approximate policies is typically due to the representational limitations of the value function architecture. Fortunately, the value function estimate and the related action choice can be improved at runtime using rollout. Rollout is a way to evaluate small changes in a policy.

Rollout was originally introduced by Tesauro [1995] as a Monte Carlo technique for off-line state-action valuation. However, rollout can also be used at runtime for policy-extraction, which is the process of extracting the policy from the value-function. Given a value-function, rollout determines the best action for the situation. While the value-function is learned offline, rollout policy-extraction is performed at runtime. Rollout should be used instead of OLFC or certainty-equivalence when it is important to properly evaluate future plan contingencies. However, rollout requires a base policy to work.

For the full details of rollout, see Bertsekas [2007]. Here we will briefly discuss the origin and applications of rollout, typical policy extraction versus rollout policy-extraction, and improved methods of blending offline and online computation.

D.6.1 History and Intuition of Rollout

The term "rollout" comes from the backgammon community, where a playing strategy can be analyzed by averaging the results of multiple games starting from a given board position (Woolsey [1991]). The expected win-rate of a strategy from a particular board position is the state-action value or Q value. Rollout is the preferred method of comparing two strategies, such that backgammon experts use it as a justification for strategy change.

Rollout is sometimes used to compare two playing strategies that only differ by the current state-action. In other words, the current actions will be different but all future actions will use a common strategy. For example, suppose you have a strategy that you use, but you would like to test the wisdom of that strategy for the current situation. To do this, we would rollout each available action from the current board position, but strictly follow the original policy thereafter. Thus, for one particular rollout you would use the alternate action followed by the base strategy. In a backgammon rollout, you would literally roll the dice and execute this alternate strategy many, many times. The rollout results are computed by averaging the outcome of each game.

If we compare via rollout all possible actions from a given state and select the best one (Bertsekas and Castanon [1999]), the expected outcome is always better than (or at least as good as) using the base policy. Therefore, it stands to reason that choosing the action at runtime via rollout would be better than the traditional policy-extraction method.

D.6.2 Typical policy-extraction versus rollout algorithm

Typically, we determine the policy at runtime by finding the control action a that maximizes the dynamic programming equation C.1, with respect to our approximate value function V_s. In other words, we simulate a one-step lookahead, and use the value function in the next state to estimate future rewards. Performing this one-step lookahead for each possible action determines the state-action values. Thus, we select and execute the action with the highest state-action value in accordance with the following equation.

$$\pi(s) = \arg\max_a \sum_{s' \in S} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')] \tag{D.6}$$

Algorithm 23 shows the steps needed to perform this operation. In line 1 we pass in the current state, value function and model. Line 2 loops through all actions in order to evaluate each of them. Line 3 resets the state-action lookup value to zero, that we are about to reevaluate. Line 4 loops through all possible next states. Line 5 adds in the weighted value of state s' into the new state-action value estimate lookup table. When done reevaluating all actions, we return the highest valued action in line 9.

Although Algorithm 23 represents, passes in value function V, if the offline policy is represented as a state-action value function Q, then the procedure is simpler still as shown in Algorithm 24. Algorithm 24 shows the steps needed to perform this operation. In line 1 we pass in the current state, approximate state-action value function and model. Line 2 and 3 initializes variables to store the best action and its value. Line 4 loops through all actions in order to compare each of them. Line 5 compares the approximated state-action value of action a to the best found so far. If it is better, then the best action is replaced with action a. When all is said and done, the best action found is returned at line 9.

Algorithm 23 Bellman Policy Extraction: Typical value function policy-extraction using V. $\bar{Q}$ is a state-action value lookup table where we store our intermediate and final state-action values based on the one-step lookahead.

```
BellmanPolicyExtraction($s \in S, V, \mathcal{M} \equiv \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$) $\Leftarrow$ {
for all $a \in A$ do
    $\bar{Q}(s,a) \leftarrow 0$
    for all $s' \in S$ do
        $\bar{Q}(s,a) \leftarrow \bar{Q}(s,a) + \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]$
    end for
end for
return $\arg\max_{a \in \mathcal{A}} \bar{Q}(s,a)$
}
```

Algorithm 24 Typical Q-function policy-extraction: $\widehat{Q}$ stands for an approximation architecture representation of the state-action value function.

```
QPolicyExtraction($s \in S, \widehat{Q}, \mathcal{M} \equiv \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$) $\Leftarrow$ {
$bestQvalue \Leftarrow -\infty$
$bestAction \Leftarrow null$
for all $a \in \mathcal{A}$ do
    if $\widehat{Q}(s,a) > bestQvalue$ then
        $bestAction \Leftarrow a$
    end if
end for
return $bestAction$
}
```

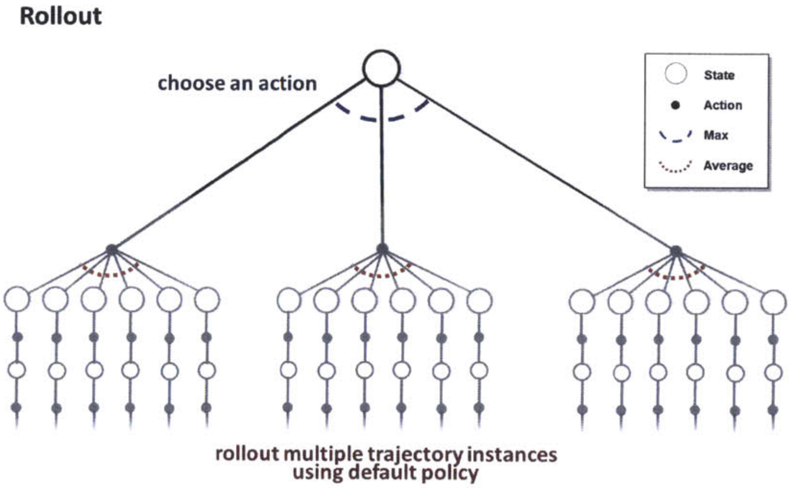


Figure D-3: Rollout re-evaluates every action at the first level, by rolling out future actions in a Monte Carlo fashion, using the base policy to choose actions. Notice that all three actions are evaluated at the starting state. But thereafter, only one action is evaluated. That is the base policy default action. Multiple instances of an entire trajectory are simulated and averaged. Rollout considers each action and simulates where the approximate policy will lead.

While, Algorithm 24 relies entirely on a state-action value approximation provided by $\widehat{Q}(s, a)$, Algorithm 23 performs a single step improvement upon the value estimate provided by V. We can take this a step further by employing *rollout*. The rollout algorithm selects the current control action that maximizes the expected outcome with respect to a multi-stage simulation of the base policy. A multi-stage rollout is performed for each of the current action choices as shown in Figure D-3. The rollout method augments the base policy on the first step by branching and maximizing over the expected outcome of the base policy for the remaining mission horizon. The base policy for the remaining mission horizon is evaluated by simulating that trajectory multiple times under the base policy, again as shown in Figure D-3 . In other words, rollout maximizes over the initial action-choice based on a Monte-Carlo evaluation of the base policy for the remainder of the mission. By doing so, rollout makes better control-action selection than the one-step look-ahead method. According to Bertsekas and Castanon [1999] rollout tends to make better action-selections than the underlying value function. Therefore, it stands to reason that rollout is a better policy extraction method.

Algorithm 25 shows the steps needed to perform this operation. In line 1 we pass in the current state, approximate state-action value function and model. Line 2 and 3 initializes variables to store the best action and its value. Line 4 loops through all actions in order to compare each of them. Line 5 compares the approximated state-action value of action a to the best found so far. If it is better, then the best action is replaced with action a. When all is said and done, the best action found is returned at line 9.

Algorithm 25 Rollout: $\bar{Q}$ denotes the state-action value lookup-table, which we are building.

```
 1: CompareRollouts($s \in S, \mathcal{M} \equiv \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle, \pi, T \equiv terminalstate) \Leftarrow \{$
 2: for all $a \in \bar{\mathcal{A}} \subseteq \mathcal{A}$ do
 3:    $\bar{Q}(s, a) \Leftarrow 0$
 4:    for $n = 1 : N$ do
 5:       $\bar{Q}(s, a) \Leftarrow \bar{Q}(s, a)+$Rollout$(s, \mathcal{M}, \pi, a)/\boldsymbol{N}$
 6:    end for
 7: end for
 8: return $\{\max_{a \in \mathcal{A}} \bar{Q}(s, a), \arg\max_{a \in \mathcal{A}} \bar{Q}(s, a)\}$
 9: $\}$
10: Rollout($s \in S, \mathcal{M} \equiv \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle, \pi, a \in \mathcal{A}, T) \Leftarrow \{$
11: $s' \Leftarrow$ MonteCarloSimulation$(s, a, \mathcal{M})$
12: if $s' \in T$ then
13:    return $\mathcal{R}^{a}_{ss'}$
14: end if
15: return $\mathcal{R}^{a}_{ss'}+$Rollout$(s', \mathcal{M}, \pi, \pi(s'), T)$
16: $\}$
```

D.6.3 Time-complexity of rollout

Rollout can viably improve upon an offline policy in the frenetic time-constrained runtime environment because it tries to find the best action for the current situation only, which is easier than finding the best action for every situation.

The runtime of rollout depends on A, the number of actions you will consider; N, the number of rollouts (trajectory samples) per action; and d, the average length or depth of a trajectory. The O notation time-complexity of rollout is: $O(ANd)$

The runtime of typical value based policy-extraction depends on A, the number of actions you will consider and N, the number of one-step look-ahead samples. The worst-case time-complexity of typical value-based policy-extraction is: $O(AN)$

D.7 Nested Rollout

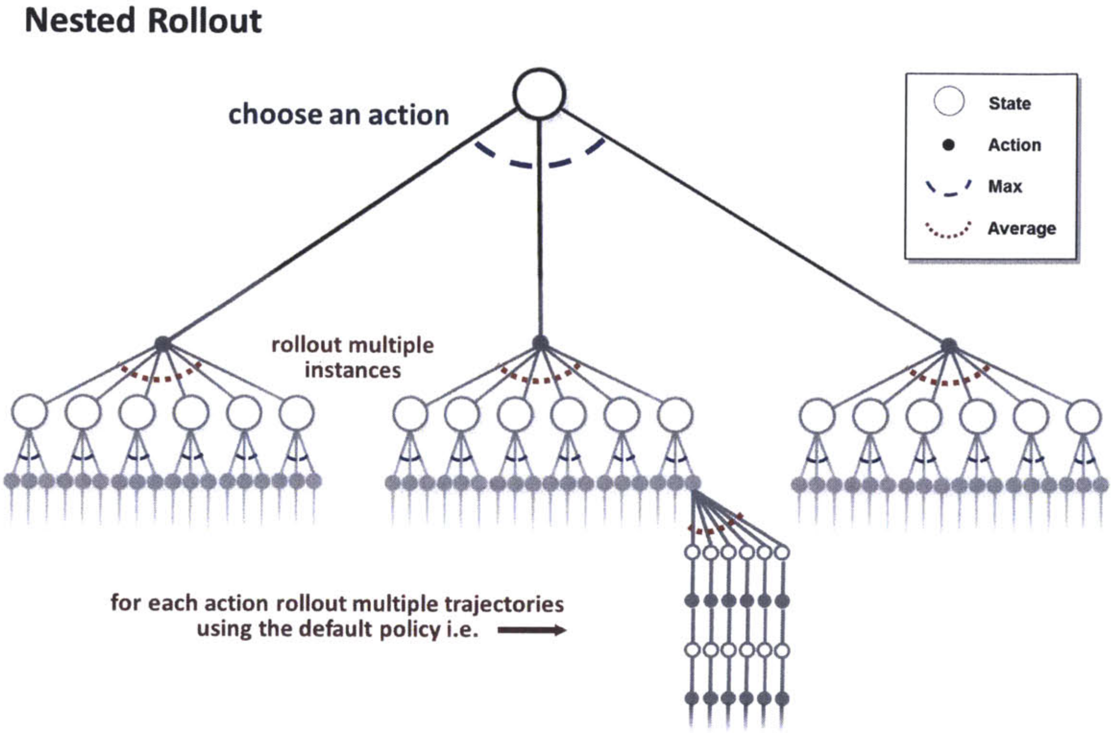

Nested Rollout
choose an action
State
Action
Max
Average
rollout multiple instances
for each action rollout multiple trajectories using the default policy i.e.

Figure D-4: Two-level nested rollout re-evaluates every action at the first and second level, by rolling out future actions in a Monte Carlo fashion. Thus all potential action choices are evaluated for the current state and the subsequent state. Meanwhile, the rollout process uses the base policy to choose actions for all but the first and second encountered states in a trajectory. In rollout, multiple instances of an entire trajectory are stochastically simulated and averaged. Thus, at the first and second level we consider every action, but thereafter follow the base policy.

D.7.1 Standard Rollout

The point of rollout is to compare polices that differ only in their initial action choice. The comparison requires a Monte Carlo analysis of each policy. The term "rollout" comes from the backgammon community where it is sometimes used to compare two playing strategies. The two backgammon game playing strategies being computed are in fact the same with the exception that one of the strategies is employing a different move at a particular game configuration. The comparison starts at that configuration of interest. To do this, you would perform Monte Carlo analysis on two actions from the current board position, but strictly follow the original policy thereafter. The two different moves are employed and the remaining game is "rolled-out" under the predominant incumbent strategy. By doing so, we can figure out if the new move is an improvement or not.

In a backgammon rollout, you would literally roll the dice and execute this alternate strategy many, many times. The rollout results are computed by averaging the outcome of each game. Rollout is the preferred method of comparing two strategies, such that backgammon experts use it as a justification for strategy change.

D.7.2 Rollout based policy extraction

In the MDP framework, we will often learn a value function offline and determine the policy at runtime by maximizing the dynamic programming equation. If we compare via rollout all possible actions from a given state and select the best one (Bertsekas and Castanon [1999]), the expected outcome is always better than (or at least as good as) using the base policy. Therefore, it stands to reason that choosing the action at runtime via rollout would be an improvement over the traditional policy-extraction method.

D.7.3 The nested rollout extension

Rollout is a bit like a search algorithm except that it only branches on actions at the very first level. In that sense it samples from the policy tree by severely limiting its policy choices, in addition to settling for Monte Carlo sampling of each policy instead of a complete evaluation. These restrictions make rollout efficient but limited. Nested rollout extends the rollout concept by allowing branching at multiple levels (see Rosin [2011]). For example, nested rollout may branch at two or three levels. This makes nested rollout more costly but also more capable. The secret is to make the right trade-off between cost and capability. Details of this algorithm are presented here.

Searching Solitaire in Real Time Bjarnason *et al.* [2007] provides a nested rollout algorithm to amplify the effectiveness of a heuristic based decision. Maximizing over the rollout of each state-action is a form of policy iteration. Policy iteration generates a sequence of improved policies (Proposition 2.4 page 30 of Bertsekas and Tsitsiklis [1996a]). Consequently, a rollout policy is guaranteed to be equally good or better than the base policy (page 267 of Bertsekas and Tsitsiklis [1996a]). However, unlike a single stage rollout, the multi-stage algorithm is not guaranteed to deliver a better or even equivalent policy. Nevertheless, multistage rollout performs well in practice.

The recursive formulation allows a variable branching factor at each stage. The modified search method also incorporates move sequences (macros) and caches visited states, similar to Kocsis and Szepesvari [2006].

D.7.4 Picture Description

Rollout re-evaluates every action at the first level, by rolling out future actions in a Monte Carlo fashion in that multiple instances of an entire trajectory are stochastically simulated and averaged. The rollout process uses the base policy to choose actions, thus all three potential action choices are evaluated from the current state.

Yet, aside from the initial actions, thereafter only one action is evaluated, which is the base policy default action. Consequently, these subsequent base policy decisions are not necessarily optimal, just as the initial decision may not be optimal. After all, that is the justification for the reevaluation.

One way to improve upon this conundrum is to do a second level of rollouts. In other words, at not just the current state, but also at the subsequent resulting state, each available action is considered and reevaluated via rollout. Thus, at the first and second level we consider every action, but thereafter follow the base policy. This process is depicted in Figure D-4.

It is logical that this process is better than standard rollout. Since it is better, why not do this at even more levels. The opposing reason is that although deeper action branching is better, it can get expensive. Therefore, choosing an appropriate depth to do this is important.

D.7.5 Algorithm Walkthrough

Algorithm 26 Nested Rollout: N-level nested rollout branches on all actions in the top N levels then re-evaluates these combinations by rolling out future actions $\pi(s)$ in a Monte Carlo fashion. The rollout process uses the base policy to choose actions for all but the top N encountered states in a trajectory. In rollout, multiple instances of an entire trajectory are stochastically simulated and averaged. $\bar{Q}$ denotes the state-action value lookup-table, which we are building.

```
 1: CompareNestedRollouts($s \in S, \mathcal{M}, \pi, T, N, nestDepth$) $\Leftarrow$ {
 2: for all $a \in \mathcal{A}$ do
 3:    $\bar{Q}(s,a) \Leftarrow 0$
 4:    for $n = 1 : N$ do
 5:       if $nestDepth > 0$ then
 6:          $s' \Leftarrow$ MonteCarloSimulation($s, a, \mathcal{M}$)
 7:          $\bar{Q}(s,a) \Leftarrow \bar{Q}(s,a) + \mathcal{R}^a_{ss'} +$CompareNestedRollouts$(s', \mathcal{M}, \pi, T, N, nestDepth)/N$
 8:       else
 9:          $\bar{Q}(s,a) \Leftarrow \bar{Q}(s,a) +$Rollout$(s, \mathcal{M}, \pi, a, T)/N$
10:       end if
11:    end for
12: end for
13: return {$\max_{a \in \mathcal{A}} \bar{Q}(s,a), \arg\max_{a \in \mathcal{A}} \bar{Q}(s,a)$}
14: }
15: Rollout($s \in S, \mathcal{M} \equiv \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle, \pi, a \in \mathcal{A}, T$) $\Leftarrow$ {
16: $s' \Leftarrow$ MonteCarloSimulation($s, a, \mathcal{M}$)
17: if $s' \in T$ then
18:    return $\mathcal{R}^a_{ss'}$
19: end if
20: return $\mathcal{R}^a_{ss'} +$Rollout$(s', \mathcal{M}, \pi, \pi(s'), T)$
21: }
```

Algorithm 26 shows pseudocode for the nested rollout algorithm. In line 1 we pass in the current state, system model, policy, terminal state criteria, sample size and nest depth. The variable *nestDepth* reflects the number of times that the standard rollout algorithm will be 'nested' inside itself. The *nestDepth* is also the depth to which the **CompareNestedRollouts** function will branch on and compare all available actions. Line 2 loops through all actions in order to compare each of them. Line 3 loops through the initiation of N Monte Carlo trajectory samples. These samples may lead to another nested rollout or a simple rollout evaluation of the trajectory given policy π. Line 5 chooses between calling nested rollout, if the depth has not yet been reached, and standard rollout.

Line 6 simulates the next state. Line 7 adds the instant reward to the lookup table and calls **CompareNestedRollouts**. If the required nesting depth has been exceeded, then line 9 calls standard rollout. Line 13 compares and returns the best action and its value.

Line 15 through 21 include the subroutine **Rollout**. In line 15 we pass in the current state, model, policy, current action and terminal state criteria. Line 16 simulates the next state. Line 17 determines if we have reached a terminal state. If we have reached a terminal state, line 18 returns the instant reward. If we have not reached a terminal state, line 20 recursively calls **Rollout**. Upon return of the **Rollout** function, line 20 adds the result to the instant rewards and returns the sum.

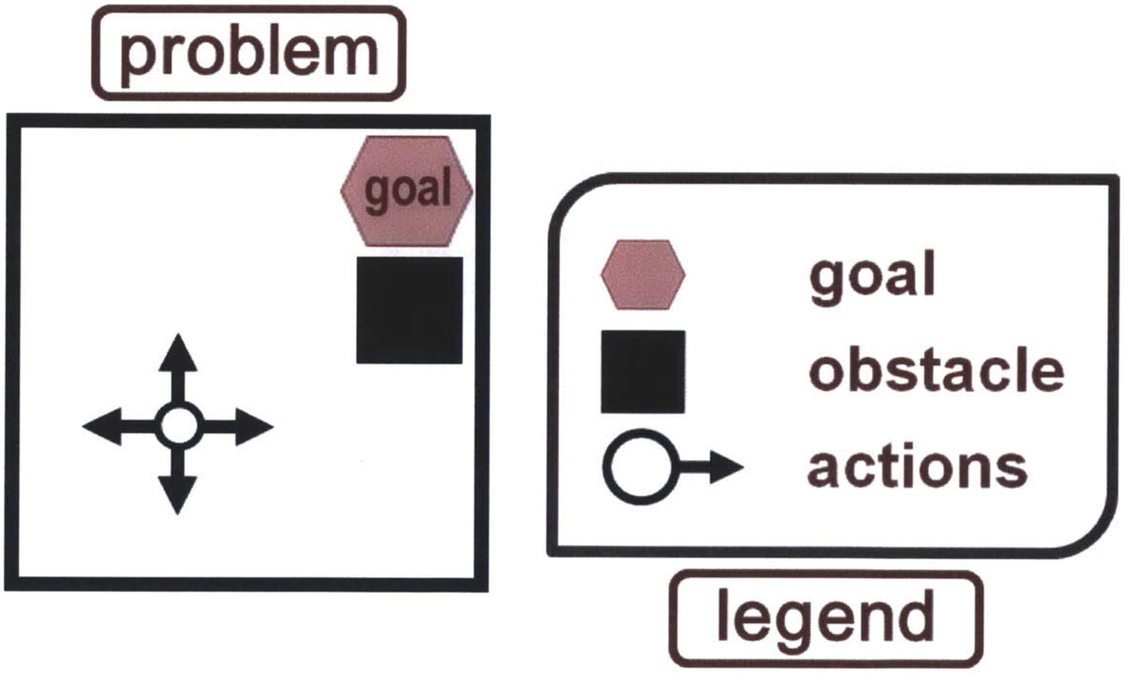


Figure D-5: Example problem : You want to create this value function that somehow guides you to the goal. The state space is continuous, thus infinite in size. We could discretize the space, but that would be too hard. We need to approximate a value function with a low order function. However, the resulting policy is imperfect.

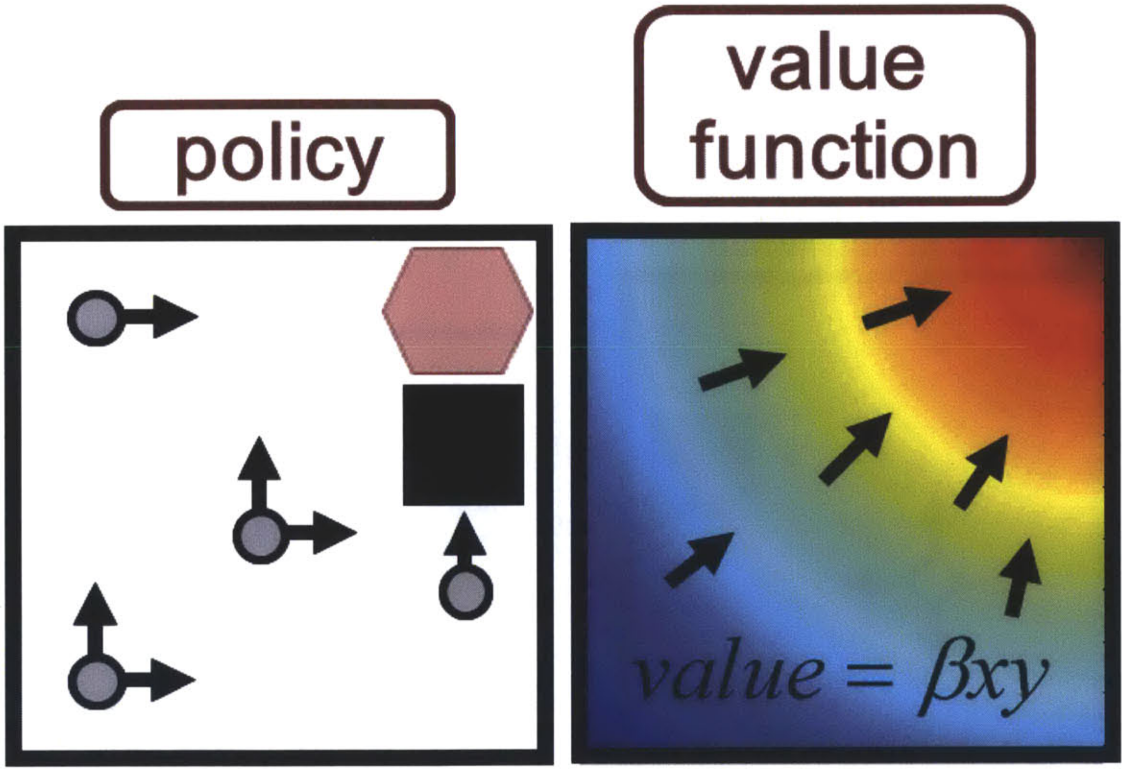


Figure D-6: Value function based policy (approximate) : You want to create this value function that somehow guides you to the goal. The state space is continuous, thus infinite in size. We could discretize the space, but that would be too hard. We need to approximate a value function with a low order function. However, the resulting policy is imperfect.

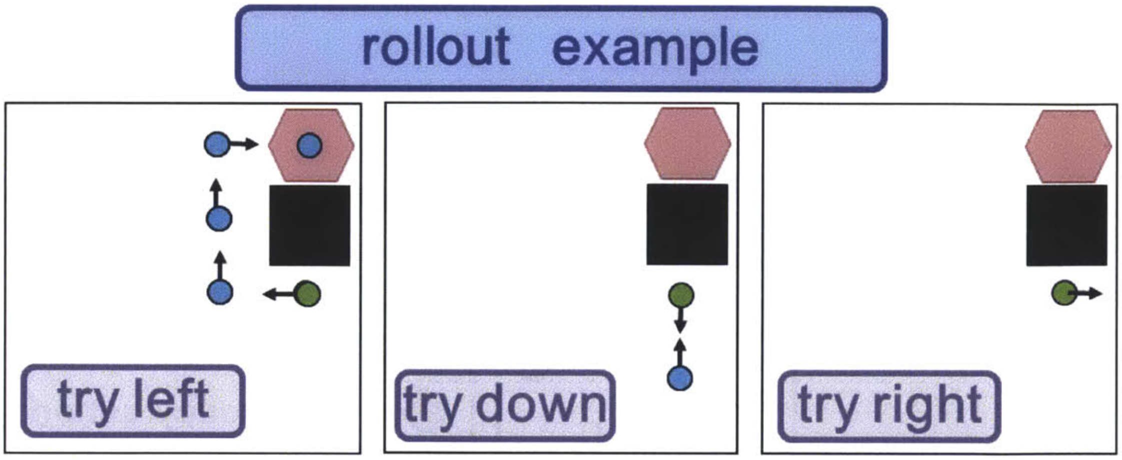


Figure D-7: Rollout example: Standard rollout is depicted above our simple vehicle guidance task. Three actions are tried and compared. The left action is tried and the base policy is followed thereafter. The same is done for the down and right actions. For a deterministic problem, each action is simulated once. For a stochastic problem each action must be simulated enough times to acquire a low variance average value estimate.

The rollout algorithm presupposes that we have a value function that guides us to the goal. Our approximate value function is helpful but imperfect. To compensate for the imperfect policy, we can re-simulate our policy for each current action. This simulating based rollout provides a more accurate value estimate, and consequently a better decision. Kit Woolsey introduced rollout as a statistical method for quantitatively determining the best move, in which candidate position is played to completion thousands of times, with different random dice sequences.

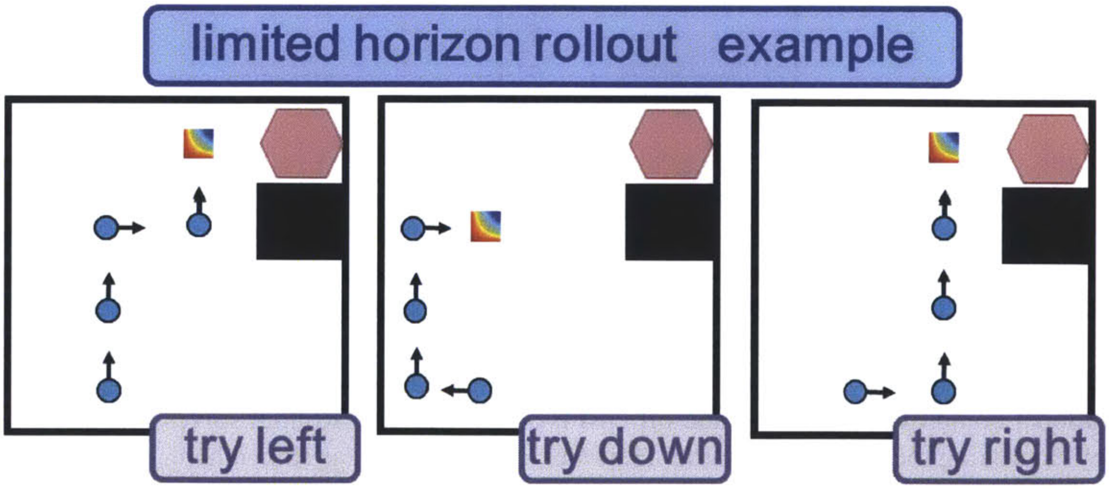


Figure D-8: Limited Horizon Rollout : We need to approximate a value function with a low order function. We compute Q-values by trying each move (Left, Right and Down). This involves simulating n-steps using the base policy. But the remaining cost-to-go is estimated using the value Function, or q-value function.

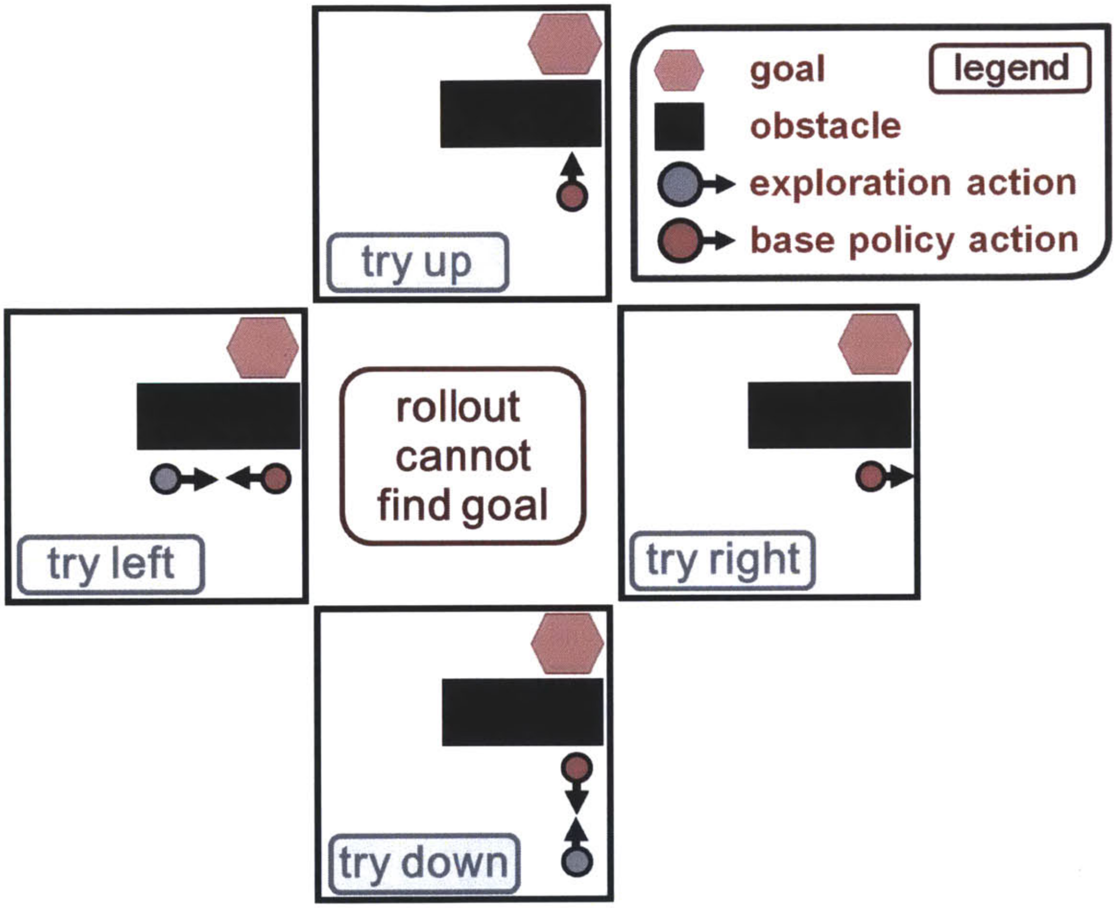
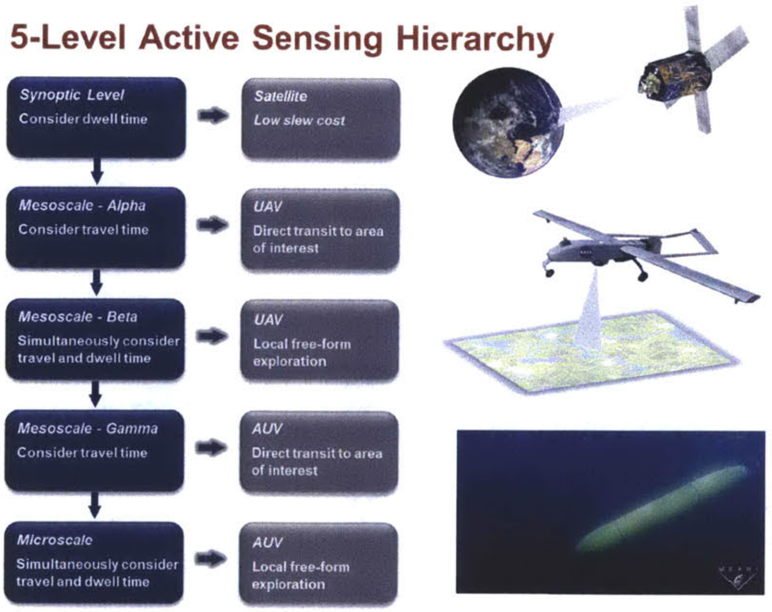


Figure D-9: Rollout cannot find goal: In the above example, 1-step rollout does not improve policy and cannot find goal. In some cases, one step rollout does not work. The base policy is based upon the approximate value function. It points the agent directly toward the goal, regardless of obstacles. The rollout algorithm attempts to go left, right and down, followed by rolling out the base policy. When the obstacle was small, rollout found its way around, with the help of the base policy. In any arbitrary problem, a value function approximates some simpler version of the problem. This is necessary to reduce representational complexity. But the trade-off cost is a worse policy. Rollout solves this dilemma by improving the policy at runtime. However, in some cases, as shown here, the rollout of a base policy is still unsatisfactory.

E An alternative 5-level Hierarchy system

The 3-level hierarchy discussed in section 5.1.1 is the basis from which we work in this thesis. However, an alternative 5-level hierarchy warrants mentioning because it makes sense for some applications. For example, suppose we wish to monitor an evolving ocean phenomenon. We can monitor it globally using a satellite and locally using an AUV. However, we could benefit from a faster moving in-situ sensor platform, such as a small unmanned air vehicle (UAV). The concept is that a team of UAVs could be deployed to followup on the information received from the satellite. The team is deployed to a large (mesoscale-alpha) area and each UAV is tasked with a set of locations to survey. The UAV data will be more accurate than the satellite data due to the closer proximity and focus. Information gained from this process is used to task a team of AUVs to a still smaller (mesoscale-gamma) area. Figure E-2 depicts this architecture. The main advantage of the 5-level hierarchy is that we can use the faster speeds of the UAVs to collect an intermediate quality of data which has a broader geographic range.



5-Level Active Sensing Hierarchy

Synoptic Level Consider dwell time → *Satellite* *Low slew cost*

Mesoscale - Alpha Consider travel time → *UAV* Direct transit to area of interest

Mesoscale - Beta Simultaneously consider travel and dwell time → *UAV* Local free-form exploration

Mesoscale - Gamma Consider travel time → *AUV* Direct transit to area of interest

Microscale Simultaneously consider travel and dwell time → *AUV* Local free-form exploration

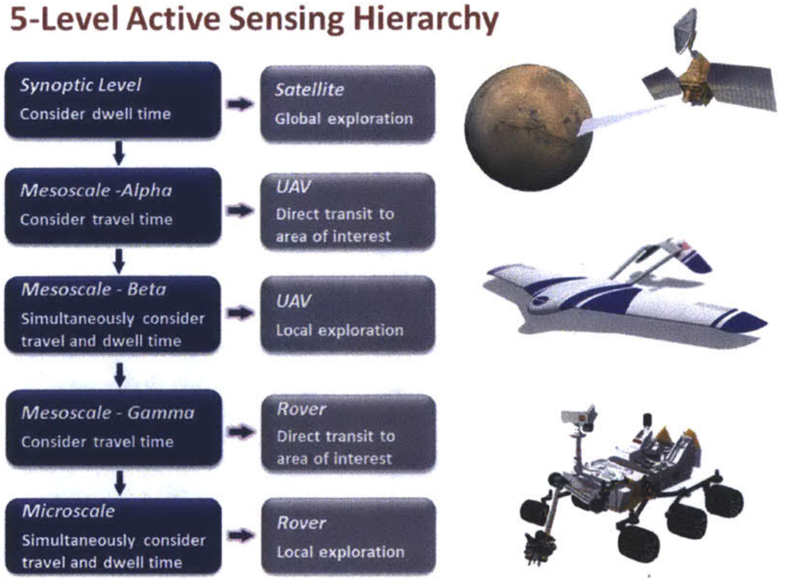
Figure E-1: 5-level Adaptive sensing hierarchy.

Scale Terms

The synoptic-meso-micro meteorlogical scale system is a horizontal length scale designed for the study of weather systems. Each level deals with phenomena of a certain size, listed below. This definition is a consistent amalgamation of various references.

- Synoptic : $\sim$ 1000 Miles (weather fronts and pressure systems)
- Meso : $\sim$ 1 miles $\leftrightarrow$ 1000 miles
  (storm-scale cumulus systems such as sea breezes and squall lines)
  - Alpha : $\sim$ 100$\leftrightarrow$ 1000 miles (squall lines and tropical cyclones)
  - Beta $\sim$ 10 miles $\leftrightarrow$ 100 miles (sea breezes and lake effect snow storms)
  - Gamma $\sim$ 1 miles $\leftrightarrow$ 10 miles
    (thunderstorm convection and complex terrain flows)
- Micro $\leq$ 1 mile (cloud ”puffs” and other small cloud features)

See Berg *et al.*, Glickman [1972], Fujita [1986] and Orlanski [1975].



5-Level Active Sensing Hierarchy

Synoptic Level Consider dwell time → *Satellite* Global exploration

Mesoscale -Alpha Consider travel time → *UAV* Direct transit to area of interest

Mesoscale - Beta Simultaneously consider travel and dwell time → *UAV* Local exploration

Mesoscale - Gamma Consider travel time → *Rover* Direct transit to area of interest

Microscale Simultaneously consider travel and dwell time → *Rover* Local exploration

Figure E-2: Space Exploration 5-level Adaptive sensing hierarchy.

F Abstract Bayesian Sensor Model : Road Detection Example

This application, which we described in detail in "Active Detection of Drivable Surfaces in Support of Robotic Disaster Relief Missions" (Wang *et al.* [2013]), builds an abstract Bayesian sensor model (ABSM) for detecting roads. In this example, the model is used for a disaster relief or search and rescue mission, where the active sensing algorithm seeks to find a safe ground-vehicle path to the victims, via a roadway or other traversable surface. We will describe the inside workings of the road detection ABSM below. However, it is important to remember that an abstract Bayesian sensor model maps real world sensor data to a posterior prediction about a phenomena of interest. A key aspect of the ABSM is that it produces a prediction distribution, thereby allowing us to integrate it into our prior belief via a Bayesian update. The ABSM only presents an abstract interface to the active sensing algorithm. The details of how it generates the posterior prediction distribution from the data are completely compartmentalized away from the planning agent. Therefore, a planning agent can be used on any active sensing application if a proper ABSM has been built for it. The ABSM can be applied to a range of active sensing tasks, such as finding water on Mars.

In our road detection tests, our multi-step method removed many spurious objects that were initially considered to be roads. With one pass of our sensor, the algorithm will definitively detect a road 20% of the time with a 40% false alarm rate. Although the sensor did sometimes lose portions of actual road, this cost is minimal compared to the potential danger of identifying obstacles such as rooftops and houses as open terrain. Our algorithm conveys the certainty level of the prediction, so that it can be integrated into the current belief via a Bayesian update. Therefore, multiple collections may be planning and integrated together.

F.1 Active Detection of Drivable Surfaces in Support of Robotic Disaster Relief Missions

Background

The use of unmanned vehicles has been growing over the past few decades, with various applications ranging from scientific exploration (Bush *et al.* [2008b]) to industrial automation (Wurman *et al.* [2008]). In order to relieve the burden on human operators, the most obvious solution is to use a fully autonomous vehicle that requires no operators; however, current models lack the algorithms to execute complicated maneuvers that human-operated unmanned vehicles can, limiting their usage in a search and rescue situation (Quigley [2004]).

Semi-autonomous vehicles provide a compromise between these two approaches to unmanned vehicles. Under this paradigm, vehicles mainly operate without the help of a human operator when it is not necessary. However, during risky maneuvers, a human takes direct control of the vehicle. This method allows a single pilot to manage several unmanned vehicles simultaneously while still maintaining a higher success rate than fully-autonomous counterparts. In this section, we present a semi-autonomous network combining the resources of humans, unmanned air vehicles (UAVs), and unmanned ground vehicles (UGVs) in order to conduct various missions such as search-and-rescue, natural disaster relief, and possibly extraterrestrial exploration. The use of both UAVs and UGVs combines the breadth of terrain UAVs can survey while maintaining the ability of UGVs to provide direct aid and capture higher quality images at a closer range than UAVs can Quigley [2006].

Outline of Semi-Autonomous Operation

We propose the following approach. Cameras attached to UAVs capture images of the terrain below and transmit the images to a nearby computing cluster, which will calculate what areas of the terrain can be traversed by the UGV being deployed. To do so, it will use an algorithm, fully developed and proposed in Wang *et al.* [2013], that estimates the position of roads using color family normalization and edge detection. Our algorithm takes advantage of the continuous nature of roads by using super-pixelation, a Markov random field, and Gaussian process machine learning (GPML). This data is then transmitted to UGVs that use the information for path-finding, as illustrated in Figure F-1.

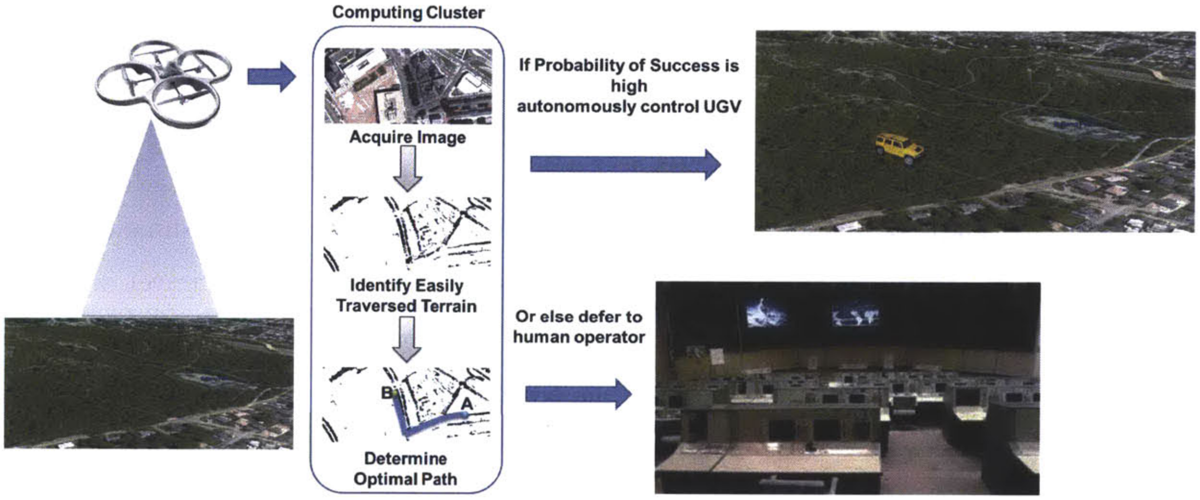

Computing Cluster
Acquire Image
Identify Easily Traversed Terrain
B
A
Determine Optimal Path
If Probability of Success is high autonomously control UGV
Or else defer to human operator

Figure F-1: Semi-autonomous network: A quadrotor, shown in the upper-left corner, takes aerial images of the terrain below and transmits these images to a computing cluster which identifies easily traversable terrain and computes the optimal paths for the UGVs to take. If the risk of traversing the path reaches a threshold point, the control of the UGV is handed over to a human operator. Otherwise, the UGV is autonomously controlled.

In addition to calculating the traversability of each pixel, our algorithm calculates the risk of traversing the pixel. When the calculated risk reaches a certain threshold, a human operator is instructed to directly control the UGV, therefore increasing the probability of the vehicle reaching its destination safely, proceeding off the assumption that humans typically are more skilled with the specific mission. Once the high-risk area is passed, the unmanned vehicle then continues on to its destination without human guidance.

Specific Aims

Our specific goal was to develop an algorithm that can identify roads or any flat surface from an aerial image and characterize the confidence level of our analysis. This is initially done using color analysis, edge effects, and object recognition. We shall refer to this as the color analysis. Unfortunately, this method is unreliable for several reasons. Color analysis often fails because many objects, like rooftops, share the same color as roads. Edge effect algorithms, which detect the surrounding borders of roads, often find the edges of rooftops, trees, trash dumpsters, etc. Object recognition looks for features that distinguish roads from other objects, such as rooftops. However, since the shape of roads vary widely, they have very few universal features.

The guiding principle behind this algorithm is that one single pixel may contain a high amount of noise. However, by taking multiple pixels that we believe to be part of a road, the collective noise is reduced. We hypothesized that our color analysis fails to account for the central design pattern that the vast majority of roads go to a destination and are therefore continuous segments. There are very few, if any, roads that travel only a few feet, for example. In order to take advantage of this pattern, we modeled the system as a Markov random field (MRF) where the probability that an individual pixel is part of a road is not only dependent on its own characteristics, but also on its surrounding pixels (Li [1995]). Then the color analysis techniques discussed above were utilized as prior information, which is refined further with a process known as loopy belief propagation (Felzenbath and Huttenlocher [2006]). We then use machine learning to finalize our results. In the following sections, we will outline our pipeline of analysis and demonstrate its reliable performance. Our discussion of the pipeline will be brief, but provided additional focus on the Gaussian Process step, as it is particularly important to the idea of the abstract Bayesian sensor model. Please refer to Wang *et al.* [2013] for a comprehensive review of the image processing pipeline.

F.2 Theory

Overview

Our strategy was to define a matrix of Bernoulli random variables that take on values of either true or false. Each random variable represents whether its corresponding pixel is part of a road. This probabilistic approach allows us to evaluate the relative uncertainties that our algorithm produces. To generate rough estimates for the marginal probabilities, we first run a color analysis, edge effect, machine learning, and object recognition algorithm on an aerial image.

We extract edge features using the HOG algorithm (Dalal [2005]) and average pixel counts. Color and edge analysis includes supervised statistical analysis of key color and edge features using an SVM (support vector machine). The color features include normalized and Gaussian smoothed color vectors, where a color vector has three parts. These features help us to detect road edges and deal with varying hues of gray. The image is then downsized by super-pixelation. The results are improved by using belief propagation that takes advantage of the underlying MRF structure to update the posterior probabilities of all the pixels. The image is then up-sized to the original resolution, at which point belief propagation is used to refine the borders of the roads. We found that this approach is not only faster than simply running belief propagation on the original color analysis, but also significantly more accurate. Finally, we run a Gaussian process machine learning (GPML) algorithm, training the algorithm by pre-labeling the roads in some of the images. The GPML algorithm returns a Gaussian distribution for each pixel stating the probability that the pixel is part of a road and the certainty of that probability. This overall process is illustrated in Figure F-2.

Gaussian Process Machine Learning

Gaussian process machine learning (GPML) is a form of regression analysis that we employed to further refine our results from the variable zoom (Rasmussen [2004]). GPML takes in *training data* and *target data* and generates a Gaussian process to model their relationship. In our scenario, our training data consisted of the output from variable zoom and the target data was derived by hand-labeling all of the roads in an image. A Gaussian process is defined as

$$\mathbf{F} \sim \mathcal{GP}(\mu, \sigma)$$

where m and n are functions that determine the mean and variance of a Gaussian distribution respectively (Rasmussen [2004]). In the case of this project, the input variable of μ and σ is the vector $\mathbf{x}$, consisting of a pixel and all of its neighbors' probabilities of being

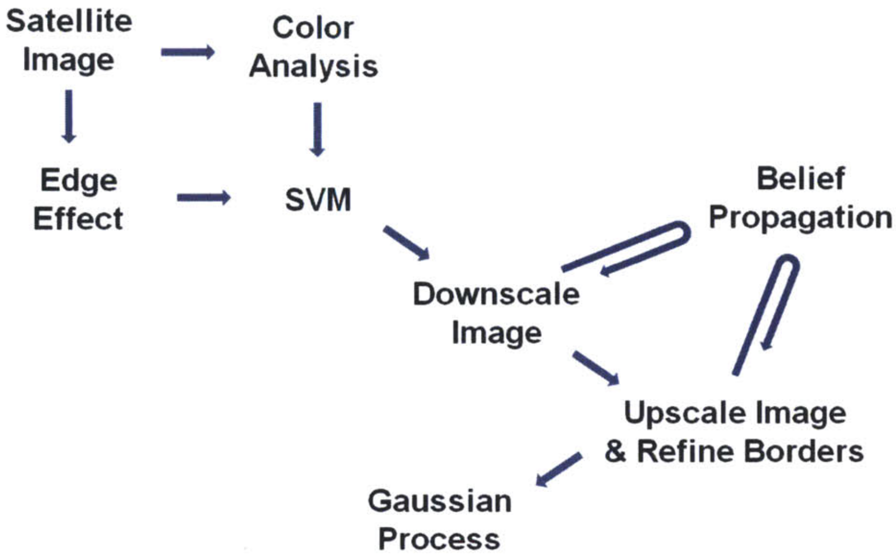


Figure F-2: To perform our road analysis, we first run a color analysis algorithm. We downscale the result. Subsequently, we improve our analysis by applying belief propagation to our downscaled image. We refine the borders of our roads by upscaling the result and running the belief propagation on the borders of identified roads. We complete our analysis using Gaussian process machine learning.

part of a road, as determined by the variable zoom. Symbolically,

$$\mathbf{F}(\mathbf{x}) = \mathbf{f} \sim \mathcal{N}(\mu, \sigma)$$

Details of the GPML are omitted for brevity. The GPML algorithm finds optimal parameters that maximize the log-likelihood function, which characterizes the probability that the relationship found by the Gaussian process is able to model reality given the data from the variable zoom and the location of the actual roads (found by hand-labeling the images). We can use this best-fit function to predict the target data (location of roads) of other images (assuming that the relationship between the training data and the target data remains relatively constant). Because a Gaussian process returns a Gaussian distribution for each pixel and its neighbors, important information about the certainty of the prediction is also returned. This is diagrammed schematically in Figure F-3.

This makes the GPML useful for our algorithm, because the MRF is often uncertain about whether or not a pixel is a road, possibly due to obstructions and low image quality

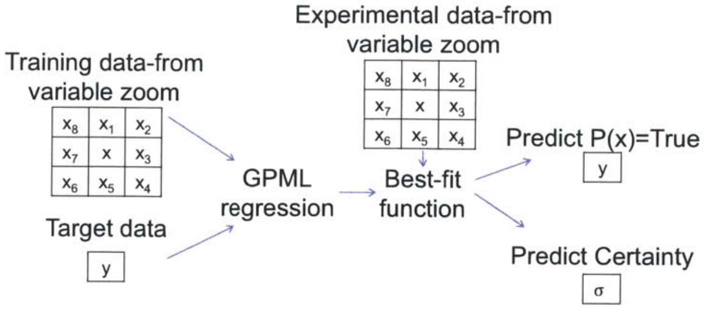


Figure F-3: The GPML regression takes in the output of the variable zoom on an image as a training set and the same image with all of the roads hand-labeled as a target set. Note that the training set for one pixel consists of that pixel and its neighboring elements. The algorithm then creates a best-fit function that models the relationship between the two. With this function, we can input variable zoom data from another image and the best-fit function will output refined probabilities for each pixel being part of the road and the certainty of each prediction point.

in certain areas. Oftentimes, information can be supplemented by sending additional UAVs to take more photographs of the area in question, provided limited availability of UAVs. Thus, we have to make sure that we make optimal use of UAVs by only sending additional surveillance to areas where we are uncertain of the presence of a road. Since the GPML can predict the uncertainty of the MRF, we can use that data for this optimization.

In addition, GPML takes advantage of our central paradigm that roads are continuous segments that go to a destination. Since the GPML can accept multiple input variables, not only can we input a pixel's marginal probability to predict its state, but we can also use those of its surroundings to improve that prediction.

F.3 Why GP?

Applying a Gaussian process machine learning algorithm not only improved our results, but also learned and represented the uncertainty in the algorithm's output. Reflecting the uncertainty in the traversability measurement allows the belief update algorithm to treat the output abstractly, like a generic sensor model that directly measures traversability and can be used to directly update our belief about it using a Kalman filter. This idea is called an abstract Bayesian sensor model. We envision that these algorithms will soon be incorporated into autonomous search and rescue teams, laying the foundation for their eventual use in extraterrestrial exploration applications.

F.4 Results and Discussion

Color Analysis and Variable Zoom

The performance of several variants of our algorithm before the application of GPML are shown in Figure F-4. In the academic literature, the most similar study to ours used a genetic algorithm machine linear approach to finding roads in satellite images (Boggess [1996]). However, the paper compared the accuracy of their model to that of a human's performance and lacked performance details. As a resul, we are unable to compare its accuracy to the model we developed.

To quantify the performance of our algorithm, we randomly selected an image and determined the accuracy of our algorithm. Accuracy was determined in two methods, denoted as precision and recall.

$$Precision = \frac{\text{Num. of Pixels Correctly Identified as Roads}}{\text{Num. of Pixels Identified as Roads}}$$

$$Recall = \frac{\text{Num. of Pixels Correctly Identified as Roads}}{\text{Num. of Actual Road Pixels}}$$

The results of our analysis are shown in Figure F-5A. Applying variable zoom brings considerable improvement over both the color analysis and the naïve application of belief propagation to the entire image for precision. Color analysis, however, is superior in regard to recall. This is most likely because all roads are gray, so pixels that are actually part of a road would be identified correctly by the color analysis. Since a MRF tends to lower the marginal probabilities of the edges of roads, this tends to worsen the performance on recall. It should be noted, however, that a lack of precision is often more detrimental to an unmanned vehicle's performance than recall, since the consequences of traversing non-road terrain are more severe than the consequences of not traversing the entire road.

Additionally, variable zoom was approximately eight times faster than naïve belief

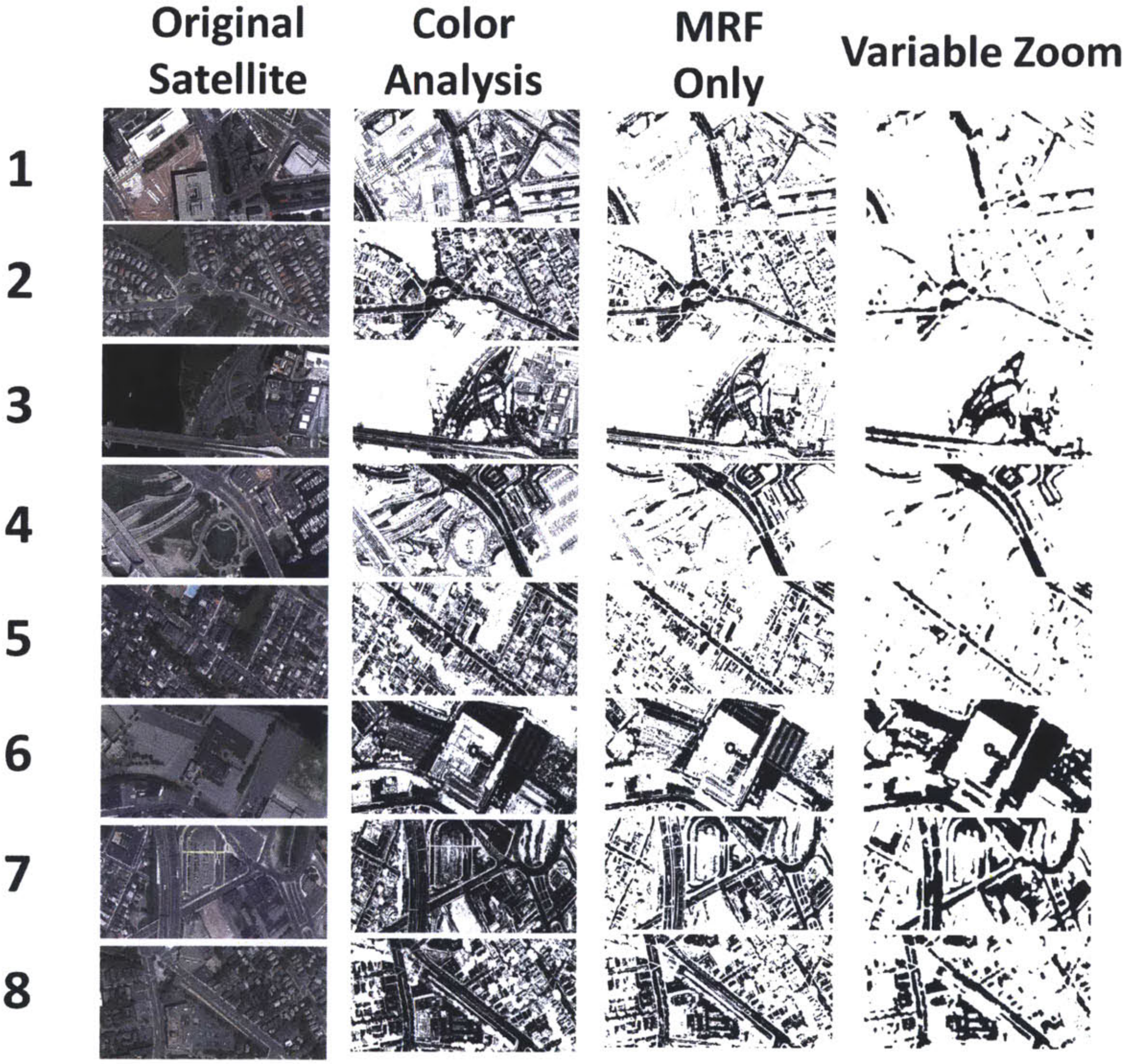


Figure F-4: The first column is the original satellite images, the second column is the result of applying the color analysis, the third column is the result of applying only the MRF to the color analysis, and the last column is the result of applying variable zoom.

propagation. This allows the roadmap to be updated on a quasi-realtime basis as information is received on a minute to minute basis.

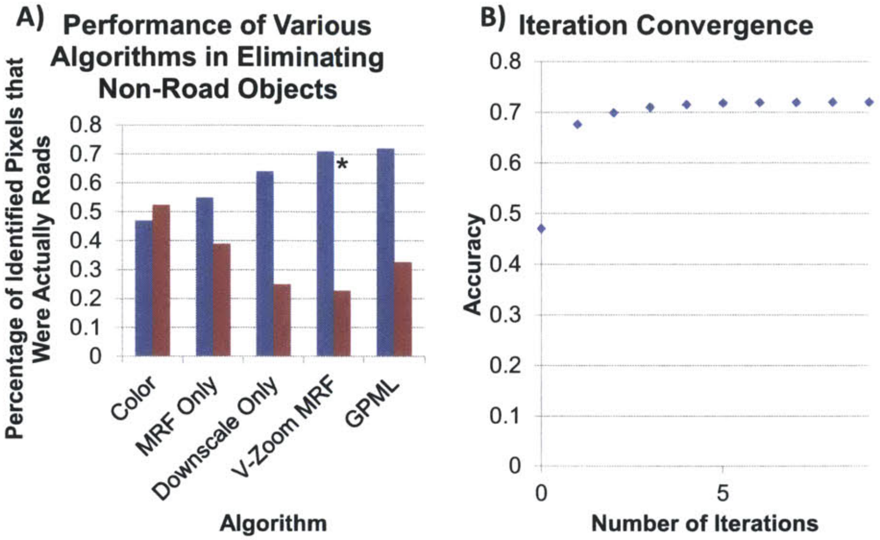


Figure F-5: A) For the output of each algorithm, we determined the percentage of pixels that were correctly identified as a road by the algorithm compared to the total number of identified pixels (shown in blue) and the percentage of actual road pixels that were correctly identified (shown in red). B) We calculated the accuracy in the same fashion for several iterations to evaluate convergence of our algorithm. In this case, variable zoom was only run for three iterations because running it for additional iterations only marginally increases the accuracy while increasing the runtime substantially.

We found that our loopy belief propagation converged to steady-state within a few iterations, indicating that our algorithm is relatively efficient. Figure F-5B demonstrates how the accuracy of the variable zoom does not improve substantially after six iterations.

Gaussian Process Machine Learning

To determine the effectiveness of the GPML, we performed cross-validation on eight images. Out of the eight images, seven were selected as the training set with their corresponding hand-labeled images as the target set to derive the best-fit function. Due to memory and run-time constraints, 1,000 pixels were randomly selected from all seven images for the fitting process. The Gaussian process was then applied to the variable zoom output from the final image, as illustrated in Figure F-6.

The results of our cross-validation are shown in Figure F-7. Figure F-5A compares all

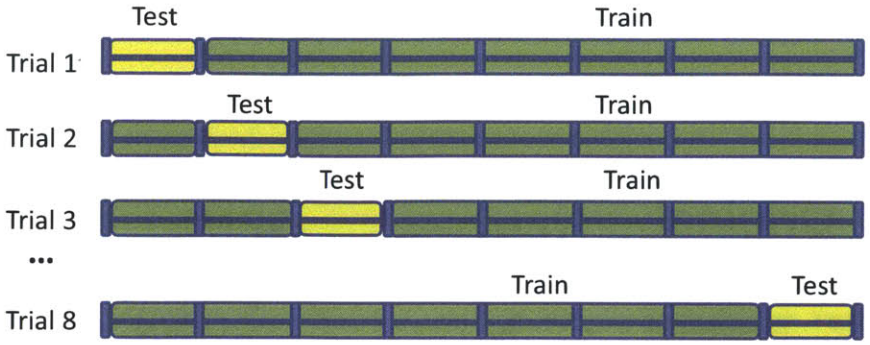


Figure F-6: Cross-validation of GPML. Seven images were used to train the Gaussian process Best-Fit Function. The resultant Gaussian process was then used to improve the variable zoom's output from the eighth image.

of the algorithms, including GPML. GPML further refines the results of the variable zoom. Furthermore, regions that were misidentified by the algorithm generally also had a high uncertainty value, which indicate that they should be avoided over identified roads with a low uncertainty.

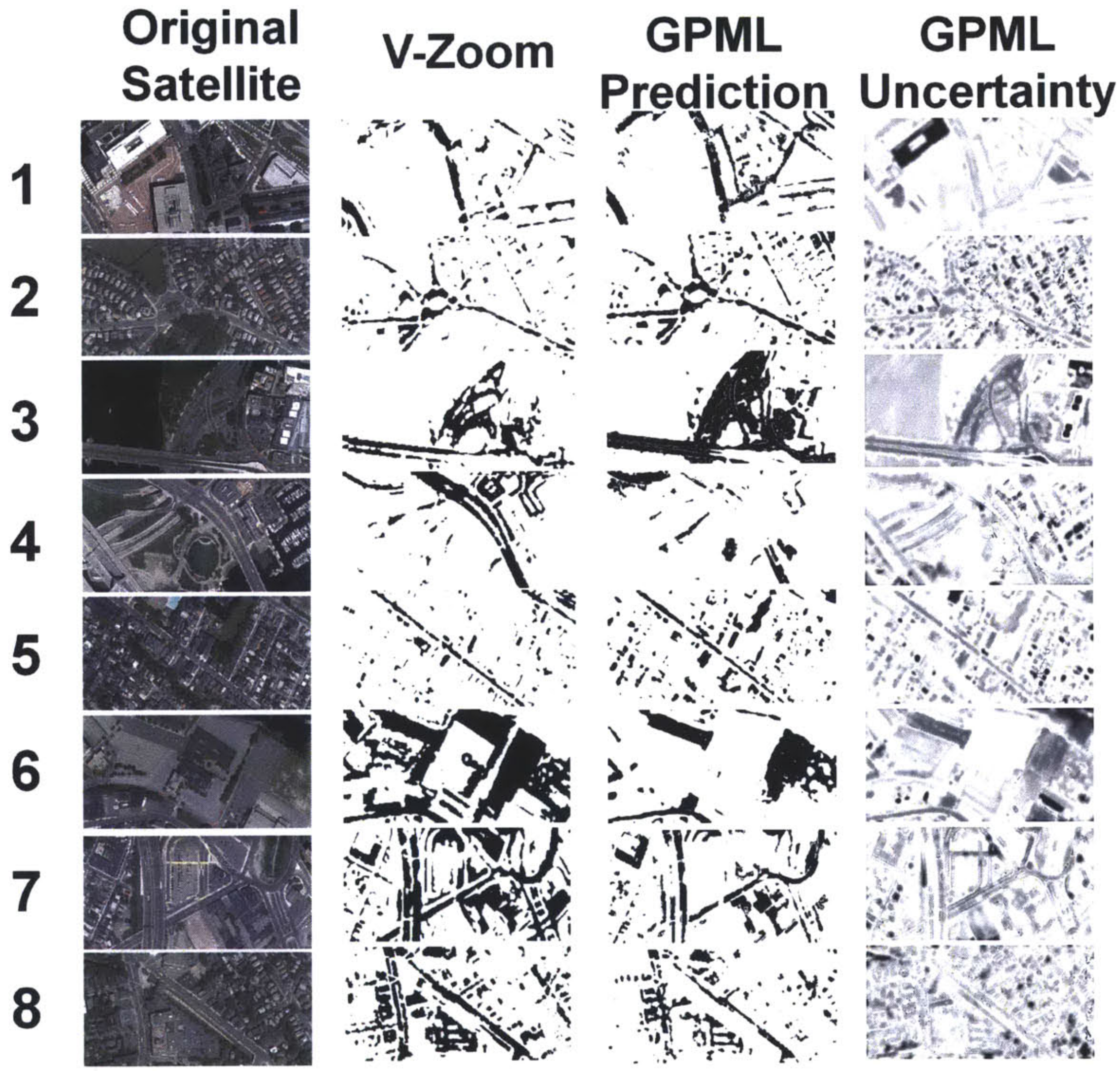


Figure F-7: GPML is shown to improve the quality of our analysis. The first column is the original satellite images, the second column is the result of applying variable zoom, the third column is the thresholded means of the Gaussian distribution for each pixel, and the last column is the standard deviations of each pixel's Gaussian distribution with darker pixels representing higher standard deviations.

G Heuristic Methods

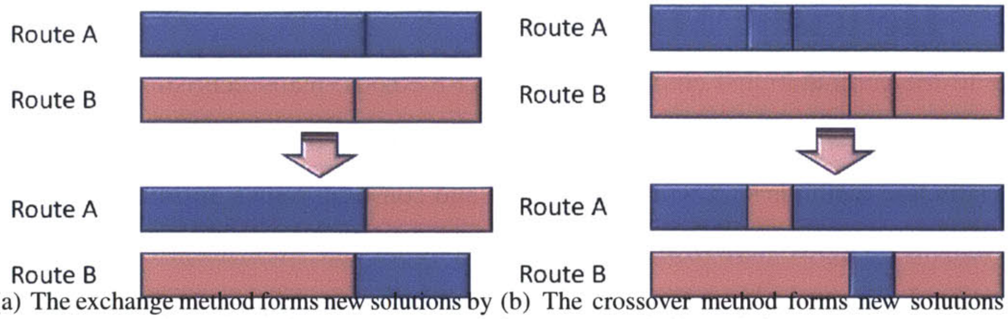
Due to the computational complexity of exact methods, many heuristic approaches have been developed. They can be categorized into constructive and improvement methods. Constructive methods start with the node set and construct routes, forming a solution. Improvement methods start with a suboptimal solution and work toward a better solution. Constructive heuristics include the insertion, savings and sweep. Improvement heuristics include crossover and exchange.

G.1 Constructive Heuristics

Insertion The insertion method is initialized by constructing n single node routes, where n is the number of locations to visit. We refer to them as "unvisited" because these routes will not be part of the final plan; however, the cost of the plan is evaluated as part of the heuristic. A partial route for the real plan is started using one of the unvisited nodes. Then, at each algorithm stage, we insert an unvisited node into the partially formed route by checking which node - position pair yields the greatest improvement to the current plan cost. A new route is formed when the current route is full. The algorithm terminates when all nodes have been inserted.

The sequential node insertion method is able to find good VRP-TW solutions. Although the resulting solution is not globally optimal, the insertion method does take into account time window constraints. Intuitively, the insertion method makes decisions based on the current stage cost g(x) (in the standard dynamic programming terminology) yet it ignores the cost-to-go. The Approximation Dynamic Programming developed later is an improved version of insertion method which approximates the cost-to-go.

Savings The Clark-Wright savings algorithm is very similar to the insertion method. It is initialized in the same manner; however, the savings method combines any routes (not just the single node routes) based on greedy cost improvement. Although the insertion algorithm performed well on the VRP-TW, the savings algorithm falls into local minima much more quickly due to the large planning steps it takes.



(a) The exchange method forms new solutions by *exchanging* portions of existing solution.

(b) The crossover method forms new solutions by *crossing* existing solution.

Figure G-1

Sweep The sweep heuristic clusters nodes using radial lines emanating outward from the start-end node. Each cluster forms a route. The number of clusters is increased until a feasible solution is found. The sweep method performs well on other multi-vehicle routing applications including the capacitated version. However, the heuristic deals with time windows very poorly and yeilds grossly suboptimal solutions.

G.2 Improvement Heuristics

Crossover The crossover heuristic attempts to improve the cost of the current feasible solution by bifurcating two routes and splicing the first half of the first route to the second half of the second route (and visa-versa) as shown in Figure A2. The algorithm keeps only those transformations which improve the solution cost.

Exchange The exchange heuristic attempts to improve the cost of the current feasible solution by simultaneously exchanging two nodes from different routes.

H Non-parametric Least Squares Regression with a Polynomial Kernel

The machine learning algorithm used to approximate the value function of our VRP-TW problem is non-parametric least squares regression with a polynomial kernel (NPLS-PK). The kernel regression method explicitly estimates future data values using weighted combinations of a current data-set. The method is fast and effective given certain conditions. Its accuracy is good but limited by the number of samples that can be handled by the regression method. It works as follows:

Suppose we have a set of state samples X and a set of target values$T$. X is an $n$by b matrix, where n is the number of state samples and b is the number of features (variables used to characterize the state). T is an n by 1 matrix where n is the number of state samples. We compute the function as follows:

$$\alpha = T' \times \left((X \times X' + 1)^d + \lambda \times I \right)^{-1}$$

where $\lambda \times I$ is a ridge penalty, and d is the kernel degree.

We can then estimate a new target value target value $\hat{t}$ given a new feature vector X_{new}. X_{new} is a single feature vector of size $1 \times b$.

$$\hat{t} = \alpha \times (X \times X'_{new} + 1)^d$$

$$X_{new} = \text{new sample}$$

$$X = \text{set of training samples}$$

Ridge Penalty and Kernel Degree

As stated above, λ is a ridge penalty. A ridge penalty first ensures that the Gram matrix is invertible. Secondly, the ridge penalty can be increased to reduce estimation variance at the expense of increased bias. d is the polynomial degree of the kernel. Thus, $X \times X'$ is the Gram matrix and $(X \times X'^{+}1)^d$is the result of applying a polynomial kernel to the Gram matrix. Essentially, this is a non-parametric way to create and use a set of polynomial basis vectors for function approximation.

One interesting aspect of this function approximation method is that if we

let $K = inv\left((X \times X'^{+}1)^{d} + \lambda \times I\right)$,
we can break up the equation

$$\alpha = T' \times \left((X \times X' + 1)^{d} + \lambda \times I\right)^{-1}$$

such that

$$\alpha = T' \times K$$

Thus, if we perform a value iteration backup (where our function approximator represents our value function) using the exact same state samples X as our original function, we can easily and quickly recreate the function approximation simply by replacing $T$with our updated one. In other words, the expensive part of the function (K) remains the same.

Bibliography

National Aeronautics and Space Administration (NASA) Ames Research Center. Mapgen : Exploring the universe. Available at `http://www.nasa.gov/centers/ames/research/exploringtheuniverse/exploringtheuniverse-mapgen.html`(2004).

Sheeraz Ahmad and Angela J Yu. Active sensing as bayes-optimal sequential decision making. *arXiv preprint arXiv:1305.6650*, 2013.

M. Athans. The role and use of the stochastic linear-quadratic-gaussian problem in control system design. *Automatic Control, IEEE Transactions on*, 16(6):529–552, 1971.

Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *The Journal of Machine Learning Research*, 3:397–422, 2003.

R. Bellman and RAND CORP. On the theory of dynamic programming. *Proceedings of the National Academy of Sciences*, 38(8):716–719, 1952.

Richard Bellmann. *Dynamic Programming*. Princeton University Press, NJ, 1957.

R. Berg, J. Clark, D. Roth, and T. Birchard. 6a. 4 the national weather service unified surface analysis.

L Mark Berliner. Bayesian analysis for oceanographers. *National Center for Atmospheric Research and Ohio State University*, 1997.

Dimitri P. Bertsekas and David A. Castanon. Rollout algorithms for stochastic scheduling problems. *Journal of Heuristics*, 5(1):89–108, 1999.

Dimitri P. Bertsekas and S. E. Shreve. *Stochastic Optimal Control: The Discrete Time Case*. Athena Scientific, Belmont, MA, 1978.

D. Bertsekas and J. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, Massachusetts, 1996.

Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

D. Bertsekas. Dynamic programming and suboptimal control: A survey from ADP to MPC. *European Journal of Control*, 11(4-5):310–334, 2005.

Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control Vol. I, 3rd Ed.* MIT Press, Cambridge, MA, 2005.

Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control Vol. II, 3rd Ed.* MIT Press, Cambridge, MA, 2007.

R. Bjarnason, P. Tadepalli, and A. Fern. Searching solitaire in real time. *ICGA Journal*, 30(3):131–142, 2007.

Julian Eugene Boggess. A genetic algorithm approach to identifying roads in satellite images. In *Proceedings of the Ninth Florida Artificial Intelligence Research Symposium, FLAIRS*, volume 96, pages 142–146. Citeseer, 1996.

Justin Boyan and Andrew W Moore. Learning evaluation functions to improve optimization by local search. *The Journal of Machine Learning Research*, 1:77–112, 2001.

Justin Boyan. *Learning Evaluation Functions for Global Optimization*. PhD thesis, Carnegie-Mellon University, 1998.

Steven J. Bradtke and Andrew G. Barto. Linear least-squares algorithms for temporal difference learning. *Mach. Learn.*, 22(1-3):33–57, 1996.

Ronen I Brafman and Moshe Tennenholtz. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *The Journal of Machine Learning Research*, 3:213–231, 2003.

Robert D Braun, Henry S Wright, Mark A Croom, Joel S Levine, and David A Spencer. Design of the ares mars airplane and mission architecture. *Journal of spacecraft and rockets*, 43(5):1026–1034, 2006.

John L Bresina, Ari K Jónsson, Paul H Morris, and Kanna Rajan. Mixed-initiative planning in mapgen: Capabilities and shortcomings. In *Proceedings of the ICAPS-05 Workshop on Mixed-initiative Planning and Scheduling, Monterey, CA*, pages 54–61. Citeseer, 2005.

Wolfram Burgard, Dieter Fox, and Sebastian Thrun. Active mobile robot localization. Technical Report IAI-TR-97-3, 25, 1997.

L.A. Bush, B. Williams, and N. Roy. Computing exploration policies via closed-form least-squares value iteration. *International Conference on Planning and Scheduling*, 2008.

Lawrence Bush, Brian Williams, and Nicholas Roy. Computing exploration policies via closed-form least-squares value iteration. *International Conference on Planning and Scheduling*, 2008.

Lawrence Bush, Brian Williams, and Nicholas Roy. Unifying plan-space value-based approximate dynamic programming policies and open loop feedback control. *AIAA Infotech@Aerospace Conference*, 2009.

Lawrence AM Bush, Andrew J Wang, and Brian C Williams. Risk-based sensing in support of adjustable autonomy. In *Aerospace Conference, 2012 IEEE*, pages 1–18. IEEE, 2012.

Lawrence AM Bush, Tony Jimenez, and Brian Williams. Adaptable mission planning for mars airplane exploration. *AIAA Infotech@Aerospace Conference*, 2013.

John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8, Issue:6:679–698, 1986.

DW Caress and DN Chayes. Mb-system: Open source software for the processing and display of swath mapping sonar data. *Internet: http://www. mbari. org/data/mbsystem*, 2008.

David W Caress, Hans Thomas, William J Kirkwood, Rob McEwen, Richard Henthorn, David A Clague, Charles K Paull, Jenny Paduan, and Katherine L Maier. High-resolution multibeam, sidescan, and subbottom surveys using the mbari auv d. allan b. *Marine Habitat Mapping Technology for Alaska*, pages 47–69, 2008.

Susan Carey. Conceptual change in childhood. In *MIT Press*, 1985.

Anthony R. Cassandra, Leslie Pack Kaelbling, and Michael L. Littman. Acting optimally in partially observable stochastic domains. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, volume 2, pages 1023–1028, Seattle, Washington, USA, 1994. AAAI Press/MIT Press.

A. R. Cassandra, L. P. Kaelbling, and J. A. Kurien. Acting under uncertainty: discrete bayesian models for mobile-robot navigation. In *In Proceedings of the International Conference on Intelligent Robots and Systems*, 1996.

Anthony R Cassandra, Leslie Pack Kaelbling, and James A Kurien. Acting under uncertainty: Discrete bayesian models for mobile-robot navigation. In *Intelligent Robots and Systems' 96, IROS 96, Proceedings of the 1996 IEEE/RSJ International Conference on*, volume 2, pages 963–972. IEEE, 1996.

Tristan Cazenave and Fabien Teytaud. Application of the nested rollout policy adaptation algorithm to the traveling salesman problem with time windows. In Youssef Hamadi and Marc Schoenauer, editors, *Learning and Intelligent Optimization*, Lecture Notes in Computer Science, pages 42–54. Springer Berlin Heidelberg, 2012.

G Clarke and JW Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, 12(4):568–581, 1964.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, pages 273–297, 1995.

N. Dalal. Histograms of orientated gradients for human detection. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 886–893, 2005.

Richard Dearden, Nir Friedman, and Stuart Russell. Bayesian q-learning. In *AAAI/IAAI*, pages 761–768, 1998.

MH DeGroot and MJ Schervish. *Probability and Statistics-International Edition*. Addison-Wesley. Publishing. Company., Reading, Massachusetts, 2001.

J. Denzler and C. M. Brown. Information theoretic sensor data selection for active object recognition and state estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(2):145 – 157, February 2002.

Martin Desrochers, Jan K Lenstra, Martin WP Savelsbergh, and François Soumis. Vehicle routing with time windows: optimization and approximation. *Vehicle routing: Methods and studies*, 16:65–84, 1988.

J. Farrell and M. Polycarpou. *Adaptive approximation based control: unifying neural, fuzzy and traditional adaptive approximation approaches*. Wiley-Blackwell, 2006.

V. Fedorov. *Theory of optimal experiments*. Academic press, New York, 1972.

V.V. Fedorov. Theory of optimal experiments. 1972.

Pedro Felzenbath and Daniel Huttenlocher. Efficient belief propagation for early vision. *International Journal of Computer Vision*, 70:41–54, 2006.

Scott Ferson, Cliff A Joslyn, Jon C Helton, William L Oberkampf, and Kari Sentz. Summary from the epistemic uncertainty workshop: consensus amid diversity. *Reliability Engineering & System Safety*, 85(1):355–369, 2004.

M.L. Fisher and K.O. Jörnsten. Vehicle routing with time windows: Two optimization algorithms. *Operations Research*, pages 488–492, 1997.

Research Institute for Advanced Computer Science. Mapgen : Automated activity plan generation. Available at `http://www.riacs.edu/research/projects/mapgen/`(2013).

TT Fujita. Mesoscale classifications: Their history and their application to forecasting. *Mesoscale meteorology and forecasting*, pages 18–35, 1986.

S. Gelly and D. Silver. Combining online and offline knowledge in UCT. In *Proceedings of the 24th international conference on Machine learning*, page 280. ACM, 2007.

Z. Ghahramani. Learning dynamic Bayesian networks. *Lecture Notes in Computer Science*, 1387:168–197, 1998.

Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated planning: theory & practice*. Access Online via Elsevier, 2004.

T.S. Glickman. Glossary of meteorology. 1972.

Google. Google maps. Available at `https://maps.google.com/`(2012/07/31).

C. Guestrin, D. Koller, R. Parr, and S. Venkataraman. Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research*, 19:399–468, 2003.

Carlos Guestrin, Andreas Krause, and Ajit Paul Singh. Near-optimal sensor placements in gaussian processes. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 265–272, New York, NY, USA, 2005. ACM.

Bradley R Hasegawa. *Continuous observation planning for autonomous exploration*. PhD thesis, Massachusetts Institute of Technology, 2004.

Trevor Hastie, Robert Tibshirani, Jerome Friedman, and James Franklin. The elements of statistical learning: data mining, inference and prediction. *The Mathematical Intelligencer*, 27(2):83–85, 2005.

Jesse Hoey, Robert St-aubin, Alan Hu, and Craig Boutilier. Spudd: Stochastic planning using decision diagrams. In *In Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 279–288. Morgan Kaufmann, 1999.

Laurie Ihrig and Subbarao Kambhampati. Plan-space vs. state-space planning in reuse and replay. *Department of Computer Science, Arizona State University, Tech. Rep*, pages 94–006, 1996.

A. Moore J. Boyan. Generalization in reinforcement learning: Safely approximating the value function. In *Tesauro, Touretzky and Leen edition of Advances in Neural Information Processing Systems*, 1995.

Tony Jimenez, Larry Bush, and Brian Bairstow. Adaptable mission planning for kinodynamic systems. *MIT 16.412 Report*, 2005.

J. Johns and S. Mahadevan. Constructing basis functions from directed graphs for value function approximation. In *Proceedings of the 24th international conference on Machine learning*, page 392. ACM, 2007.

P. Jones and S. Mitter. An iterative algorithm for autonomous tasking in sensor networks. In *Proceedings of the IEEE Conference on Decision and Control*, San Diego, CA, 2006.

f Philip Kilby, Patrick Prosser, and Paul Shaw. A comparison of traditional and constraint-based heuristic methods on vehicle routing problems with side constraints. *Constraints*, 5(4):389–414, 2000.

Armen Der Kiureghian and Ove Ditlevsen. Aleatory or epistemic? does it matter? *Structural Safety*, 31(2):105–112, 2009.

L. Kocsis and C. Szepesvari. Bandit based monte-carlo planning. *Machine Learning: ECML 2006*, pages 282–293, 2006.

Niklas Kohl. *Exact methods for time constrained routing and related scheduling problems.* PhD thesis, Institute of Mathematical Modelling, 1995.

Antoon WJ Kolen, AHG Rinnooy Kan, and HWJM Trienekens. Vehicle routing with time windows. *Operations Research*, 35(2):266–273, 1987.

Andreas Krause and Carlos Guestrin. Near-optimal nonmyopic value of information in graphical models. In *Proceedings of the 21th Annual Conference on Uncertainty in Artificial Intelligence (UAI-05)*, pages 324–33, Arlington, Virginia, 2005. AUAI Press.

A. Krause and C. Guestrin. Near-optimal observation selection using submodular functions. In *Proceedings of the National Conference on Artificial Intelligence*, volume 22, page 1650. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007.

Andreas Krause and Carlos Guestrin. Optimizing sensing: From water to the web. *Computer*, 42(8):38–45, 2009.

Jesper Larsen. Vehicle routing with time windowsfinding optimal solutions efficiently, 1999.

Thomas Léauté and Brian C Williams. Coordinating agile systems through the model-based execution of temporal plans. In *Proceedings of the National Conference on Artificial Intelligence*, volume 20, page 114. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005.

Thomas Léauté and Brian C Williams. Coordinating agile systems through the model-based execution of temporal plans. In *Proceedings of the National Conference on Artificial Intelligence*, volume 20, page 114. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005.

Daniel D. Lee and H. Sebastian Seung. Learning the parts of objects by nonnegative matrix factorization. In *Nature*, 1999.

Daniel D. Lee and H. Sebastian Seung. Algorithms for non-negative matrix factorization. In *Conference on Neural Information Processing Systems (NIPS)*, pages 556–562, 2000.

Pierre FJ Lermusiaux. Adaptive modeling, adaptive data assimilation and adaptive sampling. *Physica D: Nonlinear Phenomena*, 230(1):172–196, 2007.

Joel Levine, D. Blaney, J.E.P. Connemey, Ronald Greeley, James Head III, John Hoffman, Bruce Jakosky, Christopher McKay, Chrsitophe Sotin, and Michael Summers. Science from a mars airplane: The aerial regional-scale environmental survey (ares) of mars. *2nd AIAA Unmanned Unlimited Conference, Workshop and Exhibit*, 2003.

S. Li. *Markov Random Field Modeling in Computer Vision.* Springer-Verlag, New Jersey, 1995.

M.L. Littman. Algorithms for sequential decision making. *Brown University, Providence, RI*, 1996.

Miao Liu. Online expectation and maximization for model-free reinforcement learning in pomdps. 2013.

T Lolla, MP Ueckermann, K Yigit, Patrick J Haley, and Pierre FJ Lermusiaux. Path planning in time dependent flow fields using level set methods. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 166–173. IEEE, 2012.

D. MacKay. Information-based objective functions for active data selection. *Neural Computation*, 4(4):590–604, 1992.

D.J.C. MacKay. *Bayesian methods for adaptive models.* PhD thesis, California Institute of Technology, 1992.

David JC MacKay. *Information theory, inference and learning algorithms.* Cambridge university press, 2003.

S. Mahadevan. Proto-value functions: Developmental reinforcement learning. In *Proceedings of the 22nd international conference on Machine learning*, page 560. ACM, 2005.

Rob McEwen and Knut Streitlien. Modeling and control of a variable-length auv. *Proc 12th UUST*, 2001.

Nicolas Meuleau and Paul Bourgine. Exploration of multi-state environments: Local measures and back-propagation of uncertainty. *Machine Learning*, 35(2):117–154, 1999.

L. Mihaylova, T. Lefebvre, H. Bruyninckx, K. Gadeyne, and J. De Schutter. Active sensing for robotics - a survey. In *in Proc. 5 th Intl Conf. On Numerical Methods and Applications*, pages 316–324, 2002.

Lyudmila Mihaylova, Tine Lefebvre, Herman Bruyninckx, Klaas Gadeyne, and Joris De Schutter. A comparison of decision making criteria and optimization methods for active robotic sensing. In *Numerical Methods and Applications*, pages 316–324. Springer, 2003.

Monterey Bay Aquarium Research Institute. *Autonomous Ocean Sampling Network.* http://www.mbari.org/aosn/default.htm, 2006.

Kevin P Murphy. Bayesian map learning in dynamic environments. In *Conference on Neural Information Processing Systems (NIPS)*, pages 1015–1021, 1999.

NASA and JPL. Mars reconnaissance orbiter official web site. Available at `http://mars.jpl.nasa.gov/mro/`, 2013.

NASA. Aerial regional-scale environmental survey official web site. Available at http://marsairplane.larc.nasa.gov/, 2011.

Goddard Space Flight Center NASA. Seawifs ocean color browse. Avaiable at http://oceancolor.gsfc.nasa.gov/cgi/browse.pl?sen=sw&typ=GAC, 2011.

I. Orlanski. A rational subdivision of scales for atmospheric processes. *Bull. Amer. Meteor. Soc*, 56(5):527–530, 1975.

J. Pearl. Heuristics: intelligent search strategies for computer problem solving. 1984.

T. Perron, BA Black, S. Drummond, and DM Burr. Estimating erosional exhumation on titan from drainage network morphology. In *AGU Fall Meeting Abstracts*, volume 1, page 1796, 2011.

J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for pomdps. In *IJCAI*, 2003.

P. Poupart and C. Boutilier. Value-directed compression of POMDPs. *Advances in Neural Information Processing Systems*, pages 1579–1586, 2003.

W.B. Powell and I.O. Ryzhov. *Optimal Learning*. Wiley, 2012.

W.B. Powell. *Approximate Dynamic Programming: Solving the curses of dimensionality*. Wiley-Interscience, 2007.

M. Quigley. Semi-autonomous human-uav interfaces for fixed-wing mini-uavs. *Intelligent Robots and Systems. Proceedings. 2004 IEEE/RSJ International Conference, 2004*, 3:2457–2462, 2004.

M. Quigley. Cooperative air and ground surveillance. *Robotics & Automation Magazine, IEEE*, 13:16–25, 2006.

TK Ralphs, Leonid Kopman, William R Pulleyblank, and Leslie E Trotter. On the capacitated vehicle routing problem. *Mathematical Programming*, 94(2-3):343–359, 2003.

Carl Rasmussen. Gaussian processes in machine learning. *Advanced Lectures of Machine Learning*, 3176:63–71, 2004.

I. Robertson, D. Lucy, L. Baxter, AM Pollard, RG Aykroyd, AC Barker, AHC Carter, VR Switsur, and JS Waterhouse. A kernel based bayesian approach to climatic reconstruction. *The Holocene*, 1999.

Christopher D. Rosin. Nested rollout policy adaptation for monte carlo tree search. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence - Volume Volume One*, IJCAI'11, pages 649–654. AAAI Press, 2011.

S. Roweis and Z. Ghahramani. A unifying review of linear Gaussian models. *Neural computation*, 11(2):305–345, 1999.

Nicholas Roy, Han-Lim Choi, Daniel Gombos, James Hansen, Jonathan How, and Sooho Park. Adaptive observation strategies for forecast error minimization. *Lecture Notes in Computer Science*, 2007.

Donald Rumsfeld. Feb. 12, 2002, department of defense news briefing.

Donald Rumsfeld. *Known and unknown: a memoir.* Sentinel, 2011.

P. Schweitzer and A. Seidmann. Generalized polynomial approximations in Markovian decision processes. *Journal of mathematical analysis and applications*, 110(6):568–582, 1985.

Hart Seely. *Pieces of Intelligence: The Existential Poetry of Donald H. Rumsfeld.* Simon and Schuster, 2003.

L. G. Shapiro and G. C Stockman. *Computer Vision.* Prentice Hall, New Jersey, 2001.

R. Sim and N. Roy. Global a-optimal robot exploration in slam. In *Proceedings of the International Conference on Robotics and Automation*, Barcelona, Spain, 2005.

S. E. Smrekar. Mars reconnaissance orbiter mission and science investigations special collection. *Journal of Geophysical Research*, 112(E5), 2007.

Soloman. Type rc2 benchmark problems. Avaiable at http://www.idsia.ch/ luca/macs-vrptw/problems/r201.txt, 1987.

Edward Sondik. *The Optimal Control of Partially Observable Markov Decision Processes.* PhD thesis, Stanford University, Stanford, California, 1971.

Robert St-Aubin, Jesse Hoey, and Craig Boutilier. APRICODD: Approximate policy construction using decision diagrams. In *Conference on Neural Information Processing Systems (NIPS)*, pages 1089–1095, 2000.

Christopher Stauffer. Perceptual data mining: Bootstrapping visual intelligence from tracking behavior. In *Thesis*, 2002.

Gaurav S. Sukhatme, Amit Dhariwal, Bin Zhang, Carl Oberg, Beth Stauffer, and David A. Caron. The design and development of a wireless robotic networked aquatic microbial observing system. *Environmental Engineering Science*, 24(2):205–215, 2006.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Athena Scientific, Belmont, MA, 1998.

Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

Gerald Tesauro and Gregory R Galperin. On-line policy improvement using monte-carlo search. *Advances in Neural Information Processing Systems*, pages 1068–1074, 1997.

Gerald Tesauro. Temporal difference learning of backgammon strategy. In *Proceedings of the Ninth International Workshop on Machine Learning*, pages 451–457. Morgan Kaufmann Publishers Inc., 1992.

Gerald Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

S.K. Thompson and L.M. Collins. Adaptive sampling in research on risk-related behaviors. *Drug and Alcohol Dependence*, 68:57–67, 2002.

S.K. Thompson, G.A.F. Seber, et al. *Adaptive sampling*. Wiley New York, 1996.

Sebastian Thrun. Learning occupancy grid maps with forward sensor models. *Autonomous robots*, 15(2):111–127, 2003.

Paolo Traverso, Malik Ghallab, and Dana Nau. Automated planning: theory and practice. *Morgan Kauffman*, 2004.

J.N. Tsitsiklis and B. Van Roy. Feature-based methods for large scale dynamic programming. *Machine Learning*, 22(1):59–94, 1996.

D. Wang, P.F.J. Lermusiaux, P.J. Haley, D. Eickstedt, W.G. Leslie, and H. Schmidt. Acoustically focused adaptive sampling and on-board routing for marine rapid environmental assessment. *Journal of Marine Systems*, 78:S393–S407, 2009.

Maxwell B Wang, Lawrence A Bush, Albert Chu, and Brian C Williams. Active detection of drivable surfaces in support of robotic disaster relief missions. In *Aerospace Conference, 2013 IEEE*. IEEE, 2013.

D. Wang. *Autonomous Underwater Vehicle (AUV) path planning and adaptive on-board routing for adaptive rapid environmental assessment.* PhD thesis, Massachusetts Institute of Technology, 2007.

Brian C Williams and Robert J Ragno. Conflict-directed a* and its role in model-based embedded systems. *Discrete Applied Mathematics*, 155(12):1562–1595, 2007.

K. Woolsey. Rollouts. *Inside Backgammon*, 1991.

Peter Wurman, Raffaello D'Andrea, and Mick Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine*, 29:9–19, 2008.

Fei Xu and Vashti Garcia. Intuitive statistics by 8-month-old infants. In *PNAS*, 2008.

N. K. Yilmaz, C. Evangelinos, PFJ Lermusiaux, and NM Patrikalakis. Path planning of autonomous underwater vehicles for adaptive sampling using mixed integer linear programming. *IEEE Journal of Oceanic Engineering*, 33(4):522–537, 2008.

Contents

List of Figures

List of Tables

List of Algorithms