CORE
Metadata, citation and similar papers at core.ac.uk
Provided by ScholarBank@NUS

PLANNING UNDER UNCERTAINTY: FROM INFORMATIVE PATH PLANNING TO PARTIALLY OBSERVABLE SEMI-MDPS

Lim Zhan Wei

B.Comp. (Hons.), NUS

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

SCHOOL OF COMPUTING

NATIONAL UNIVERSITY OF SINGAPORE

2015

DECLARATION

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

Lim Zhan Wei

22 MAY 2015

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to Professor Lee Wee Sun and Professor David Hsu for their guidance and support throughout my PhD journey. They are brillant computer scientists and inspiring intellectual mentors. This thesis would not be possible without them. I am also grateful to Professor Leong Tze Yun and Professor Bryan Low for their helpful suggestion towards improving this thesis.

I am lucky to have many smart and fun-loving colleagues in my research group. Thanks for the all the beautiful memories: Wu Dan, Amit, Liling, Kok Sung, Sylvie, Ye Nan, Haoyu, Benjamin, Kegui, Dao, Vien, Wang Yi, Ankit, Shaojun, Ziquan, Zongzhang, Chen Min, Jue Kun, Neha, and Andras. I would also like to thank my friends in school of computing whom together with my colleagues made lunch time the highlight of my work day: Zhiqiang, Jianxin, Nannan, Chengwen, and Jovian.

To my parents and my brothers, thanks for taking good care of me and supporting my PhD career even though I have never been able to explain my research to you. To my fiancée, Eunice, thank you for giving me the strength and motivation I need to complete this journey.

Contents

ABSTRACT

Planning under uncertainty is crucial to the success of many autonomous systems. An agent interacting in the real-world often has to deal with uncertainty due to unknown environment, noisy sensor measurements, and imprecise actuation. It also has to continuously *adapt* to circumstances as the world unfolds. Partially Observable Markov Decision Process (POMDP) is an elegant and general framework for modeling planning under such uncertainties. Unfortunately, solving POMDPs grows computationally intractable as the size of state, action, and observations space increase. This thesis examines useful subclasses of POMDPs and algorithms to solve them efficiently.

We look at informative path planning (IPP) problems where an agent seeks a minimum cost path to sense the world and gather information. IPP generalizes the well-known optimal decision tree problem from selecting subset of tests to selecting paths. We present *Recursive Adaptive Identification* (RAId), a new polynomial time algorithm and obtain a polylogarithmic approximation bound for IPP problems without observation noise.

We also study adaptive stochastic optimization problems, a generalization of IPP from gathering information to general goals. In adaptive stochastic optimization problems, an agent minimizes the cost of a sequence of actions to achieve its goal under uncertainty, where its progress towards the goal can be measured by an appropriate function. We propose the marginal likelihood rate bound condition for pointwise submodular functions as a condition that allows efficient approximation for adaptive stochastic optimization problems. We develop *Recursive Adaptive Coverage* (RAC), a near-optimal polynomial time algorithm that exploits properties of the marginal likelihood rate bound to solve problems that optimize these functions. We further propose a more general condition, the marginal likelihood bound that contains all finite pointwise submodular monotone functions. Using a modified version of RAC, we obtain an approximation bound that depends on a problem specific constant for the marginal likelihood bound condition.

Finally, scaling up POMDPs is hard when the task takes many actions to complete. We examine the special case of POMDPs that can be well approximated using sequences of macro-actions that encapsulate several primitive actions. We give sufficient conditions for macro actions model to retain good theoretical properties of POMDP. We introduce *Macro-Monte Carlo Value Iteration* (Macro-MCVI), an algorithm that enables the use of macro actions in POMDP. Macro-MCVI only needs a generative model for macro actions, making it easy to specify macro actions for effective approximation.

List of Tables

List of Figures

Chapter 1

Introduction

Planning is at the core of many intelligent systems. Planning comprises of formulation, evaluation and selection of sequence of actions to achieve a desired goal. Classical planning in AI assumes that an agent has complete knowledge of the world state and actions have precise and deterministic effects. However, these assumptions do not hold in most real world environments. Real world sensor measurements are often limited and inaccurate. Actions are seen to have imprecise effects due to lack of perfect physics model for every matter in the environment. In a dynamic and unstructured environment, an agent does not have complete information about its world. It needs to reason over its prior knowledge about the world and its sensor measurements to decide its next action. Furthermore, the agent has to deal with contingencies and needs to improvise as the world unfolds.

Consider a household robot tasked to clean up after a dinner. It has to clear the leftovers, pick up dirty utensils on the dining table, place them in the dishwasher and wipe the table. This seemingly easy but tedious task for human can be very difficult for a robot. First, the robot does not have complete information about the clutter; the dining area may not fit into the field of view of its camera; some utensils may be occluded by rubbish. Second, the robot has imprecise control when it is manipulating the plates with varying weight of leftover on it. Moreover, the robot may encounter unexpected obstacles such as human occupants or other household objects while cleaning up.

State and action uncertainty is inherent in the real world. Even a human cannot have perfect knowledge of his environment. Instead, a human works around the lack of

complete knowledge or seeks to acquire information that are critical. How can a robot plan its actions to tackle such a complex task in face of uncertainties?

The household robot example and many problems in AI can be modeled as *Partially observable Markov decision processes* (POMDPs). The POMDP is a mathematically elegant and general framework for modeling planning under uncertainty. Unfortunately, solving POMDP is computationally intractable. However, there exist subclasses of POMDP where it is possible to obtain good approximate solutions. This thesis aims to identify interesting subclasses of POMDPs with properties that allow efficient approximate solution.

We begin by examining path planning problems where an agent needs to gather information efficiently. For example, the household cleaning robot needs to plan a series of maneuvers around the table to locate the dirty utensils on the dining table. Such information gathering tasks can be formulated as an *informative path planning* (IPP) problems. We are interested in computing an *adaptive* solution to IPP that selects actions using both prior information and new knowledge that the agent acquires along a path.

The second part of this thesis looks at adaptive stochastic optimization, a generalization of adaptive informative path planning from information gathering to achieving general goals under uncertainty, where the goals can be characterized by appropriate functions. For instance, the amount of leftovers and dirty utensils cleared by the robot can be modeled as a function that has the maximum value when the dining table is cleared. However, the robot does not know the exact state of the dining table at first and the function associated with the state. In other words, it is unsure of the locations of dishes and how to clear the table. The robot has to plan a series of movement to reveal the leftovers and clear them.

IPP and adaptive stochastic optimization problems have limited model expressiveness to trade-off for increased computational efficiency over POMDPs. One of their model limitations is the lack of ability to model actions with uncertain effects, such as plate slipping out of the robot hand while trying to grasp it. Another distincton is that the same action cannot be repeated in IPP and adaptive stochastic optimization. This is suitable for the household robot figuring out the positions of leftovers on the dining

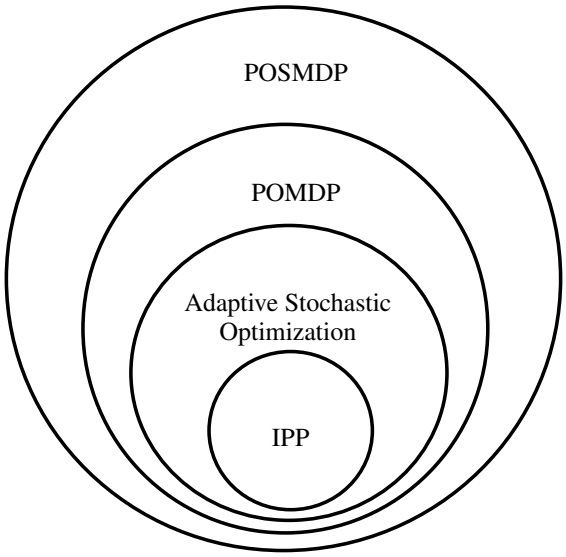

table as there is no benefit in looking from the same angle twice. However, this model constraint may not be suitable for other planning problems.

Without sacrificing model expressiveness, POMDPs can be made easier if we consider temporally abstracted macro actions. An example of a macro action for a household robot is "move from table to kitchen". Such macro actions can replace long sequence of actions to achieve some intermediate goal. The final part of this thesis considers POMDP problems that can be well approximated by a sequence of macro actions. Adding macro actions to POMDP gives us a *Partially observable semi-Markov decision process* (POSMDP).

We show the relationship between the problems we study in this thesis in the following Venn diagram (fig. 4.1).

1.1 Informative Path Planning

One of the hallmarks of an intelligent agent is its ability to gather information necessary to complete its task. *Informative path planning* (IPP) seeks a path for the robot to sense the world and gain information. IPP is useful in a range of information gathering applications:

- An unmanned aerial vehicle (UAV) searches a disaster region to pinpoint the location of survivors (A. Singh, Krause, and Kaiser, 2009).
- An autonomous underwater vehicle inspects the submerged part of a ship hull

(Hollinger et al., 2013).

IPP is also an important component in robotic applications where a robot needs to acquire missing information in order to complete its task. For example, the household cleaning robot has to move its sensors to locate the dirty dishes before it can place them in the dishwasher. Also, a mobile manipulator needs to move around and sense an object with laser range finders (Platt Jr et al., 2011) or tactile sensors (Javdani, Klingensmith, et al., 2013) in order to estimate the object pose for grasping.

In these tasks, the robot has a set of hypotheses on the underlying state of the world—the location of survivors, the pose of an object, *etc.*—and must move to different locations in order to sense and eventually identify the true hypothesis. Each sensing operation provides new information, which enables the robot to act more effectively in the future. To acquire this information, the robot, however, must move around and incur movement cost, in addition to sensing cost. A key issue in designing efficient IPP algorithms is the *trade-off* between information gain and robot movement cost.

There are two general classes of algorithms for IPP, nonadaptive and adaptive. In *nonadaptive* planning, we compute a sequence of sensing operations in advance. A robot executes these operation in order, regardless of the outcomes of earlier operations. In *adaptive* planning, we choose, in each step, new sensing operations conditioned on the outcomes of earlier sensing operations. For example, the household robot may change its direction of search for dirty dishes after some initial assessment of the scene. In this thesis, we examine the adaptive planning case.

IPP contains, as a special case, the well-studied optimal decision tree (ODT) problem (Chakaravarthy et al., 2007), in which we want to build a decision tree that minimizes the expected number of tests required to identify a hypothesis. ODT is basically IPP with a single location containing all sensing operations. Thus the costs of all sensing operations are the same. Unfortunately, ODT, even with noiseless sensing, is not only NP-hard, but also NP-hard to approximate within a factor of $\Omega(\log n)$, where n is the total number of hypotheses (Chakaravarthy et al., 2007).

Nevertheless, there are interesting structures in information gathering that can be exploited. For example, making more observations at various locations in the environment has "diminishing returns" in the information gained due to "overlaps" in information.

These structures allow us to construct efficient algorithm with performance guarantees.

1.2 Adaptive Stochastic Optimization

Many planning and learning problems in AI require an agent to adaptively select actions based on both prior information and newly acquired knowledge as the agent is executing its plan. While POMDP is sufficiently general to model these problems, some of these problems can be modeled using a simpler formulation called the adaptive stochastic optimization.

Suppose the household cleaning robot knows the exact state of the leftovers on the dining table, then the problem becomes a deterministic optimization problem where the actions can be picking up leftovers at a position or sweeping through a region on the table and the objective function is the amount of leftovers cleared. The robot needs to pick the minimum cost subset of actions to get the table cleared. However, in reality the robot does not know the exact state of the leftovers at the start due to limted sensors range. It needs to find out the positions of leftovers while clearing them at the same time. Such problems with unknown environment state can be better modeled as an *adaptive stochastic optimization*.

In adaptive stochastic optimization, we model the hidden aspects of the agent's environment as a random variable whose values are *scenarios* that the agent could encounter, such as the positions of the leftovers on the dining table. The objective function depends on the actions taken and the true scenario of the world. For instance the usefulness of a sweeping action by the household robot depeonds on whether there are leftovers in the region it is sweeping. The agent can gain information about the scenario as it takes actions and make observations. The aim of adaptive stochastic optimization is to adaptively choose a minimum cost sequence of actions to achieve its goal for the scenario the agent encounters.

Adaptive stochastic optimization can be applied to a range of planning and learning tasks. When we use adaptive stochastic optimization to do information gathering, the objective function may be a general measure of information such as Shannon's entropy. Other planning applications include adaptive viral marketing campaign where we pick influential members in a social network to offer promotional deals so that they

spread the marketing message to other members in the social network. In this case, the objective function is the total number of individuals the marketing message reached.

Prior works on adaptive stochastic optimization are mostly restricted to set optimization problems where an agent's action is to select a subset of items and the cost of each items is fixed. We refer to these problems, which include sensor placement, viral marketing and active learning problems as adaptive stochastic optimization on *subsets*. Our work considers adaptive stochastic optimization on subsets as well as a richer formulation where an agent's actions form a path in a metric space, and the cost of visiting a location to gather information depends on the current location of the agent. We call this latter problem adaptive stochastic optimization on *paths*.

Adaptive stochastic optimization in general can be computationally intractable (Golovin and Krause, 2011). In Golovin and Krause (2011), a condition called *adaptive submodularity* was shown to be sufficient for an efficient greedy algorithm to provide a good approximation for adaptive stochastic optimization problem on subsets However, it is unclear if adaptive submodularity is sufficient for an efficient approximation algorithm to exist for the adaptive stochastic optimization problem on paths.

1.3 Partially observable Markov decision processes

IPP enables us to model the information gathering process in an uncertain environment. In some robotics task such as estimating the object pose for grasping (Javdani, Klingensmith, et al., 2013), we can acquire the necessary information and then plan according to the acquired information. In spite of that, many complex robotic tasks cannot be modelled as such a two-stage process. Uncertainties can creep into the environment continuously if the environment is dynamic. The robot's actions may also change the environment with uncertain effects. For instance, the robot in the dining table cleaning example may accidentally knock into other object and thus changing the object's position. Adaptive stochastic optimization is able to model a range of planning objective through its stochastic objective function but it is unable to model actions with uncertain effects.

Partially observable Markov decision process (POMDP) provides a principled and general framework for planning with imperfect state information. In POMDP planning,

we represent an agent's possible states probabilistically as a *belief* and systematically reason over the space of all beliefs to derive a policy that is robust under uncertainty. POMDPs have been successfully applied to a wide range of tasks, for example pedestrians avoidance in self-driving vehicle where human intentions are uncertain (Bandyopadhyay et al., 2013), collision avoidance systems in unmanned aircraft (Bai, Hsu, Kochenderfer, et al., 2011), assisting person with dementia during handwashing (Hoey et al., 2007), and spoken dialog systems (Williams and Young, 2007). Both IPP and adaptive stochastic optimization are special case of POMDP.

Despite POMDP's expressive power, its real-world applications are limited due to its intractability. Computing an optimal policy is PSPACE-complete (Papadimitriou, C.H and Tsitsiklis, J.N., 1987) while classical planning problems are mostly NP-hard. Optimal solutions are only possible for tiny problems. The focus in POMDP research has been on developing scalable approximate algorithms that finds good solutions to real-world problems.

POMDP planning faces two major computational challenges. The first challenge is the "curse of dimensionality". POMDP reasons over a belief space with dimension equal number of states. A complex planning task involves a large number of states resulting in a extremely high dimensional belief space. The second obstacle is the "curse of history". In applications such as robot motion planning, an agent often takes many actions before reaching the goal, resulting in a long planning horizon. The complexity of the planning task grows quickly with the horizon due the exponential number of sequences of actions and observations to consider. Together, they compound the difficulty of POMDP planning.

On the other hand, planning for the dinner clean up task is easier for humans because we are able to *abstract* a series of coordinated eyes and hand movement to pick up a plate into a single concept of "pick up that plate". Humans have pre-learned the action sequence to pick up a plate from young, we just need to initiate this "lower level" action plan, monitor and troubleshoot if something goes wrong. Thus, humans make "higher level" plan in terms of stacking up plates, clearing leftover, carry plates to dishwasher, etc. Clearly, planning on an abstract level gives us a problem with much shorter horizon. Individual sub-tasks are decomposed in various components with limited interference

between them. For example, the number of trips a human makes to the rubbish bin to clear leftover should not affect the task of putting the dishes in the washer after we have cleared all rubbish; A human should never have to consider all combinations of action sequences to clear rubbish and dish washing action sequences.

A straightforward approach to deal with "curse of history" is to exploit temporal abstraction using macro actions to transform the problem to one with shorter horizon. Macro actions extend the usual primitive actions to operate over multiple time steps. We can abstract complex action plans into one macro action, similar to humans' pre-learned action plan to pick up a plate. Macro actions isolate and hide the primitive actions it use internally from the planner. Adding macro actions to a POMDP gives a partially observable semi-Markov decision process (POSMDP).

However, using macro actions may lead to a sub-optimal solution as there may exist an optimal solution that cannot be composed from macro actions. This is a reasonable price to pay for improving computational tractability. We view macro actions as composable sub-policies that work well together most of the time, but sub-optimal in some uncommon and inconsequential situations.

Temporal abstraction is not a novel concept in computer science. However, using macro actions is not as straightforward for planning under uncertainty, compared to using them in classical planning. The second part of this thesis aims to bridge the gap between temporal abstraction and planning under uncertainty. Our focus is on developing the theory and algorithm to attack the "curse of dimensionality" using macro actions in POMDPs.

1.4 Contributions

Chapter 3 describes the *Recursive Adaptive Identification* (RAId) algorithm. RAId is a new algorithm to solve a noiseless version of IPP problems in polynomial time with a polylogarithmic bound on the solution quality when the robot travels in a metric space. Our experiments suggest that RAId is efficient in practice and provides good approximate solutions for several distinct robot planning tasks.

Chapter 4 proposes two novel conditions for objective functions of adaptive stochastic optimization problems to be efficiently approximable, the marginal likelihood rate

bound and the marginal likelihood bound. We further describe the *Recursive Adaptive Coverage* (RAC) to compute approximate solutions to problems optimizing these functions. RAC together with the marginal likelihood rate bound and the marginal likelihood bound conditions extends existing results for adaptive stochastic optimization problems from subsets to paths and expands the class of objective functions for adaptive stochastic optimization problems that have polynomial time approximation.

We apply RAC to the IPP problems with noisy observations by optimizing a suitable objective function. This approach is near-optimal for IPP with noisy observations when the hypothesis can always be identified. Empirically, we show promising results using this approach on IPP problems with noisy observations.

Chapter 5 examines theoretical properties of POSMDP where we add macro actions to POMDP. POSMDP does not retain theoretical properties of POMDP without additional assumptions. In particular, its value function is not necessarily piecewise linear and convex. This property is desirable because it means that the value function of the process can be approximated arbitrarily closely with α-vectors, which is a convenient representation for many analyses and algorithms. We prove that it is sufficient for value function of the process to retain piecewise linearity and convexity if primitive action policies within the macro actions do not depend on the belief, although it may depend on any observed subset of states in the belief. This assumption is still general enough to represent many useful macro actions.

We extend an existing POMDP planning algorithm to use predefined macro actions. MCVI is an algorithm to solve POMDPs with very large state or continuous state space, using Monte-Carlo simulations to evaluate and construct policies. The new algorithm, Macro-MCVI, attained significant performance improvement in simulation of long horizon robotic tasks. Theoretical bound for MCVI was shown to hold in Macro-MCVI as long as the POSMDP retains piecewise linearity and convexity of its value function. Furthermore, Macro-MCVI only needs a generative model for each macro action, making it easier to define and use macro actions.

1.5 Outline

This thesis is organized in order of increasing complexity of the problems. Chapter 2 first reviews the background of the planning problems discussed in this thesis, namely IPP, adaptive stochastic optimization, and POMDP. Chapter 3 studies the first and simplest problem in this thesis, the IPP problem and proposes RAId to solve IPP without observation noise. Chapter 4 proposes two novel conditions of objective functions for adaptive stochastic optimization problems and give the RAC algorithm to solve problems satisfying these conditions. Finally, Chapter 5 looks at the hardest problem, POMDP. To scale up POMDPs to long horizon task, we extend POMDP by adding macro actions which gives us a POSMDP. We then propose the Macro-MCVI algorithm that approximates its solution.

Chapter 2

Background

This chapter reviews the IPP, adaptive stochastic optimization, and POMDP problems. We formally describe each problem, give the scope of our work, and cover the background necessary for this thesis. IPP, adaptive stochastic optimization, and POMDP are closely related problems. IPP is a special case of adaptive stochastic optimization and both IPP and adaptive stochastic optimization are special cases of POMDP. We proceed from the most specific problem to the most general one.

2.1 Informative Path Planning

This section provides a formal specification of the IPP problem. We discuss our choice of formulation and its implications. Finally, we highlight a few potential applications of IPP.

The IPP problem seeks a path for robot to sense the world and gain information. Formally an IPP problem is specified as a tuple $\mathcal{I} = (X, d, H, \rho, O, \mathcal{Z}, r)$. First, X is a finite set of sensing locations, with associated distance metric $d(x, x')$ for any two locations $x, x' \in X$. Next, H is a finite set of hypotheses, and $\rho(h)$ specifies the prior probability of hypothesis $h \in H$ occurring. We also have a finite set of observations O and a set of observation functions $\mathcal{Z} = \{Z_x \mid x \in X\}$, with one observation function Z_x for each location location x. For generality, we define the observation functions probabilistically: $Z_x(h, o) = p(o|x, h)$. For noiseless observations, $Z_x(h, o)$ is either 1 or 0. Finally, r is the robot's start location. To simplify the presentation, we assume $r \notin X$. Either r provides no useful sensing information or the robot has already visited

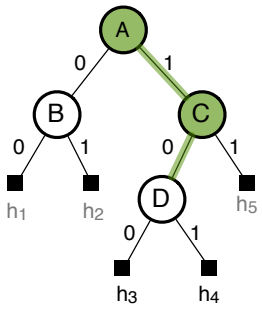

Figure 2.1: A policy tree with sensing locations $\{A, B, C, D\}$, observations $\{0, 1\}$, hypotheses $\{h_1, h_2, \ldots, h_5\}$. With noiseless observations, every path in a policy tree from the root to a leaf uniquely identifies a hypothesis. Suppose that a robot follows the shaded path σ. Then a hypothesis h is consistent with all observations received along σ if and only if h belongs to the subtree rooted at the node D, *i.e.*, $\{h_3, h_4\}$.

r and acquired the information.

We say that a hypothesis h is *consistent* with an observation o at a sensing location x if $Z_x(h, o) = 1$. Otherwise, it is *inconsistent*. If a hypothesis is inconsistent with a received observation, it clearly is not the true hypothesis and can be eliminated from further consideration.

In adaptive planning, the solution is a *policy* π, can be represented as a tree. Each node of the policy tree is labeled with a sensing location $x \in X$, and each edge is labeled with an observation $o \in O$ (see Figure 2.1). To execute such a policy, the robot starts by moving to the location at the root of the policy tree and receives an observation o. It then follows the edge labeled with o and moves to the next location at the child node. The process continues until the robot identifies the true hypothesis. Thus every path in the policy tree of π uniquely identifies a hypothesis $h \in H$. Let $C(\pi, h)$ denote the total cost of traversing this path. Our goal is to find a policy that identifies the true hypothesis by taking observations at the chosen locations and minimizes the expected cost of traveling.

We now state the problem formally:

Problem 1. *Given an IPP problem $\mathcal{I} = (X, d, H, \rho, O, \mathcal{Z}, r)$, compute an adaptive policy π that minimizes the expected cost*

$$C(\pi) = \mathrm{E}_H C(\pi, h) = \sum_{h \in H} C(\pi, h)\rho(h). \tag{2.1}$$

We assume without loss of generality that in the worst case, the true hypothesis can be

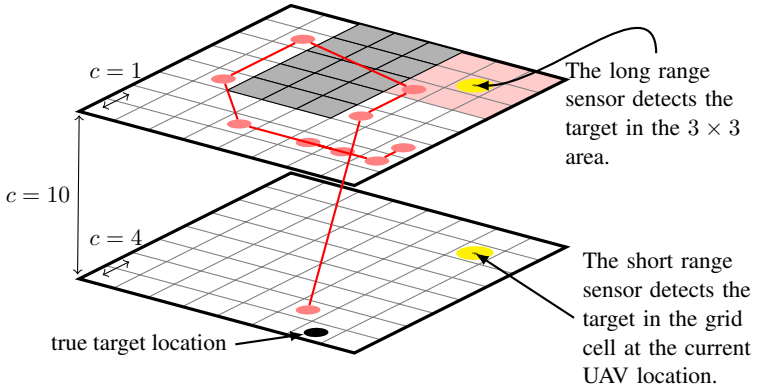

Figure 2.2: Search for a stationary target in an 8×8 grid. At the high altitude, the long-range sensor provides no information in the area shaded in gray, due to occlusion. The red curve indicates a sample path generated by RAId.

identified by visiting all locations in X.

Example. We now illustrate the definition above with a concrete example. A UAV searches for a stationary target in an area modeled as an 8×8 grid and must identify the grid cell that contains the target (Figure 2.2). Initially the target may lie in any of the cells with equal probabilities.

The UAV can operate at two different altitudes. At the high altitude, it uses a long-range sensor that determines whether the 3×3 grid around its current location contains the target. At the low altitude, the UAV uses a more accurate short-range sensor that determines whether the current grid cell contains the target. Some grid cells are not visible from the high altitude because of occlusion, and the UAV must descend to the low altitude in order to search these cells.

The UAV starts at the low altitude. We use the Manhattan distance between two grid cells as the basis of calculating the movement cost. The cost of flying between two adjacent cells at the high altitude is 1. The corresponding cost at the low altitude is 4. The cost to move between high and low altitudes is 10.

Compared with the high altitude, the low altitude offers more accurate information over a smaller area and incurs a higher cost. The challenge is then to manage this trade-off.

In this example, the hypotheses are the grid cells that may contain the target: $H = \big\{(i, j) \mid i, j \in \{1, 2, \ldots, 8\}\big\}$. The prior probability ρ over the hypotheses is the uni-

form distribution. The sensing locations are the UAV locations at the two altitudes: $X = \{(i, j, k) \mid i, j \in \{1, 2, \ldots, 8\}, k \in \{1, 2\}\}$. The metric d is the shortest distance to move between two grid cells. There are two observation: $O = \{1, 0\}$, indicating whether the target is detected or not. The observation function $Z_x(h, o)$ specifies whether the hypothesis h is consistent with the observation o received at location x. For example, if the UAV receives observation $o = 0$ at a low-altitude location $x = (2, 2, 1)$, a single hypothesis $h = (2, 2)$ is inconsistent and can be eliminated. In comparison, if the UAV receives $o = 0$ at the corresponding high-altitude location $x = (2, 2, 2)$, nine hypotheses corresponding to grid cells adjacent to $(2, 2)$ are inconsistent and can all be eliminated. □

2.1.1 Alternative Formulation

An alternative formulation of IPP problems allows us to impose a budget on movement and sensing cost while maximizing the information gathered. Theoretically, we can apply algorithms designed for one formulation to IPP problem in another formulation easily. Suppose an algorithm $A(\theta)$ computes a policy that minimizes expected cost to achieve some target quantity of information θ. To apply algorithm A to maximizing information gathered given a budget β, we can run a binary search of the value of θ such that $\arg\max_\theta \mathrm{E}_H[C(A(\theta), h)] \leq \beta$. We can do binary search in the same way to algorithms that are designed for the alternative formulation to obtain an algorithm for our formulation.

However, empirically there are applications that are better modeled by one formulation than the other. Maximizing information gathering given a budget is useful when a robot has to do environmental sensing in a large area given limited fuel or battery power. An example of such applications is lake and river monitoring (A. Singh, Krause, Guestrin, et al., 2009; Low, Dolan, and Khosla, 2009). In these applications, even if we discretize the state values, it is not practical to model every possible environment state as a hypothesis because the number of environment states can be exponential in the area of operation when we represent each discrete unit area as a random variable. We need further manipulation to practically model these applications using our formulation, such as sampling environment states as hypothesis.

On the other hand, minimizing expected cost to gather a certain amount of information is naturally suitable in robotic tasks where a robot has to reduce its uncertainty about its world before it act. For example, when a robot is searching for an object (Hollinger et al., 2013) or identifying the pose of an object of interest (Javdani, Klingensmith, et al., 2013). Battery life is usually not a concern for these applications and the number of world states is small enough for each state to be modeled as a hypothesis.

2.1.2 Adaptivity

Consider the simple problem of searching for an element in an sorted array of n numbers. Linear search is nonadaptive. It chooses each element for comparison in order. The outcome of a comparison does not affect the next element chosen. In contrast, the observation received at each step of an adaptive algorithm affects the choice in future steps, as in choosing a branch in binary search. Each comparison splits the array into two halves, and the outcome of the comparison determines which half is processed further. While linear search requires $O(n)$ comparisons, binary search requires only $O(\log n)$ comparisons. Clearly adaptive planning is more powerful in general.

However, adaptive planning may not be feasible for some robotic scenarios. When the robot's onboard computer is not powerful enough to process its sensors measurement in real-time, the robot cannot follow an adaptive plan. In this case, the robot should follow a nonadaptive plan to ensure it gather all the information it needs by the time it finishes the path.

2.1.3 IPP Applications

IPP problems are embedded in a range of robotic tasks. Robotic systems frequently have to move its sensors around to learn about its world before it acting on its assigned task. Hsiao (2009) modeled grasping an object under uncertainty as a POMDP. Grasping under uncertainty involves three distinct activities: information gathering to localize the object, re-orientating the object for reachability, and grasping the object. However, POMDP can be hard to scale up in general. The first activity that localizes the object can be modeled as an IPP problem.

Javdani, Klingensmith, et al. (2013) framed the problem of localizing a door knob as an adaptive submodular maximization problem. They generate a set of probing actions for a robot hand with contact sensor at the tip of its finger. After each probing action, the robot hand has to go back to a home position so that the cost of each action is always fixed regardless of the previous action for efficient greedy optimization. This problem can be framed as an IPP problem so that robot hand does not have to go back to a home position.

In underwater inspection, an autonomous underwater vehicle has to move around a submerged object to determine its nature. Hollinger et al. (2013) performs an initial coarse survey of a ship hull and model the uncertainty as a Gaussian Process (Rasmussen, 2006). Then in the planning phase, a set of informative location are greedily selected and a low-cost tour is approximated using the Traveling Salesman Problem. The planning phase can be modeled as an IPP which combines location selection and path planning.

2.2 Adaptive Stochastic Optimization

We first introduce the notations and definitions for a general class of adaptive stochastic optimization problems. We use UAV search and rescue task to illustrate the notations. We then define adaptive stochastic optimization on paths and on subsets. We also review submodularity, a property of functions which we will use to define our the marginal likelihood rate bound and the marginal likelihood bound conditions.

2.2.1 Uncertainty in Environment

Let X be the set of actions, *e.g.*, flying to grid cells at high and low altitude. Let O be the set of observations, *e.g.*, whether the UAV's sensor detects the survivor or not. An agent takes an action $x \in X$ and receive an observation $o \in O$. For example, the UAV flies to a grid cell and observe whether the sensor detects a survivor. We denote a scenario $\phi : X \to O$ as a function mapping from action to observation. Each scenario corresponds to an environment state initially unknown to the agent, *e.g.*, the true position of the survivor. If ϕ is the scenario corresponding to the true environment state, $\phi(x)$ specifies the observation the agent will receive at after taking action x. For

instance, the survivor's position determines whether the UAV receives a positive sensor reading at every grid cell.

2.2.2 Agent's knowledge of Environment

We adopt a Bayesian approach to represent an agent's belief of the environment. We denote a random scenario as Φ and use a prior distribution $\rho(\phi) = \mathbb{P}[\Phi = \phi]$ over the scenarios to represent our prior knowledge of the world. In the UAV search and rescue example, we encode the prior knowledge of likely area to contain the survivor as a probability distribution over the possible survivor's positions. The environment can be viewed as a random scenario drawn from the prior ρ. After taking a subset of actions $S \subseteq X$ and receiving observations, the agent's experience forms a *history*, which can be written as the set of tuples of actions and the observation received, $\psi = \{(x_1, o_1), (x_2, o_2), \dots\}$. The history of the UAV in our example is the set of grid cells visited and the corresponding sensor readings for instance. The domain of ψ, denoted $\text{dom}(\psi)$, is the set of actions in ψ. We say that a scenario ϕ is consistent with a history ψ when the observations of the scenario never contradict with the history for all action and observation tuples in the history, i.e. $\phi(x) = o$ for all $(x, o) \in \psi$. That is, an environment state, such as the survivor's position has not been ruled out given the history,*i.e.*, the sensors readings after flying to various grid cells. We denote this by $\phi \sim \psi$. We can also say that a history ψ' is consistent with another history ψ if $\text{dom}(\psi') \supset \text{dom}(\psi)$ and $\psi'(x) = \psi(x)$ for all $x \in \text{dom}(\psi)$.

2.2.3 Objective Function

An agent's goal can be characterized by a stochastic set function $f : 2^X \times O^X \to \mathbb{R}$. In this thesis, we assume the objective functions are pointwise monotone *i.e.*, $f(A, \phi) \geq f(B, \phi)$ for any scenario ϕ and for all subsets $A \subseteq B \subseteq= X$. Hence, $\max_{S' \subseteq X} f(S', \phi) = f(X, \phi)$ for all scenarios ϕ. The function f measures progress toward the goal given the actions taken and the true scenario. The objective function for UAV search and rescue problem for instance, is the version space function which is the sum of prior probabilities of potential survivor's positions eliminated from consideration given the grid cells visited and sensor readings received. Given a scenario,

the function f becomes a set function $f' : 2^X \to \mathcal{R}$. The agent achieves its goal when the function has maximum value given the actions taken $S \subseteq X$ and the scenario ϕ that corresponds to the true environment state, *i.e.*, $f(S, \phi) = f(X, \phi)$. *e.g.*, the UAV search and rescue is completed when all potential survivor's positions are eliminated except the one containing the survivor. We say the agent *covers* the function f when it achieves its goal.

2.2.4 Policies

An agent's strategy for adaptively taking actions can be denoted by a policy π, which is a mapping from history to the next action. For instance, a π tells the UAV which grid cell for to fly to next given the grid cells visited and whether it has a positive sensor at those cells or not. The policy π can be represent as a policy tree. Each node of the policy tree is labeled with an action $x \in X$, and each edge is labeled with an observation $o \in O$. To execute such a policy, the agent starts by taking the action at the root node and receives an observation o. It then follows the edge labeled with o and takes the action at the child node. This is repeated until the agent reaches a leaf node of the policy.

2.2.5 Adaptive Stochastic Optimization on Paths

Formally, an adaptive stochastic optimization problem on paths consists of the tuple (X, d, ρ, O, r, f). For adaptive stochastic optimization problems on path, the set of actions X is the set of locations the agent can visit, r is the starting location of the agent, and d is a metric that gives the distance between any pair of locations $x, x' \in X$, *e.g.*, the manhattan distance between any two grid cells on high or low altitude.

We say that a policy π covers the function f when the agent executing π always achieves its goal. That is, $f(\text{dom}(\psi), \phi) = f(X, \phi)$ for all scenarios $\phi \sim \psi$, where ψ is the history when the agent executes π. For example, an agent executing π always locate the survivor if π covers the function.

The cost of the policy π, $C(\pi, \phi)$, is the length of the path starting from location r traversed by the agent until the policy terminates, when presented with scenario ϕ, *e.g.*, the distance travelled by UAV executing policy π for a particular survivor location. In

adaptive stochastic optimization on paths, we want to find a policy π that minimizes the cost of traveling to cover the function, *e.g.*, minimize the distance travelled by UAV to locate the survivor.

We formally state the problem:

Problem 2. *Given an adaptive stochastic optimization problem on paths $\mathcal{I} = (X, d, \rho, O, r, f)$, compute an adaptive policy that minimizes the expected cost*

$$C(\pi) = \mathrm{E}[C(\pi, \phi)] = \sum_{\phi} C(\pi, \phi)\rho(\phi). \tag{2.2}$$

subject to $f(dom(\psi), \phi') = f(X, \phi')$, where ψ is the history encountered when executing π on ϕ', for all ϕ'.

Our formulation of adaptive stochastic optimization problem uses the *minimum cost coverage* criteria. The dual form of this is the *maximization* problem where we are given a budget on the length of the path and we maximize the value of the stochastic set function. The dual form can be stated as:

Problem 3. *Given an adaptive stochastic maximization problem on paths $\mathcal{I} = (X, d, \rho, O, r, f, B)$, where B is the budget, compute an adaptive policy that maximizes the function f subject to $C(\pi) < B$.*

To show the equivalence between these two forms, we can do binary search on the budget B until we find the smallest B that covers the function f.

2.2.6 Adaptive Stochastic Optimization on Subsets

Adaptive stochastic optimization problems on subsets can be formally defined by a tuple, (X, c, ρ, O, f). The set of action X is a set of items that an agent may select. Instead of a distance metric, the cost of selecting an item is defined by a cost function $c : X \to \mathbb{R}$ and the cost of a policy $C(\pi, \phi) = \sum_{x \in S} c(x)$, where S is the set of items selected by π when presented with scenario ϕ.

In this thesis, we will present our algorithm and agruments for adaptive stochastic optimization on paths. The same algorithm and agruments will apply to adaptive stochastic optimization on subsets as well unless otherwise specified. To see why most

agruments for problems on paths apply to problems on subsets, we give a transformation for problems on subsets to problems on paths. We can model distance between items by a star-shaped graph where all elements are peripheral nodes connected to a root node. The distance between an item and the root node is half of the cost of selecting the item. Hence, selecting an item is represented as traveling to the corresponding peripheral node and going back the root node in a problem on paths.

2.2.7 Submodularity

We can obtain approximate solution efficiently for deterministic set function optimization when the objective function is submodular and monotone. Submodularity means that adding an item to a smaller set is more beneficial than adding the same item to a bigger set. This captures a *diminishing return* effect that is present in many natural phenomena. For example, adding a new temperature sensor when there are few sensors helps more in mapping temperature in a building than adding one when there are already many sensors. Submodularity and monotonicity allows a simple greedy heuristic (Nemhauser, Wolsey, and Fisher, 1978) which always add the item that maximally increase the objective value to perform near-optimally for set function optimization. Submodularity is relevant to informative path planning and many real world phenomena because it captures the "diminishing return" of selecting a new location as the size of visited set increases. Information can often be quantified using submodular functions such as mutual information (Caselton and Zidek, 1984) and version space function (Tong and Koller, 2002).

Submodular Set Function

Given a finite set X and a function on the set of subsets of X, $f : 2^X \rightarrow \mathbb{R}$, the function f is submodular if

$$f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$$

for all $A, B \subseteq X$. It is also *monotone* if $f(A) \leq f(B)$ for all $A \subseteq B$.

One formulation of submodular set function optimization is the minimum cost coverage problem. The goal is to find a subset $A \subseteq E$ such that $f(A) = \max_B f(B) =$

$f(E)$. We say that a function f is covered when the set of items gives the maximum value of the function. In this thesis, we focus on minimum cost coverage.

For minimum cost coverage problem, we can achieve near-optimal performance with a greedy selection policy that always choose the element with highest marginal gain to cost ratio, *i.e.* $\max_{x \in X \setminus S}(f(S \cup \{x\}) - f(S))/c(x)$, where S is the subset of elements selected.

Lemma 1. *Given a submodular set function $f : X \to \mathbb{R}$, let π^G be the greedy selection policy. We have,*

$$C(\pi^G) \leq \left(1 + \ln \frac{f(X) - f(\emptyset)}{f(X) - f(S^{T-1})}\right) C(\pi^*)$$

where the subset S^{T-1} is the set of elements selected before the last step of the greedy policy (Wolsey, 1982).

Submodular Orienteering

The goal of submodular orienteering problem is to find a path that maximizes a submodular function given a budget on the movement cost. Given a set of locations X, a metric d that gives the distance between any pair of locations $x, x' \in X$, a starting location r, and a submodular function f of the set of locations, the submodular orienteering problem seeks to find a tour starting from r that covers the function f.

By modeling information gathering as a submodular function and apply the this procedure for submodular orienteering, we can compute a nonadaptive path for IPP problems in polynomial time (see Section 3.3.1).

Chekuri and Pal (2005) gives a submodular orienteering algorithm but it runs in quasi-polynomial time. We give a SUBMODULARORIENTEER procedure that runs in polynomial time to approximate solution to a submodular orienteering problem. In the first step, we compute an approximation for distance metric d with a tree (Fakcharoenphol, Rao, and Talwar, 2003). Then we run a greedy approximation algorithm (Calinescu and Zelikovsky, 2005) for Polymatroid Steiner tree problem with the submodular function and approximation tree as input. Finally, we apply Christofides' metric TSP (Christofides, 1976) to obtain an approximate solution.

Lemma 2. *Assuming the submodular function f is integer-valued, the* SUBMODULARORIENTEER *procedure in RAC computes a 2α-approximation to the Submodular orienteering tour with $\alpha \in O((\log|X|)^{2+\epsilon} \log \nu)$ and $\nu = f(X)$ for any $\epsilon > 0$.*

Proof. The greedy approximation in SUBMODULARORIENTEER computes an α-approximation T to the optimal polymatroid Steiner tree T^*, with $\alpha \in O((\log|X|)^{2+\epsilon} \log \nu)$, where ν is the required value (Calinescu and Zelikovsky, 2005). The total edge-weight of an optimal polymatroid Steiner tree, $w(T^*)$, must be less than that of an optimal submodular orienteering tour, W^*, as we can remove any edge from a tour and turn it into a tree. Thus, $w(T) \leq \alpha\, w(T^*) \leq \alpha\, W^*$. Applying Christofides' metric TSP to the vertices of T produces a tour τ, which has weight $w(\tau) \leq 2w(T)$, using an argument similar to that in (Christofides, 1976). It then follows that $w(\tau) \leq 2\alpha W^*$. In other words, SUBMODULARORIENTEER obtains a 2α-approximation to the submodular orienteering tour. □

2.3 Informative Path Planning, Adaptive stochastic optimization, and Related Problems

If we ignore robot movement cost, adaptive IPP problems become Bayesian active learning problems. Bayesian active learning aims to identify the true hypothesis with high certainty by sequentially choosing and observing the outcomes of a set of tests where each test has a cost associated with it. Golovin, Krause, and Ray (2010) proposed a new algorithm to solve noisy Bayesian active learning by optimizing an adaptive submodular objective function and proves that it is competitive with the optimal adaptive policy. RAC uses the same objective function 4.6 but applies it on paths.

Also closely related to Bayesian active learning, IPP, and adaptive stochastic optimization is pool-based active learning. In pool based active learning, training data are sequentially chosen and labeled from a pool of unlabeled examples. The objective is identify a hypothesis that achieves good prediction performance after choosing a small number examples to label.

Another class of problem related to IPP and adaptive stochastic optimization is the model-based Bayesian reinforcement learning problem. Model-based Bayesian rein-

forcement learning generalizes IPP's action set from robot movement actions to any finite set of actions with non-deterministic effects and its cost/reward function from movement cost to arbitrary function of state and action. In model-based Bayesian reinforcement learning problem, we maintain a probability distribution over the unknown parameters of a Markov decision process (MDP) model. The objective is to maximize its long term cumulative reward. The key challenge in Bayesian reinforcement learning is to optimize the exploration/exploitation trade off: to decide between sacrificing short term reward to reduce uncertainty about its unknown model parameter (exploration) or maximizing short term reward given its current information it has (exploitation). The unknown MDP parameters are similar to true hypothesis in IPP and scenario in adaptive stochastic optimization where it can only be unveiled by trying new actions or locations.

Although active localization (Fox, Burgard, and Thrun, 1998) and simultaneous localization and mapping (SLAM) (Feder, Leonard, and C. Smith, 1999) bear some similarity to IPP, they are in fact different, because IPP assumes that the robot location is fully observable. Reducing active localization or SLAM to IPP incurs significant representational and computational cost.

IPP, as well as other information-gathering tasks mentioned above, can all be modeled as POMDPs (see Section 2.4) which provide a general framework for planning under uncertainty. However, solving large-scale POMDP models near-optimally remains a challenge, despite the dramatic progress in recent years (Pineau, Gordon, and Thrun, 2003; T. Smith and Simmons, 2005; Kurniawati, Hsu, and Lee, 2008). The underlying structure of IPP allows simpler and more efficient solutions.

2.4 POMDP

Partially Observable Markov Decision Processes provide a principled and general planning and decision-making framework for acting optimally in partially observable domains. POMDP was first introduced to the operation research community (Smallwood and Sondik, 1973) and was brought into the artificial intelligence community by Kaelbling, Littman, and A. R. Cassandra (1998) as principled way to handle uncertainty. This section first explains the components and important concepts of POMDPs. Next, we draw the connections between IPP and POMDP. Finally, we review a few classic ex-

act solution algorithms to gain an understanding of the challenges involved in POMDP planning.

2.4.1 Model Description

Formally, a POMDP is specified as a tuple $(S, A, O, T, Z, R, \gamma)$. In practice, each of the components must be specified by domain expert or learned from data. Here, we describe the components in details.

State space S

The set of states S models the world. The state $s \in S$ should contain all information relevant to the planning task. The number of states can be finite, countably infinite, or continuous.

Action space A

An agent seeks to maximize its total reward by taking a sequence of actions from the set A. These are the choices available to the agent to take at every step to achieve its overall goal. The number of actions may be finite, countably infinite, or continuous.

Transition function T

In each time step, the agent lies in a state $s \in S$, takes an action $a \in A$, and moves from a start state s to an end state s'. Due to uncertainty in action effect, the world has a certain probability of transiting into any state in S. The stochastic action effects is captured by the transition function $T(s, a, s') = p(s'|s, a)$, which denotes the probability of transiting to state s' when action a is executed in state s.

Observation space O

In POMDP, the agent is not directly aware of its current state. Instead, the agent makes an observation $o \in O$ through its sensors after executing an action. Observation space O denotes the set of possible measurements that the agent's sensors can perceive from the world. Note that irrelevant information may be perceptible by the agent.

Observation function Z

An observation provides information on its new state s' after taking an action. Due to the uncertainty in observation, the observation result $o \in O$ is again modeled as a conditional probability function $Z(s', a, o) = p(o|s', a)$, which denotes the probability of observing o in state s' after taking action a.

Reward function R

To elicit desirable agent behavior, we define a suitable reward function $R(s, a)$. In each step, the agent receives a real-valued reward $R(s, a)$, if it takes action a in state s. The agent's goal is to maximize its expected total reward by choosing a suitable sequence of actions.

Discount factor γ

The goal of a decision theoretic agent is to maximize the reward gained over some number of time-steps. The horizon h is the number of time-steps an agent needs to plan for. The discount factor γ weighs the reward received at different time-step such that reward at time t is discounted by γ^t. When the horizon is infinite, we typically specify a discount factor $\gamma \in (0, 1)$ so that the total reward is finite and the problem is well defined. In practice, the discount factor is used to induce an agent to finish its task as quickly as possible to maximize its discounted reward. This thesis assumes infinite horizon POMDPs with a discount factor strictly less than 1, unless otherwise stated.

2.4.2 Policies

The solution to a POMDP is an optimal *policy* that maximizes the expected total discounted reward. A *policy* is an action strategy that specifies the action an agent should take at every time step based on information it has. A POMDP agent does not have complete knowledge of the state. All the information it has is the initial probability distribution over s, known as *initial belief* b_0 and sequence of action executed and observations received, known as *history*. Roughly speaking, a policy is a mapping from agent's information to action. To be more precise, we now introduce the concept of *belief*.

Belief

The information an agent has may be summarized by a probability distribution over s known as the *belief*. A belief is a sufficient statistic for a initial belief and history. A POMDP may be viewed as a special case of MDP over *belief*-state.

Given current belief b, action a executed by the agent, and observing observation o, the next belief b_{ao} can be computed as a posterior probability distribution using the Bayes rule as follows:

$$\begin{aligned} b_{ao}(s') &= \eta Z(s', a, o) \sum_{s \in S} T(s, a, s') b(s) \\ &= \eta \sum_{s \in S} p(s'|s, a) p(o|s', a) b(s) \end{aligned}$$

where η is a normalizing constant. Hence, a policy is mapping from *belief* to action.

Policy Representations

The policy π for a t-step horizon POMDP can be represented as a policy tree shown in figure 2.3. Each node in the tree represents a particular history and dictates the action to take when agent is at the node. An agent with t-steps to go executes the t-step plan starting from the root node. When the agent is at a node, it executes the action associated with the node and follows the edge that is labeled by the observation it received to next policy sub-tree. The agent recursively traverse the policy tree until it has no more step to go.

Value functions for POMDPs

We now look at the expected discounted reward an agent can earned from executing a policy π. Consider the case where π is a one step policy tree (a single root node with one action), the value of executed it in state s is:

$$V_\pi(s) = R(s, a(\pi))$$

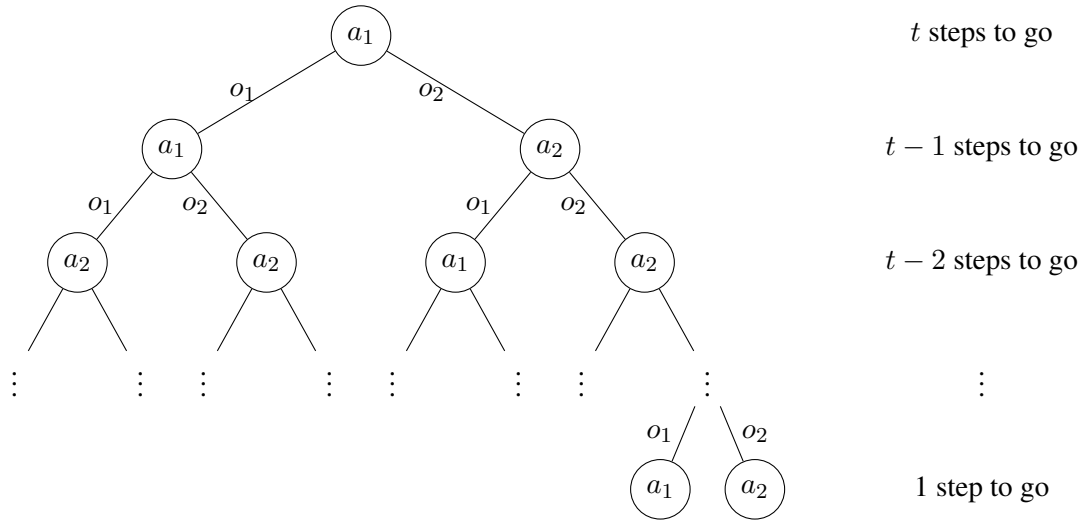

Figure 2.3: A t-step policy tree.

where $a(\pi)$ is the action specified by the root node. In the case where π a t-step policy, then

$$\begin{aligned} V_\pi(s) &= R(s, a(\pi)) + \gamma \cdot (\text{expected value of future}) \\ &= R(s, a(\pi)) + \gamma \cdot \sum_{s' \in S} T(s, a(\pi), s') \sum_{o \in O} Z(o, s', a(\pi)) V_{o(\pi)}(s') \end{aligned}$$

where $V_{o(\pi)}$ gives the $t-1$-step policy that is the child node of π linked by edge o.

Since an agent does not have complete knowledge of its world state but has a belief state b, the value of executing a policy in belief state b is then:

$$V_\pi(b) = \sum_{s \in S} b(s) V_\pi(s) \tag{2.3}$$

and the optimal t-step value of a starting belief b is the value of executing the best policy tree such that:

$$V_t(b) = \max_{\pi \in \Pi} b \cdot V_\pi(b)$$

where Π is the finite set of all policy tree.

Representing the value function can be tricky because its domain is $|S| - 1$ dimensional continuous. Fortunately, an important geometric insight due to Smallwood and Sondik (1973) allows us to represent value functions in a compact form. Notice from equation 2.3 that each policy tree π induces a function that is linear in b. We call this

linear function together with the first action of the policy tree an α-*vector*. For finite horizon t, the set of t-step policy tree is finite. Therefore, the optimal value function can be represented by a finite collection of α-vectors. The t-step value function V_t is the upper surface of this collection. Hence, the value function is piecewise linear and convex (see Figure 2.4).

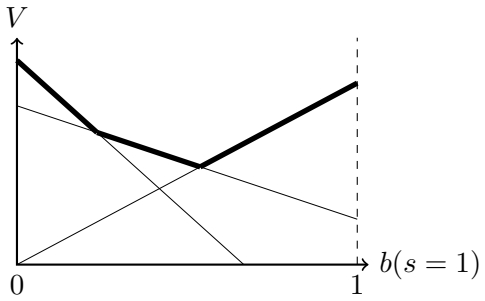

Figure 2.4: Optimal value function for a t-step 2-state POMDP (states are s=0 and s=1). Due to simplex constraint, belief for 2 state POMDP can be expressed as a point on x-axis. Value function is upper envelope of the collection of alpha vectors.

2.4.3 POMDP and its special cases

POMDP contains many important classes of problems as special cases, a few of which we will discuss in this thesis. Table 2.1 summarizes the key components of these special cases in terms of a POMDP model. To simplify presentation, we factor the POMDP's state variable into observable state variable X and partially observable state variable Y. Optimal decision tree is the simplest problem out of the special cases we consider. Its Y states are the hypotheses and actions are tests that reveal information about the true hypothesis. The cost of each test is a constant; its aim is to minimize sum of cost of tests to identify the true hypothesis. Both IPP and adaptive stochastic optimization generalizes optimal decision tree to have locations on a metric as its X state. Their actions are deterministic movement to each location and cost of each action is the distance between the current location and the destination location. Adaptive stochastic optimization generalizes IPP to cover an objective function that depends on the Y state. Another special case of POMDP is the Bayesian reinforcement learning. A Bayesian reinforcement learning problem has an unknown underlying MDP model. Its X states are the MDP states and the Y are the unknown MDP parameters. It has a transition function of the X state that depends Y. These special cases have one common feature: they all

have partially observable state that never change; their Y state transition functions are all identity.

	Optimal Decision Tree	**IPP**	**Adaptive stochastic optimization**	**Bayesian reinforcement learning**	**POMDP**
Reward/Cost	Constant	Metric	Metric	MDP	R
X	Nil	Locations	Locations	MDP	X
Y	Hypotheses	Hypotheses	Scenarios	Unknown MDP parameters	Y
Action	Test	Locations	Locations	MDP	A
X Transition	Nil	Deterministic movement	Deterministic movement	Depends on Y state	$T(X)$
Y Transition	Identity				$T(Y)$
Observation	Test Outcome	O		X State	O
Goal	Minimize cost to identify hypothesis		Minimize cost to cover	Maximize Reward	

Table 2.1: Relationship between POMDP and its subclass

POMDP formulation of IPP

We now formally represent an IPP problem $\mathcal{I}$ as a POMDP problem $\mathcal{P}$. The state space of the corresponding POMDP $\mathcal{P}$ is $S = H \times X$ consisting of all combination of hypotheses $h \in H$ and robot location $x \in X$. The action space A of $\mathcal{P}$ contains two kinds of actions. The first kind of actions is the set of movement action is X, the set of all locations accessible by the robot. The second kind is the set of "submit" actions that consists of all hypotheses $h \in H$. The "submit" action h indicates that it is ready to declare the true hypothesis as h. The transition function T of $\mathcal{P}$ does not the hypothesis part of the state. T is a deterministic function where the robot always reaches its intended destination as specified by action and "submit" actions always terminate the process. The observation space O_P of $\mathcal{P}$ is $X \times O_I$, where O_I is the observation space of IPP $\mathcal{I}$. The observation function Z is defined such that we always observe the location of the robot on top of the observations from the sensors. The reward function R of $\mathcal{P}$ returns the movement cost from the current robot location to its destination for movement actions and a large constant G when the "submit" action that is equal to the true hypothesis is invoked.

IPP and adaptive stochastic optimization are easier problems compared to POMDP because the uncertainty is restricted to the hypothesis state and scenario that never change in the process. Nevertheless, it captures an interesting aspect of POMDP: the balance between value of information and the cost to acquire it.

2.4.4 Exact POMDP Algorithms

Finite Horizon Optimal Policy

Exact POMDP algorithms are intractable in general. We briefly reviewed some algorithms here to understand the sources of intractability. The simplest algorithm is (Monahan, 1982)'s exhaustive enumeration algorithm. It is a dynamic programming procedure that computes the optimal value function for $t+1$ steps to go V^{t+1} from optimal value function of t steps to go V^t. The Bellman's equation for POMDP is:

$$V^{t+1}(b) = \max_a \left(R(b,a) + \gamma \sum_{o \in O} p(o|a,b) V^t(b_{ao}) \right) \tag{2.4}$$

Here, $R(b,a) = \sum_{s \in S} R(s,a)b(s)$ and $p(o|a,b) = \sum_{s \in S} p(o|a,s)b(s)$ are the expectations for reward function and observation function over the state space respectively. Due to continuous nature of b, it is not feasible to compute V^{t+1} for every possible b. However, since the value function is piecewise linear and convex, V^{t+1} and V^t can be represented as collections of α-vectors Γ^{t+1} and Γ^t respectively. The idea of Monahan's algorithm is to exhaustively enumerate all possible α-vectors at $t+1$ step from Γ^t. Each α-vector in Γ^t corresponds to a t-steps policy tree. A $t+1$-step policy tree can be define as a root node and $|O|$ subtrees of t-step policy trees, hence we can enumerate the α-vectors for all $t+1$-step policy tree as follows:

$$\Gamma^{t+1} = r_a + \gamma \sum_{o \in O} T^{a,o} \alpha_i | a \in A, \alpha_i \in \Gamma^t \tag{2.5}$$

where r_a is the $|S|$ column vector such that $r_a(i) = R(a, s_i)$. $T^{a,o}$ is the $|S| \times |S|$ joint observation and transition matrix such that $T^{a,o}(i,j) = p(s_j|s_i,a)p(z|s_j,a)$.

The optimal value function for t-step POMDP is the upper envelope of the collection

of α-vectors Γ_t. This value function can be expressed as:

$$V^t(b) = \max_{\alpha_i \in \Gamma^t} b \cdot \alpha_i$$

Note that we need not explicitly store the t-step policy tree associated with the each α-vector. The optimal policy for each step is to take the action associated with the α-vector that is maximal for the current belief.

This algorithm is intractable due to the exponentially growing number of α-vectors such that $|\Gamma^{t+1}| = |A||\Gamma^t||O|$. However, very often not all α-vectors (and its corresponding policy tree) are useful. There may be α-vectors that is not part of the upper envelope of Γ^t and therefore does not contribute to the value function. Hence, we can prune *dominated* α-vectors to maintain the parsimonious set of α-vectors to represent V^t. Other exact algorithms such as the Sondik's One pass algorithm (Smallwood and Sondik, 1973), Littman *et al.*'s witness algorithm (Kaelbling, Littman, and A. R. Cassandra, 1998), and incremental pruning (A. Cassandra, Littman, and Zhang, 1997) focus on avoid generating too many α-vector and pruning the dominated α-vector efficiently.

Infinite Horizon ϵ-optimal Policy

So far we have seen that the optimal t-step value function is always piecewise linear and convex. This is not necessarily true for the infinite-horizon discounted value function. There may be infinitely many facets. However, we can approximate infinite-horizon discounted value function arbitrarily closely with finite-horizon value function. The idea is to iteratively compute a series of t-step discounted value function until the L_∞ difference between the successive value functions is bound by some δ. If δ is set to $\epsilon(1-\gamma)/2\gamma$, then the policy is ϵ-optimal. Since the last value function computed can only differ from the optimal one by at most $2\delta\gamma/(1-\gamma)$ for all belief state.

Policy Iteration

The dynamic programming procedure we seen is known as *value iteration* as it iteratively step through each time step and computing the optimal value function for it. Another approach known as *policy iteration* iteratively improves on a policy through the dynamic programming backup. Policy iteration was first proposed by Sondik (Small-

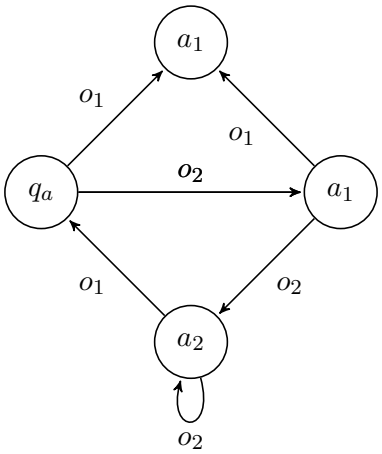

Figure 2.5: Finite State Machine

wood and Sondik, 1973) but was complex and impractical to implement. Sondik's algorithm uses mapping from finite number of polyhedral region in the belief space to action as policy, where the regions are bound by a set of linear inequalities

An improved policy iteration algorithm proposed by Hansen (E. A. Hansen, 1998) uses *finite state controller* as its policy representation. Finite state controller is simpler both conceptually and computationally compared to the policy used in Sondik's policy iteration. Here, we skip the original policy iteration and look at the improved version one.

A finite state controller is directed graph that is similar to the policy tree introduced in section 2.4.2 but has cycles. A finite state controller is executed in the same way as the policy tree except that edges may bring it to any node in the graph. Each node in the graph may be defined as a tuple of action and successor function $\langle a, \sigma \rangle$. The successor function σ is a mapping from observation to the next node.

With some initial finite state controller, the policy iteration algorithm alternates between a policy evaluation step and policy improvement step until it converges to an ϵ-optimal policy. The algorithm terminates when the Bellman residual is less than equal to $\epsilon(1-\gamma)/\gamma$ and its resultant policy is ϵ-optimal.

Policy Evaluation

Analogous to the value function of a policy tree, the value function of a finite state controller can be represented as the upper envelope of a collection of α-vectors. These

α-vectors can computed by solving the following system of linear equations:

$$V^n(s) = R_a(s) + \gamma \sum_{s' \in S} p(s'|s,a) \sum_{o \in O} p(o|a,s') V^{\sigma(n,o)}(s') \; \forall s \in S, \forall n \in \Gamma^\pi \quad (2.6)$$

where n is a node in the finite state controller and $\sigma(n,o)$ gives the successor node of n given observation o. Γ^π is the set of all nodes in the policy π. Each node in the finite state controller contributes one α-vector to the value function. In each equation, $V^n(s)$ is the expected total reward for executing the finite state controller starting from node n in state s, which is also the s component in the α-vector corresponding to node n.

Policy Improvement

In the policy improvement step, dynamic programming update (Equation 2.5) is applied to value function V that was obtained from evaluating policy π, to get a improved value function V'. Pruning is done on collection of α-vectors representing V' to keep the representation small. This dynamic programming update can be viewed as transforming the finite state controller π to an improved finite state controller π'.

To get the improved finite state controller π' from V', we need to extract a policy node from each α-vector in V'. For each α-vector in V', we create a new node with action corresponding to the action of the α-vector and successor function maps each observation to the node associated with the α-vector in V that was used to produce it. If the new node duplicates an existing node n in π, then node n remains unchanged. If the α-vector used to create the new node pointwise dominates an α-vector associated an old node, the action and successor function is replaced by that of the new one. Otherwise, the new node is simply added to the finite state machine. Existing nodes in π whose α-vectors is no longer in V' (due to pruning) is then removed. We have then transformed a finite state controller π to an improved one π'.

Performance Issues

POMDP suffers limited scalability due to two interdependent reasons. The two infamous reasons are the curse of *dimensionality* (Kaelbling, Littman, and A. R. Cassandra, 1998) and the curse of *history*. The curse of dimensionality refers to the difficulty the planner faces in reasoning about *belief* states in $|S|$-1 dimensional continuous belief

space. Even though we can represent the value function over the belief space as a set of α-vectors, the length of these α-vectors is equal to the number of states. Belief are typically represented using $|S|$-1 which gives the probability of the agent being in each state such that $b = \langle p(s = 0), p(s = 1), \ldots, p(s = |S| - 2)\rangle$. Due to simplex constraint on the belief, the probability of the last state is simply $p(s = |S| - 1) = 1 - \sum_{i=0}^{|S|-2} p(s = i)$. For efficient computation, the transition and observation are usually stored as matrices of size $|A| \times |S| \times |S|$ and $|A| \times |S| \times |O|$ respectively. This type of vector and matrix representation runs into problem when the state space is very large, or even continuous. Both the space required storing them and time taken to carry out mathematical operations on them increases with the size of state space and becomes infeasible when state space is continuous. This also limits the size of the POMDP we can practically solve. A naive approximation that discretizes the belief space and approximate the values at each discretization points will have number of discretization point exponential to the number of states.

The curse of *h*istory refers to the exponential growth in the number of distinct action-observation history with the planning horizon. Its effect can be seen in the exponential number of α-vectors in exact algorithms. Dynamic programming in POMDP may also be viewed as a breadth-first search in the belief space from some initial belief. A belief state b in the search tree is expanded by simulating all possible action-observation transition and adding the posterior belief state b_{ao} to the search tree. Each path in the search tree corresponds to a particular action-observation history. In the worst case, an optimal POMDP solution necessarily has to account for all such paths regardless of the size of the policy it produce in the end.

Chapter 3

Noiseless Informative Path Planning

3.1 Introduction

This chapter describes the *Recursive Adaptive Identification* (RAId) algorithm. RAId solves the adaptive informative path planning problem in absence of observation noise. RAId performs adaptive planning, just as binary search. Each recursive step of binary search chooses a single most discriminating comparison test that prunes half of all hypotheses. RAId shares this basic idea, but is more complex. There are two difficulties. In binary search, each comparison has the same cost. In IPP, the costs of traveling to different sensing locations vary, and we must address the key trade-off between information gain and movement cost. Further, we cannot choose sensing locations independently one at a time, because different locations provide different sensing information and moving to a location affects future choices. The main idea of RAId is to construct a near-optimal adaptive plan in each recursive step by solving a group Steiner problem (Calinescu and Zelikovsky, 2005), a generalization of minimum spanning tree problem. Under the plan, the robot traverses a subset of sensing locations and terminates the traversal when it encounters an "informative" observation, which guarantees to eliminate a significant fraction of existing hypotheses.

In the following, Section 3.3 gives the RAId algorithm. Section 3.4 shows that RAId achieves polylogarithmic approximation bound on solution cost. Empirically,

RAId outperforms greedy heuristic algorithms and nonadaptive IPP algorithm in different robotic tasks in simulation (see Section 3.5).

3.2 Related Work

IPP is important to robotics and various related fields. The importance and the difficulty of computing optimal solutions for IPP have attracted significant interest in recent years. One idea is to choose a set of "informative" sensing locations and then construct a minimum-cost tour to traverse them (Hollinger et al., 2013). The heuristic algorithm often works in practice, but it does not provide any theoretical performance guarantee. Another idea is to search for a plan over a finite horizon (Hollinger, Mitra, and Sukhatme, 2011). The guarantee, if any, is limited by the search horizon. Finite-horizon search can also be combined with sampling-based motion planning to achieve asymptotic optimality (Hollinger and Sukhatme, 2013). The NAIVE algorithm replans in each step, using a nonadaptive IPP algorithm, in order to achieve adaptivity (A. Singh, Krause, and Kaiser, 2009). It guarantees near-optimal performance when the *adaptivity gap* is small, in other words, when adaptive planning does not have significant advantage over nonadaptive planning. Unfortunately the adaptive gap can be exponentially large even for very simple problems (Hollinger et al., 2013). This is unsurprising in light of the well-known benefit of acting adaptively (Dean, Goemans, and Vondrdk, 2004; Golovin and Krause, 2011). Furthermore, to achieve nontrivial performance bound, NAIVE requires explicit construction of a submodular function with the *locality* property (A. Singh, Krause, and Kaiser, 2009). This is not always easy or possible. A strength of NAIVE is its ability to handle noisy observations. This chapter makes the assumption of noiseless observations, though we are extending the algorithm to handle noisy observations (see Section 3.6).

IPP is closely related to the adaptive traveling salesman (ATSP) problem (Gupta, Nagarajan, and Ravi, 2010). In contrast to the standard TSP, the traveling salesman here services only a subset of locations with *requests*, but does not know this subset initially. When the salesman arrives at a location, he finds out whether there is a request there. The goal is to find an adaptive strategy for the salesman to service all requests and minimize the expected cost of traveling. IPP contains ATSP as a special case. Each

hypothesis represents a subset of locations with requests. Each "sensing" operation is binary and answers the query whether the current location has a service request or not. RAId has its root in the isolation algorithm for ATSP (Gupta, Nagarajan, and Ravi, 2010). To provide the theoretical performance bound, the isolation algorithm uses linear programming in the inner loop to solve the group Steiner problem (see Section 3.3.1). This is impractical. RAId solves the more general IPP problem, which allows arbitrary hypothesis space and removes the restriction of binary sensing. To solve the group Steiner problem, it uses a combinatorial approximation algorithm (Calinescu and Zelikovsky, 2005) that is far more effective in practice.

Our IPP algorithm contains three main ingredients: information gathering, robot movement cost, and adaptivity. It touches on several important research topics, which contain one or two, but not all three ingredients. If we focus on information gathering only and ignore location-dependent robot movement cost, IPP becomes sensor placement, view planning, or ODT, which admits efficient solutions through, *e.g.*, submodular optimization, in both non-adaptive (Krause and Guestrin, 2009) and adaptive settings (Golovin and Krause, 2011; Javdani, Klingensmith, et al., 2013; Javdani, Chen, et al., 2014). Our work does not rely on adaptive submodularity in either the algorithm or the proofs. If we account for movement cost, there are several nonadaptive algorithms with performance guarantee, *e.g.*, (Hollinger, S. Singh, et al., 2009; A. Singh, Krause, Guestrin, et al., 2009).

Although active localization (Fox, Burgard, and Thrun, 1998) and simultaneous localization and mapping (SLAM) (Feder, Leonard, and C. Smith, 1999) bear some similarity to IPP, they are in fact different, because IPP assumes that the robot location is fully observable. Reducing active localization or SLAM to IPP incurs significant representational and computational cost.

IPP, as well as other information-gathering tasks mentioned above, can all be modeled as partially observable Markov decision processes (POMDPs) (Kaelbling, Littman, and A. R. Cassandra, 1998), which provide a general framework for planning under uncertainty. However, solving large-scale POMDP models near-optimally remains a challenge, despite the dramatic progress in recent years (Pineau, Gordon, and Thrun, 2003; T. Smith and Simmons, 2005; Kurniawati, Hsu, and Lee, 2008). The underlying

structure of IPP allows simpler and more efficient solutions.

3.3 Algorithm

3.3.1 Preliminaries on Group Steiner Trees

RAId makes critical use of an seemingly unrelated problem, the *group Steiner* problem, to trade off information gain and robot movement cost. A group Steiner problem is defined by two elements. One is an edge-weighted graph $G = (V, E, W_E)$. The other is a collection of *groups* $\mathcal{V} = \{V_1, V_2, \ldots, V_m\}$ with corresponding group-weights $W_{\mathcal{V}} = \{\nu_1, \nu_2, \ldots, \nu_m\}$. Each group V_i contains a subset of vertices in V. A subgraph of G *covers* a group $V_i \subseteq V$ if the subgraph contains at least one vertex in V_i. In the standard group Steiner problem, the goal is to find a *minimum-edge-weight tree* that covers a sub-collection of groups with total group-weight at least ν, for some given constant ν.

The group Steiner algorithm used in RAId constructs a tree T in a greedy manner (Calinescu and Zelikovsky, 2005). Define the *density* of a tree as the ratio of its total edge-weight over the total group-weight of the groups covered by the tree. Each step of the greedy algorithm constructs a low-density subtree T' and adds it to a partial solution T being constructed. The greedy step repeats until the total weight of groups covered by T exceeds the target ν. Each subtree T' roughly is constructed by recursively applying the greedy algorithm on its children nodes with a series of different target values ν and then picking the lowest density one among all the subtrees found. It can be shown that a low-density subtree always exists and can be constructed efficiently. The algorithm uses a series of technical ideas to limit the number of recursive calls so that it runs in polynomial time. Furthermore, the union of low-density trees remains a low-density tree, which provides an approximately optimal solution to the group Steiner problem.

Theorem 3 (Calinescu and Zelikovsky (2005))**.** *Assume that the group-weights of a group Steiner problem are represented as non-negative integers. For any constant* $\epsilon > 0$*, there is a polynomial-time algorithm that computes a near-optimal group Steiner tree within a factor* $O(\log |V|)^{2+\epsilon} \log \nu)$ *of the optimal one.*

The constant ϵ is a parameter that can be tuned to improve the approximation ratio, but

at a greater computational cost.

3.3.2 Informative Observations

Let $H_{x,o} \subseteq H$ be the subset of hypotheses consistent with observation o at x:

$$H_{x,o} = \{h \in H \mid Z_x(h, o) = 1\}.$$

Let $p(H_{x,o})$ be the sum of probabilities of hypotheses in the subset. We consider an observation o at location x *informative* if $p(H_{x,o}) \leq 0.5$ and define the *informative observation set* at $x \in X$:

$$\Omega_x = \{o \in O \mid p(H_{x,o}) \leq 0.5\}.$$

If an observation o is informative, then by definition, $H_{x,o}$ has small probability (less than 0.5), and $H \backslash H_{x,o}$, the set of hypotheses inconsistent with o, has large probability (greater than 0.5). The observation o is informative, because it narrows down the consistent hypotheses to a small set measured in probability.

Let o_x^* be the most likely observation at x: $o_x^* = \arg\max_{o \in O} p(H_{x,o})$. It is interesting to observe that there are only two possibilities for Ω_x:

$$\Omega_x = \begin{cases} O & \text{if } p(H_{x,o}) \leq 0.5 \text{ for all } o \in O, \\ O \setminus \{o_x^*\} & \text{otherwise.} \end{cases}$$

Consider the UAV search example again. Initially, the observation $o = 1$ at every low-altitude location x is informative, as $p(H_{x,1}) = 1/64 \leq 0.5$. The notion of being informative is intuitively correct here, because the observation $o = 1$ at a low-altitude location identifies the target location exactly. In contrast, the observation $o = 0$ is not informative at any low-altitude location x, as $p(H_{x,0}) = 63/64 > 0.5$. It eliminates a single inconsistent hypothesis with probability $1/64$ and does not help narrow down consistent hypotheses significantly.

Algorithm 1 RAId

procedure RAId($X, d, H, \rho, O, \mathcal{Z}, r$)
  if $|H| = 1$ **then**
    return H.
  else
    $\nu \leftarrow \min\big(0.5, 1 - \max_{h \in H} \rho(h)\big)$.
    $\tau \leftarrow$ GROUPSTEINERTOUR(X, $X \times X$, d, $\{X_h\}_{h \in H}$, ρ, ν), where $\tau = (x_0, x_1, \ldots, x_t)$ and $x_0 = x_t = r$.
    $(H, r) \leftarrow$ EXECUTEPLAN(τ, H, r).
    Renormalize the probability $\rho(h)$ for all $h \in H$ so that $\sum_{h \in H} \rho(h) = 1$.
    RAId($X, d, H, \rho, O, \mathcal{Z}, r$)

procedure EXECUTEPLAN(τ, H, r)
  $i \leftarrow 1$.
  repeat
    $r \leftarrow x_i$.
    Visit location r and receive observation o.
    Remove from H all hypotheses inconsistent with o.
    $i \leftarrow i + 1$.
  until $o \in \Omega_r$ or $i = t$.
  $r \leftarrow x_t$.
  Move to location r.
  return (H, r).

3.3.3 RAId

RAId is a recursive divide-and-conquer algorithm. Each recursive step constructs a near-optimal adaptive plan to traverse a subset of sensing locations in X and eliminates inconsistent hypotheses using the observations received. The traversal terminates when it reduces the probability of the current hypothesis set H by at least a half. RAId then recurses on the remaining hypotheses until only one hypothesis remains. A sketch of the algorithm is shown in Algorithm 1.

The key step in RAId is to construct a traversal that significantly reduces the current hypothesis set at a low cost. Informative observation helps in eliminating inconsistent hypotheses. If a traversal encounters an informative observation o at location x, we can eliminate all hypotheses in $H \backslash H_{x,o}$, which has probability greater than 0.5 by definition, and end the traversal. However, what happens if a traversal does not encounter any informative observations? To guarantee that each traversal reduces the probability of the current hypothesis set H by at least a half, RAId constructs and solves a group Steiner problem.

The underlying graph for the group Steiner problem is the complete graph over X, and the edge-weight between two vertices x and x' is $d(x, x')$.

Next, we define one group for every hypothesis $h \in H$:

$$X_h = \{x \in X \mid Z_x(h, o) = 1 \text{ for some } o \in \Omega_x\}, \tag{3.1}$$

which consists of all locations with *informative observations* consistent with h. The group-weight for X_h is simply $\rho(h)$. Our definition of a group implies that an uninformative observation $o \notin \Omega_x$ must be inconsistent with h at a location $x \in X_h$, because observations are noiseless and there is only one observation consistent with a given hypothesis. Thus, if a traversal encounters an uninformative observation at a location $x \in X_h$, we can eliminate h.

Finally, we set the target $\nu = \min\big(0.5, 1 - \max_{h \in H} \rho(h)\big)$. It would be desirable, but is not possible to simply set $\nu = 0.5$. If the true hypothesis has high probability, RAId may not be able to achieve substantial pruning, as the remaining hypotheses have small total probability.

RAId guarantees that a traversal constructed from the group Steiner problem prunes inconsistent hypotheses that have total probability at least ν. If the robot encounters an informative observation o at a location x during the traversal, the inconsistent hypotheses $H \backslash H_{x,o}$ have probability greater than 0.5 by definition. Now suppose that the robot encounters only uninformative observations during the traversal. At each location $x \in X_h$ along the way, the robot eliminates the hypothesis h. Each hypothesis has an associated group in the group Steiner problem. The target value ν ensures that the total weight of groups covered by the traversal is greater than ν. So is the probability of eliminated hypotheses. The formal proof is given in Lemma 4.

In Algorithm 1, the procedure GROUPSTEINERTOUR$(V, E, W_E, \mathcal{V}, W_{\mathcal{V}}, \nu)$ solves the group Steiner problem defined in Section 3.3.1. However, it computes a group Steiner *tour*, *i.e.*, a cycle in a graph-theoretic sense, instead of a tree. GROUPSTEINERTOUR consists of two steps. First, it solves for a group Steiner tree T using a greedy approximation algorithm (Calinescu and Zelikovsky, 2005). Next, it applies Christofides' metric TSP approximation algorithm (Christofides, 1976) to the vertex set of T and generates a tour. Both approximation algorithms rely critically on the metric property

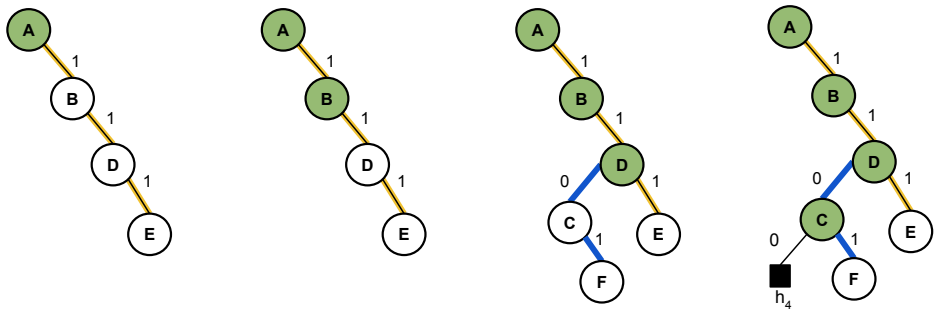

Figure 3.1: An example run of RAId

An example run of RAId with two recursive calls. Shaded edges in color indicate the two tours planned in the recursive calls. Shaded nodes indicate the locations that the robot traverses while executing the plans. We assume that observation 0 is informative and observation 1 is uninformative. (a) RAId's first recursive call generates the tour (A, B, D, E). The robot moves to the first location A on the tour and receives observation 1. (b) Since observation 1 is uninformative, the robot next moves to B and receives observation 1 again. (c) Upon receiving the first informative observation 0 at D, the robot ends the traversal. RAId replans in the second recursive call and generates a new tour (C, F). (d) The robot moves to C. It receives 0 at C and identifies the hypothesis h_4.

of the edge weight d.

RAId is an online algorithm, which interleaves planning and plan execution. In the planning phase, RAId computes a tour (Algorithm 1, line 6), which is a partial plan. The robot executes the plan by traversing the locations on the tour (Algorithm 1, line 7). At each location, the robot prunes all hypotheses inconsistent with the received observation. If the robot receives an uninformative observation, it moves to the next location on the tour. If the robot receives an informative observation or exhausts the tour, it ends the traversal and returns to the start location. RAId then replans a new tour, and the whole process repeats.

Returning to the start location simplifies the analysis in Section 3.4. However, it is not required in practice. In our experiments in Section 3.5, the robot starts the new traversal from its current location without returning. Figure 3.1 shows an example run of RAId.

3.4 Analysis

The analysis of RAId focuses two main issues:

- the total probability of hypotheses eliminated in each traversal, and
- the associated travel cost.

We proceed in two main steps. In the first step, we analyze a variant of IPP, called *rooted IPP*, in which the robot must return to the start location r in the end. Our main idea is to show that each group Steiner tour computed enables the robot to either prune inconsistent hypotheses with probability at least 0.5 or identify the true hypothesis (Lemma 4). Furthermore, the robot traversing such a tour incurs a cost not more than twice the expected cost of an optimal policy (Lemmas 5 and 6). By bounding the number of recursive calls to RAId, we then obtain a result on its performance for rooted IPP (Theorem 8). In the second step, we exploit this result to bound the performance of RAId for IPP itself (Theorem 10).

We consider only rooted IPP for Lemma 4–7 and Theorem 8.

Lemma 4. *Let $H' \subset H$ be the set of remaining hypotheses after a single recursive call to RAId. Then, either $p(H') \leq 0.5$ or $|H'| = 1$.*

Proof. In each recursive call to RAId, the robot follows a group Steiner tour τ. If it receives an observation $o \in \Omega_x$ at some location x on τ, then the robot returns to r immediately (Algorithm 1, line 19) and $p(H') = p(H_{x,o}) \leq 0.5$ by definition of Ω_x. Otherwise, the robot visits every location x on τ and receives at every x an observation $o^*_x \notin \Omega_x$. Consider $x \in X_h$ for some x on τ and $h \in H$. If the robot receives the observation $o^*_x \notin \Omega_x$ at x, then h is inconsistent with o^*_x by the definition of X_h and is pruned. Since the target of our group Steiner problem is ν, the pruned hypotheses has probability at least ν, and the remaining hypothesis set H' has probability at most $1-\nu$. If there is a single hypothesis h^* with $p(h^*) \geq 0.5$, then h^* must be the only remaining hypothesis. Otherwise, $p(H') \leq 1 - \nu \leq 0.5$. □

Next, we bound the edge-weight of an optimal group Steiner tour.

Lemma 5. *Let π^* be an optimal policy for a rooted IPP problem $\mathcal{I}$. Let W^* be the total edge-weight of an optimal group Steiner tour for $\mathcal{I}$. Then $W^* \leq 2C(\pi^*)$.*

Proof. First, we extract a path σ from an optimal policy tree π^* and use σ to construct a feasible, but not necessarily optimal solution σ_{r} to the group Steiner problem for $\mathcal{I}$.

Next, we show that the optimal policy traverses σ with probability at least 0.5. This allows us to bound the total edge-weight of σ_{r} and thus that of an optimal group Steiner tour by the cost of the optimal policy.

Let $(r, x_1, x_2, \ldots, r)$ be a path in the optimal policy tree π^* such that every edge following a node x_i in the path is labeled with the most likely observation $o^*_{x_i} = \arg\max_{o \in O} p(H_{x_i,o})$. For any subpath ϕ, $H_\phi = \{h \in H \mid Z_{x_i}(h, o^*_{x_i}) = 1 \text{ for all } x_i \text{ in } \phi\}$ is the set of hypotheses consistent with the observations received at all locations in ϕ. Let $\sigma = (r, x_1, x_2, \ldots, x_s)$ be the shortest subpath of $(r, x_1, x_2, \ldots, r)$ such that $p(H_\sigma) \leq 1 - \nu$, where the length of σ is measured in the number of nodes in the path.

We now show that the tour $\sigma_{\mathrm{r}} = (r, x_1, x_2 \ldots, x_s, r)$ is a feasible solution to the group Steiner tour problem. The key issue is to determine the total group-weight of $\mathcal{X}$, the collection of groups covered by $x_1, x_2, \ldots, x_s$. At each location x_i on σ, the robot receives an observation $o^*_{x_i}$. If a hypothesis $h \in H$ is inconsistent with $o^*_{x_i}$, then h must be consistent with some $o \neq o^*_{x_i}$, *i.e.*, $Z_{x_i}(h, o) = 1$ for $o \in \Omega_{x_i}$. Then $x_i \in X_h$ by definition. In other words, x_i covers X_h if h is inconsistent with $o^*_{x_i}$ at x_i, and $\mathcal{X} = \{X_h \mid Z_{x_i}(h, o^*_{x_i}) = 0 \text{ for some } x_i \text{ in } \sigma\}$. Since $p(H_\sigma) \leq 1 - \nu$, the total group-weight of $\mathcal{X}$ must be least ν. This proves that σ_{r} is a feasible group Steiner tour.

Now consider the subpath $\sigma' = (r, x_1, x_2, \ldots, x_{s-1})$. We have $p(H_{\sigma'}) > 1 - \nu$, as σ is the *shortest* path with $p(H_\sigma) \leq 1 - \nu$. To bound the expected cost of the optimal policy π^*,

$$C(\pi^*) = \sum_{h \in H} \rho(h) C(\pi^*, h) \geq \sum_{h \in H_{\sigma'}} \rho(h) C(\pi^*, h).$$

$H_{\sigma'}$ can be interpreted as the set of hypotheses that visit $x_1, \ldots, x_s$ but not necessarily receive $o^*_{x_s}$ at x_s. Hence for any $h \in H_{\sigma'}$, the path that leads to h in the optimal policy tree π^* must contain σ as a subpath. Thus,

$$C(\pi^*) \geq \sum_{h \in H_{\sigma'}} \rho(h) w(\sigma_{\mathrm{r}}) \geq (1 - \nu) w(\sigma_{\mathrm{r}}) \geq (1 - \nu) W^*,$$

where $w(\sigma_{\mathrm{r}})$ is the total edge-weight of the tour σ_{r}. Rearranging the inequality above, we get

$$W^* \leq \frac{1}{1 - \nu} \cdot C(\pi^*) \leq 2C(\pi^*).$$

□

Lemma 6. *If RAId computes an optimal group Steiner tour, then the robot travels a path with cost at most $2C(\pi^*)$ in each recursive step of RAId.*

Proof. In each recursive step of RAId, the robot travels a path whose cost is bounded by the total edge-weight of the group Steiner tour computed. The conclusion then follows directly from Lemma 5. □

Before moving to our first theorem, we need to connect a rooted IPP problem to its subproblems, as RAId is recursive.

Lemma 7. *Suppose that π^* is an optimal policy for a rooted IPP problem $\mathcal{I}$ with hypothesis set H and prior probability distribution ρ. Let $\{H_1, H_2, \ldots, H_n\}$ be a partition of H, and let π_i^* be an optimal policy for the subproblem $\mathcal{I}_i$ with hypothesis set H_i and prior probability distribution ρ_i, where $\rho_i(h) = \rho(h)/\rho(H_i)$ for each $h \in H_i$. Then we have*

$$\sum_{i=1}^{n} \rho(H_i) C(\pi_i^*) \leq C(\pi^*).$$

Proof. For each subproblem $\mathcal{I}_i$, we can construct a feasible policy π_i for $\mathcal{I}_i$ from the optimal policy π^* for $\mathcal{I}$. Consider the policy tree π^*. Every path from the root of π^* to a leaf uniquely identifies a hypothesis $h \in H$. So we choose the policy tree π_i as the subtree of π^* that consists of all the paths leading to hypotheses in H_i. Clearly π_i is feasible, as it identifies all the relevant hypotheses. Then,

$$\begin{aligned}
\sum_{i=1}^{n} \rho(H_i) C(\pi_i^*) &\leq \sum_{i=1}^{n} \rho(H_i) C(\pi_i) \\
&\leq \sum_{i=1}^{n} \rho(H_i) \sum_{h \in H_i} \frac{\rho(h)}{\rho(H_i)} \cdot C(\pi_i, h) \\
&= \sum_{h \in H} \rho(h) C(\pi^*, h) = C(\pi^*).
\end{aligned}$$

□

We are now ready to bound the performance of RAId for rooted IPP, under an assumption which we relax later.

Theorem 8. *Let π denote the policy that RAId computes for a rooted IPP problem. If*

RAId computes an optimal *group Steiner tour in each step, then*

$$C(\pi) \leq 2\left(\log\left(1/\delta\right)+1\right)C(\pi^*),$$

where $C(\pi)$ is the expected cost of RAId and $\delta = \min_{h\in H}\rho(h)$.

Proof. By Lemma 4, if a recursive step of RAId does not terminate, it reduces the probability of consistent hypotheses by a factor of $1/2$. For any $h \in H$, the number of recursive steps required is then at most $\log(1/\delta)+1$.

We now complete the proof by induction on the number of recursive calls to RAId. For the base case of $k = 1$ call, $C(\pi) \leq 2C(\pi^*)$ by Lemma 6. Assume that $C(\pi) \leq 2(k-1)C(\pi^*)$ when there are at most $k-1$ recursive calls. Now consider the induction step of k calls. The first recursive call partitions the hypothesis set H into a collection of mutually exclusive subsets, $H_1, H_2, \ldots, H_n$. Let $\mathcal{I}_i$ be the subproblem with hypothesis set H_i and optimal policy π_i^*, for $i = 1, 2, \ldots, n$. After the first recursive call, it takes at most $k-1$ additional calls for each $\mathcal{I}_i$. In the first call, the robot incurs a cost at most $2C(\pi^*)$ by Lemma 6. For each $\mathcal{I}_i$, the robot incurs a cost at most $2(k-1)C(\pi_i^*)$ in the remaining $k-1$ calls, by the induction hypothesis. Putting together this with Lemma 7, we conclude that the robot incurs a total cost of at most $2kC(\pi^*)$ when there are k calls. □

Finally, we use Theorem 8 to analyze the performance of RAId on IPP rather than rooted IPP. To start, we argue that a rooted IPP solution provides a good approximate solution for IPP.

Lemma 9. *An α-approximation algorithm for rooted IPP is a 2α-approximation algorithm for IPP.*

Proof. Let C^* and C_{r}^* be the expected cost of an optimal policy for an IPP problem $\mathcal{I}$ and for a corresponding rooted IPP problem $\mathcal{I}_{\mathrm{r}}$, respectively. Since any policy for $\mathcal{I}$ can be turned into a policy for $\mathcal{I}_{\mathrm{r}}$ by retracing the solution path back to the start location, we have $C_{\mathrm{r}}^* \leq 2C^*$. An α-approximation algorithm for rooted IPP computes a policy π for $\mathcal{I}_{\mathrm{r}}$ with expected cost $C_{\mathrm{r}}(\pi) \leq \alpha C_{\mathrm{r}}^*$. It then follows that $C_{\mathrm{r}}(\pi) \leq \alpha C_{\mathrm{r}}^* \leq 2\alpha C^*$ and this algorithm provides a 2α-approximation to the optimal solution of $\mathcal{I}$. □

To obtain our main result, we need to address two remaining issues. First, Theorem 8 assumes that RAId computes an optimal group Steiner tour. This is, however, not achievable in polynomial time under standard assumptions. RAId uses a polynomial-time greedy algorithm (Calinescu and Zelikovsky, 2005) that computes a group Steiner tree T with a guaranteed approximation factor. It then applies Christofides' metric TSP algorithm (Christofides, 1976) to the vertex set of T and generates a tour, instead of traversing T directly, because Christofides algorithm provides a guaranteed $3/2$-approximation to the optimal TSP tour. Second, the greedy group Steiner approximation algorithm assumes integer group-weights. To apply this algorithm and obtain the approximation bound, we assume that the prior probabilities are coded in non-negative integers. We remove the renormalization step (Algorithm 1, line 8) and make other minor changes accordingly. Normalization of probabilities is not necessary for RAId. It only simplifies presentation.

Theorem 10. *Let $\mathcal{I} = (X, d, H, \rho, O, \mathcal{Z}, r)$ be an IPP problem. Assume that the prior probability distribution ρ is represented as non-negative integers with $\sum_{h\in H} \rho(h) = P$. Let $\delta = \min_{h\in H} \rho(h)/P$. For any constant $\epsilon > 0$, RAId computes a policy π for $\mathcal{I}$ in polynomial time such that $C(\pi) \in O((\log|X|)^{2+\epsilon} \log P \log(1/\delta) C(\pi^*))$.*

Proof. In the group Steiner problem for $\mathcal{I}$, the vertex set is X. From Theorem 3, the greedy approximation in RAId computes an α-approximation T to the optimal group Steiner tree T^*, with $\alpha \in O((\log|X|)^{2+\epsilon} \log P)$. The total edge-weight of an optimal group Steiner tree, $w(T^*)$, must be less than that of an optimal group Steiner tour, W^*, as we can remove any edge from a tour and turn it into a tree. Thus, $w(T) \leq \alpha\, w(T^*) \leq \alpha\, W^*$. Applying Christofides' metric TSP to the vertices of T produces a tour τ, which has weight $w(\tau) \leq 2w(T)$, using an argument similar to that in (Christofides, 1976). It then follows that $w(\tau) \leq 2\alpha W^*$. In other words, RAId obtains a 2α-approximation to the optimal group Steiner tour. Putting this together with Theorem 8 and Lemma 9, we get the desired approximation bound. The algorithm clearly runs in polynomial time.

The computational bottleneck of RAId lies in the recursive calls to GROUPSTEINERTOUR. RAId makes at most $\log 1/\delta$ calls to GROUPSTEINERTOUR. The running time of GROUPSTEINERTOUR is dominated by the greedy group Steiner procedure, which has running time $O((|X| + |H|)(b \cdot \beta \cdot \log P \cdot \Delta \cdot \log_{1+\lambda} b)^{\beta})$, where $\beta = O\left(\frac{\log|X|}{\epsilon \log\log|X|}\right)$,

Table 3.1: The main characteristics of algorithms under comparison.

	RAId	IG, IG-Cost	IG-Cost-2	NAId-Replan
Nonmyopic	yes	no	finite horizon	yes
Adaptive	yes	replanning	replanning	replanning

$\Delta = O(\log |X|)$, $1/\lambda = \log |X|$, and $b = \Delta(1 + 1/\lambda)(1 + \lambda)$ (Calinescu and Zelikovsky, 2005). Clearly RAId runs in polynomial time. □

IPP is an NP-hard optimization problem. RAId provides a polylogarithmic approximation algorithm that runs in polynomial time. The computational bottleneck of RAId lies in the recursive calls to GROUPSTEINERTOUR, which computes an approximate solution to the group Steiner problem. The running time of GROUPSTEINERTOUR is roughly linear in the number of hypotheses and the number of locations.

3.5 Experiments in Simulation

3.5.1 Setup

For comparison, we implemented three types of algorithms: greedy algorithms, finite-horizon lookahead search, and submodular optimization. They represent the classes of methods available from existing literature we reviewed in Section 3.2. The experiments focus on performance comparison of two main differentiating characteristics of these algorithms: planning horizon and adaptivity. See Table 3.1 for a summary and the subsections below for detailed explanation.

Greedy Algorithms

We first describe two greedy algorithms, which are simple and widely used in practice: *information gain* (IG) and *information gain with cost* (IG-Cost). Let Q denote the random variable representing the true hypothesis. Suppose that the robot is currently located at x. If it receives observation o at the next location x', the information gain is $\mathbb{H}(Q) - \mathbb{H}(Q|x', o)$, where $\mathbb{H}$ denotes the Shannon entropy. Entropy measures the uncertainty in a random variable. Reducing entropy is the same as gaining information.

IG always chooses the next location x' to maximize the *expected* information gain

$$f_{\mathrm{IG}}(x') = \sum_{h \in H} \sum_{o \in O} \Big(\mathbb{H}(Q) - \mathbb{H}(Q \mid x', o) \Big) p(o|x', h) p(h).$$

in a greedy manner. When there are only two observations, IG is equivalent to *generalized binary search* (Zheng, Rish, and Beygelzimer, 2005).

To account for robot movement cost, IGC maximizes information gain per unit movement cost

$$f_{\mathrm{IGC}}(x') = \sum_{h \in H} \sum_{o \in O} \frac{\mathbb{H}(Q) - \mathbb{H}(Q \mid x', o)}{d(x, x')} p(o|x', h) p(h),$$

again in a greedy manner.

Greedy algorithms are *myopic*: they do not reason over the long term. They achieve limited adaptivity by replanning in each step.

Finite-Horizon Lookahead Search

To alleviate the weakness in greedy algorithms, one idea is to search over a finite horizon k for a depth-k policy tree (Figure 2.1) with the best expected heuristic value (Hollinger, S. Singh, et al., 2009). We use IG-Cost as the heuristic and call the resulting algorithm IG-Cost-k. The original greedy IG-Cost algorithm corresponds to IG-Cost-1. IG-Cost-k replans in each step. It performs a lookahead search for the best policy tree, and the robot executes the first step of the chosen policy. The process then repeats. Since each policy tree node chooses among $|X|$ sensing locations and branches on $|O|$ observations, there are $O(|X|^k |O|^{k-1})$ policy trees of depth k. Clearly, with large $|X|$ and $|O|$, k must be kept small for the finite-horizon search to be practical. Some of the tasks in our experiments can have up to 170 sensing locations and 22 observations at each location. We had to set $k = 2$ to keep the total running time reasonable.

The planning horizon IG-Cost-k is longer than that of its greedy counterpart, but is bounded by the finite constant k a priori. IG-Cost-k achieves limited adaptivity through replanning, just as the greedy algorithms.

Submodular Optimization

Submodular optimization is another interesting idea for IPP, *e.g.*, the NAIVE algorithm (Section 3.2). NAIVE requires a submodular function with the locality property for guaranteed performance. It is unclear how to construct such functions for the tasks in our experiments. Instead, we use an expected version space reduction function to search for a near-optimal path σ:

$$f_{\mathrm{VSR}}(\sigma) = 1 - \sum_{h \in H} \big(p(H_{\sigma,h}) - p(h)\big)p(h),$$

where $H_{\sigma,h}$ denotes the set of hypotheses with the same observation as h at every location on σ. Intuitively, maximizing f_{VSR} results in a path that maximally reduces the set of confounding hypotheses. If a path σ always eliminates all confounding hypotheses, then $p(H_{\sigma,h}) = p(h)$ for all $h \in H$, and $f_{\mathrm{VSR}}(\sigma) = 1$. The function f_{VSR} is submodular, but may not satisfy the locality property required by NAIVE.

Finding a minimum-cost path σ such that $f_{\mathrm{vsr}}(\sigma) = 1$ is a minimum-cost submodular coverage problem. To solve it, we use the greedy polymatroid Steiner algorithm (Calinescu and Zelikovsky, 2005). Although both submodular optimization and RAId make use of the polymatroid Steiner algorithm (group Steiner algorithm is a special case), they differ in their objectives. Submodular optimization searches for an open-loop plan, *i.e.*, a path that maximizes f_{VSR}. It does not consider future observations during planning and is nonadaptive.

There are two ways to execute the computed path. One is to have the robot traverse every location on the path until the end. Alternatively, NAIVE replans in every step. It plans a path, but the robot visits only the first location on the path. The process then repeats. We follow NAIVE's approach: it is more adaptive, but has a higher computational cost. We call the resulting algorithm *nonadaptive hypothesis identification with replanning* (NAId-Replan).

An alternative way of solving the minimum-cost submodular coverage problem is the recursive greedy algorithm (Chekuri and Pal, 2005) used in A. Singh, Krause, and Kaiser (2009). We implemented this algorithm, but found it too slow to be practical for our tasks.

In summary, NAId-Replan is nonmyopic. It shares the same basic idea as RAId, but performs nonadaptive planning. It achieves limited adaptivity through replanning. NAId-Replan is also related to NAIVE. It performs submodular optimization, but the submodular function used does not possess the locality property required by NAIVE for theoretical performance guarantee.

We implemented all algorithms in the Clojure language and compared their performance on a set of tasks in simulation. For each task, we ran the algorithms on every hypothesis in H and calculated the average policy cost weighted by the prior probabilities. The running times were obtained on a computer server with an Intel Xeon 2.4GHz processor.

3.5.2 Results

Overall, RAId obtains the best or nearly the best policies in all tasks in our experiments, according to their average policy costs (Table 3.2). The other algorithms may perform well in some tasks, but very poorly in others. While RAId has performance guarantees, it will be not surprising for greedy algorithm to outperform RAId on some problems due to the approximation factors in the performance bound. In general, it is difficult to tell the effectiveness of an algorithm in advance. As IG-Cost is easy to implement, one could try it as a first approach for the problem of interest.

While the average policy cost is our main performance measure, we also report the total *planning* time for completeness (Table 3.3). RAId is slower that the greedy algorithms. This is expected, as greedy algorithms perform only short-term planning. RAId are much faster than IG-Cost-2 and NAId-Replan, which both perform longer-term planning.

Although our implementation is not optimized as a result of the implementation language, the running times, which are on the order of seconds for these moderate-scale tasks, are useful for a range of online robot planning tasks.

2-Star Graph

We start with a simple example to gain some understanding of the key issues. There are a total of 2^n possible hypotheses $H = \{0, 1, 2, \ldots, 2^n - 1\}$, with equal probability of

Table 3.2: Average cost of a computed policy over all hypotheses.

	Cost				
	RAId	IG	IG-Cost	IG-Cost-2	NAId-Replan
2-Star (d=10, n=5)	**19.0**	25.3	32.9	33.9	19.0
2-Star (d=10, n=6)	**21.0**	27.9	22.3	52.5	21.0
2-Star (d=53, n=6)	65.0	102.1	**62.0**	68.9	78.8
2-Star (d=53, n=7)	**66.0**	102.4	127.4	118.5	66.0
2-Star (d=53, n=8)	**68.0**	100.9	257.7	258.7	**68.0**
Adaptive 2-Star	**73.0**	84.3	127.8	132.2	136.2
Grasping	**562.8**	2822.9	839.9	775.1	597.3
UAV Search	**83.6**	97.2	142.7	133.6	151.4

Table 3.3: Average total planning time, excluding the time for plan execution.

	Time (seconds)				
	RAId	IG	IG-Cost	IG-Cost-2	NAId-Replan
2-Star (d=10, n=5)	0.5	0.0	0.0	1.6	7.8
2-Star (d=10, n=6)	1.4	0.1	0.1	15.3	68.6
2-Star (d=53, n=6)	1.3	0.1	0.8	16.9	1045.4
2-Star (d=53, n=7)	5.2	0.4	5.6	3.3	684.2
2-Star (d=53, n=8)	22.6	1.4	3.4	4305.5	8415.7
Adaptive 2-Star	3.3	0.2	3.4	417.5	10290.6
Grasping	22.9	2.4	4.1	88.7	4523.5
UAV Search	25.5	0.5	2.5	157.4	16753.5

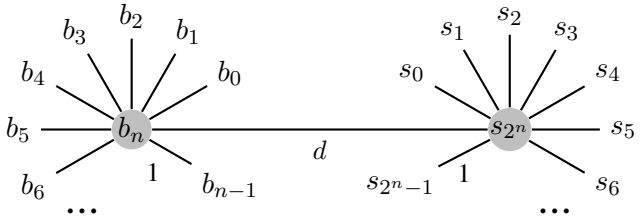

Figure 3.2: The 2-star graph.

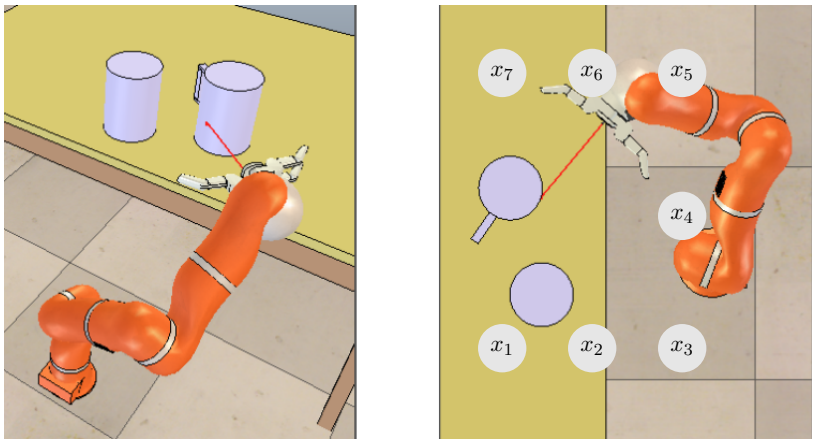

Figure 3.3: Grasp the cup with a handle. The figure shows the side view (left) and the top view (right) of the same robot configuration with the robot hand on the right side of the table.

occurring. Each hypothesis $h \in H$ is coded in its binary representation.

To identify the true hypothesis, the robot visits the nodes in a graph consisting of two connected stars (Figure 3.2). One star has center b_n and n peripheral nodes $b_0, b_2, \ldots, b_{n-1}$. The other star has center s_{2^n} and 2^n peripheral nodes $s_0, s_1, \ldots, s_{2^n-1}$. There is an edge connecting the two centers nodes, with edge-weight d. The weight of an edge between a center and a connected peripheral node is 1. The set X contains only the peripheral nodes and not the two centers, b_n and s_{2^n}, which serve only the purpose of connecting the peripheral nodes. The robot is initially located at s_{2^n}.

At each node b_i in X, the robot receives observation 1 if the ith bit of the true hypothesis h is 1, and receives 0 otherwise. At each node s_i in X, the robot receives observation 1 if $h = i$, and receives 0 otherwise. Clearly the b-nodes provide much more informative observations than the s-nodes. Visiting b-nodes is similar to binary search, while visiting s-nodes is similar to linear search. Since the robot starts at s_{2^n}, the main issue is to decide whether to pay the high cost of traversing the inter-star edge in order to benefit from the more informative observations at the b-nodes. Unfortunately, even in this very simple example, the issue cannot be resolved locally in a greedy manner.

In this experiment, the two nonmyopic algorithms, RAId and NAId-Replan, consistently obtain good policies (Table 3.2).

The greedy algorithms do not perform as well. Curiously IG sometimes outperforms IG-Cost. This is, however, coincidence. By completely ignoring the movement cost, IG

naturally moves to the more informative b-nodes. IG-Cost reasons about cost, but it is unable to decide optimally whether to jump to b-nodes or stay on s-nodes. In the two instances with $d = 10$, the optimal policy stays with the s-nodes when $n = 5$; it jumps to the b-nodes when $n = 6$. IG-Cost always moves to the b-nodes, simply because the movement cost is low. Hence, IG-Cost underperforms when $n = 5$. In the instances with $d = 53$, IG-Cost is again misled by the greedy local analysis and decides to stay at the s-nodes, because it is cheaper to reach them. This is optimal when $n = 6$, but the performance degrades quickly when $n = 7$ or 8. In fact, IG-Cost's regret, measured against the optimal policy, increases exponentially, as n grows.

Compared with the greedy algorithms, IG-Cost-2 has longer planning horizon. Although it takes more computational time, IG-Cost-2 fails to obtain better policies. It seems that a horizon of 2 is still insufficient for the tasks here.

It is somewhat surprising that the optimal policies for our 2-star graph instances are in fact nonadaptive. Intuitively the optimal policy would either (i) always stay on the s-nodes or (ii) jump to the b-nodes and stay there, depending on the d and n values, until the true hypothesis is identified. The traversal does not depend on the observations received, and adaptivity is not required. This is confirmed by examining the results computed by RAId. The nonadaptive optimal policies explain why RAId and NAId-Replan achieve comparable performance.

Adaptive 2-Star Graph

To better understand the issue of adaptivity, let us now modify the 2-star graph so that the optimal policy is adaptive. For $i = 0, 1, \ldots, n-1$, replace each peripheral node b_i in the 2-star graph by m copies, $b_{i,0}, b_{i,1}, \ldots b_{i,m-1}$, each connected to the center b_n by an edge of weight 1. For each i, only one of the m copies is *informative*. A function $g(h, i)$ specifies the index of the informative node for every $h \in H$ and $i \in [0, n-1]$. At an informative node $b_{i,j}$, the observation provides two values: the binary value of the ith bit of h and the index of the informative node for the next bit, $g(h, i+1)$. At an uninformative node, the observation provides no information. With this modification, an optimal policy must locate the informative b-nodes based on the observation received.

With suitable d and n values, an optimal policy visits $b_{0,0}, b_{0,1}, b_{0,2}, \ldots$ until reach-

ing the first informative node. It then uses the information from the received observation to move to the next informative node and so on. This is clearly an adaptive policy. A nonadaptive policy cannot change its behavior based on the observation received and is suboptimal.

This example is constructed, but not necessarily artificial. The basic idea is that each informative node contains a "map" that points to the next location of interest.

In the experiment, $d = 53$, $n = 7$, and $m = 5$. The function g is randomly generated, but remains fixed for all runs. RAId significantly outperforms all of IG-Cost, IG-Cost-2, and NAId-Replan. Although NAId-Replan achieves some level of adaptivity through replanning, it is inadequate.

Grasping a Cup

There are two cups on the table, one with a handle and one without. A robot arm needs to lift the cup with a handle by grasping on the handle (Figure 3.3). Using an external camera placed on the left side of the table, the robot can accurately sense the positions of the two cups. However, due to occlusion, it is uncertain which cup has a handle and where the handle is.

Each hypothesis (κ, θ) has two parameters: κ is a binary value that indicates which cup has a handle, and θ is the cup's orientation, which determines the handle location. The handle faces away from the external camera. So those hypotheses have higher prior probabilities.

The robot arm has a single-beam laser range finder mounted at its the wrist. The range finder reports the (discretized) distance to the nearest object in the direction that the range finder is facing.

We sample seven wrist positions $x_1, x_2, \ldots, x_7$ around the cups (Figure 3.3). At each position, the robot can pan the range finder in the plane parallel to the tabletop. Panning by a fixed amount incurs a cost of 4. Moving the wrist from one position to another incurs a higher cost: the distance between the current position and the target position, scaled up by a factor of 15. The robot arm starts at wrist position x_1 on the left side of the table.

RAId achieves the lowest cost in this experiments. Under RAId, the robot moves

progressively from x_1 to x_7 and pans the range finder at each position to take observations. This is a good strategy, because it avoids excessive robot arm movement, which incurs high cost.

IG performs very poorly, because it completely ignores the difference in action costs and moves the robot arm excessively between the various wrist positions in order to seek sometimes minor additional information gain. IG-Cost does not perform well either. Under IG-Cost, the robot moves to x_6 in the first step, because it expects to see the handle from there with high probability according to the prior. However, with small probability, the cup is oriented so that the handle is not visible from x_6. In this case, the robot must pay a high cost to travel back to the other positions. On the average, the aggressive move to x_6 does not pay off. This example clearly shows the weakness of greedy strategies, which do not plan *multiple steps* ahead.

IG-Cost-2 achieves lower cost than IG-Cost, because of its slightly longer planning horizon, but it is substantially worse than RAId.

NAId-Replan achieves comparable, but slightly worse result than RAId. NAId-Replan is nonmyopic. It is also adaptive, to a limited extent. We suspect that similar to the 2-star graph, adaptivity has limited benefit for this task, but there is no easy way to verify this.

UAV Search

This is the example described in Section 2.1. One may think that the optimal strategy is for the UAV to rise to the high altitude, search and locate the target in a 3×3 area, and finally descend to the low altitude in order to localize the target precisely. RAId, however, does not always do this, because the cost of descending is high. Figure 2.2 shows a sample run of RAId. After identifying the 3×3 area, the UAV stays at the high altitude. It moves around in the neighborhood and fuses the observations received to localize the target precisely without descending.

IG-Cost does not perform well, again because it does not plan multiple steps ahead. It fails to recognize that although the cost of climbing to the high altitude seems high in one step, the cost can be amortized over many future high-altitude observations, which are more informative. Under IG-Cost, the UAV always stays on the low altitude and

does not climb up. The result does not improve much even with 2-step lookahead in IG-Cost-2. Under IG-Cost-2, the UAV climbs up only occasionally in some instances. NAId-Replan does not perform well, either. Replanning does not provide sufficient adaptivity for this task.

3.6 Informative Path Planning with Noisy Observation

We have considered adaptive information path planning with noiseless observations, *i.e.*there is only one possible observation outcome given the true hypothesis at each location. This formulation is inadequate to model many real-world problems where there is uncertainty in sensor measurements. When we have noisy observations, there might be several possible observation outcomes. In this section, we give a simple extension to RAId to handle noisy observations.

RAId maintains a set of consistent hypotheses and recursively remove hypotheses that are inconsistent with new observation made when the robot moves to a new sensing location from this set. As a consequence of having noisy observations, a hypothesis may have multiple observation outcomes that are consistent with it. Sensing operations cannot be guaranteed to remove at least one hypothesis from the set of consistent hypothesis. Hence, we cannot apply RAId directly to IPP with noisy observations.

Instead of working with sets of consistent hypotheses, we need to work with beliefs that are probability distributions over the hypothesis space. After the robot move to a sensing location x and receives a new observation o, we can update the probabilities of hypotheses using Bayes rule:

$$b(h) \leftarrow \eta\, Z_x(h, o) b(h) \text{ for every } h \in H,$$

where η is a normalization constant.

There are works on IPP with noisy observations (Hollinger et al., 2012; Hollinger, Mitra, and Sukhatme, 2011; A. Singh, Krause, and Kaiser, 2009). However, as discussed in chapter 3, they are either nonadaptive or they do not provide any theoretical performance guarantee.

3.6.1 Sampling Deterministic Instance

We describe a simple extension, Sampled-*RAId*, to handle noisy observations. Our strategy is first to create a noiseless IPP problem $\mathcal{I}' = (X, d, H', \rho', O, \mathcal{Z}', r)$ from the original noisy one $\mathcal{I} = (X, d, H, \rho, O, \mathcal{Z}, r)$, by associating a hypothesis with observations. For noiseless observations, each hypothesis h has a unique observation vector $(o_1, o_2, \ldots, o_{|X|})$, where $Z_{x_i}(h, o_{x_i}) = 1$ for each location $x_i \in X$. This one-to-one relationship allows us to represent a hypothesis by its associated observation vector. The hypothesis space H is then simply a set of points in $O^{|X|}$. For noisy observations, the one-to-one relationship no longer holds, but the intuition of associating hypotheses with their observation vectors remains valid.

Formally we set $H' = O^{|X|}$. For a hypothesis $h' = (o_1, o_2, \ldots, o_{|X|})$ in H', the prior probability of h' is the probability of observing h' if the robot visits all locations in X: $\rho'(h') = \sum_{h \in H} \rho(h) \prod_{i=1}^{|X|} Z_{x_i}(h, o_i)$. Finally, the observation function $Z'_{x_i}(h', o) = 1$ if $o = o_i$.

Sampled-RAId applies RAId to $\mathcal{I}'$ with three changes:

- For computational efficiency, we sample a set of n hypotheses from H' in each recursive step of RAId and use it an approximate representation of H'.

- Although $\mathcal{I}$ is transformed into $\mathcal{I}'$, our goal is still to acquire information on the original hypothesis space H. We maintain a probability distribution over H. Initially, $b = \rho$. Because of noise, we cannot use an observation to *eliminate* a hypothesis $h \in H$, but we can update their probabilities using the Bayes rule. Suppose that the robot receives a new observation o at location x. We replace Algorithm 1, line 15 with

$$b(h) \leftarrow \eta \, Z_x(h, o) b(h) \text{ for every } h \in H,$$

  where η is a normalization constant.

- Finally, we terminate RAId if the most likely hypothesis $h^* = \arg\max_{h \in H} b(h)$ has probability greater than a given constant $\gamma \in (0, 1]$).

Sampled-RAId is shown in Algorithm 2.

Algorithm 2 *Sampled*-RAId

procedure *Sampled*-RAId($X, d, H, \rho, O, \mathcal{Z}, r$)
  $h^* = \arg\max_{h \in H} \rho(h)$
  if $\rho(h^*) \geq$ threshold **then**
    return h^*.
  else
    Create a new noiseless IPP problem $(X, d, H', \rho', O, \mathcal{Z}', r)$ by sampling n observation vectors from $O^{|X|}$
    $\nu \leftarrow \min\big(0.5, 1 - \max_{h \in H'} \rho'(h)\big)$.
    $\tau \leftarrow$ GROUPSTEINERTOUR(X, $X \times X$, d, $\{X_h\}_{h \in H'}$, ρ', ν), where $\tau = (x_0, x_1, \ldots, x_t)$ and $x_0 = x_t = r$.
    $(b, r) \leftarrow$ EXECUTEPLAN(τ, ρ, r). that $\sum_{h \in H} \rho(h) = 1$.
    RAId($X, d, H, b, O, \mathcal{Z}, r$)

procedure EXECUTEPLAN(τ, b, r)
  $i \leftarrow 1$.
  repeat
    $r \leftarrow x_i$.
    Visit location r and receive observation o.
    $b(h) \leftarrow \eta\, Z_x(h, o) b(h)$ for every $h \in H$, where η is a normalization constant.
    $i \leftarrow i + 1$.
  until $o \in \Omega_r$ or $i = t$.
  $r \leftarrow x_t$.
  Move to location r.
  return (b, r).

Table 3.4: The performance of *Sampled*-RAId on the UAV Search task with noisy observations. Noise level σ means that the high-altitude sensor reports a false observation with probability σ, and n is the number of samples.

Noise	**Cost**		
	$n = 128$	$n = 192$	$n = 320$
0.01	110.1	104.6	106.1
0.05	131.9	135.5	131.3

Under the assumption of noiseless observations, Noisy RAId reverts back RAId. In the first change, $H' = H$. We do not need to sample. In the second change, $Z_x(h, o)$ is either 1 or 0. Bayesian update is then equivalent to hypothesis elimination. In the third change, we set $\gamma = 1$.

We performed preliminary experiments to evaluate this idea on the UAV Search task (Section 3.5.2) with two different noise levels for the high-altitude sensor. The termination condition γ was set to 0.99. We evaluated multiple settings with different numbers of samples. For each setting, we run one trial for every hypothesis $h \in H$ and averaged performance statistics. The results, reported in Table 3.4 and Table 3.5, are

Table 3.5: The average total planning time of Noisy RAId on UAV Search with noisy observations.

Noise	Time (seconds)		
	$n = 128$	$n = 192$	$n = 320$
0.01	20.8	28.8	40.7
0.05	44.1	52.1	55.7

promising. Although the size of H' is 2^{128}, the algorithm identifies the true hypothesis correctly for every trial with only a few hundred samples in all settings. In other words, it always identifies the correct hypothesis according to the ground truth. In general, the robot's travel cost increases with noisy observations, as expected. With more samples, we expect the algorithm to compute a better policy with lower cost. However, the trend in the data is not definitive. Either a small number of samples is sufficient in this case to produce a near-optimal policy or a much larger number of samples is needed for significant improvement. Further investigation is required. We provide more detailed comparison between *Sampled*-RAId and other algorithms in Section 4.6.3.

While *Sampled*-RAId show promising results in preliminary evaluation, it is optimizing the wrong objective, and therefore it does not have performance guarantee. *Sampled*-RAId seeks to identify the sampled noisy observation vector instead of the true underlying hypothesis that generates it. The set of sensing locations that is good for differentiating those sampled observation vectors may not be not the same set that is good for differentiating the true hypothesis. Furthermore, we can only afford to sample and process a small set of observation vectors compared to the set of possible observation vectors, which could be exponential in number. Even though this problem can be partially mitigated by re-sampling them at each recursive step, it could be sub-optimal because the algorithm is designed to "overfit" to the noise from small amount of training data (sampled observation vectors). In the next chapter, we to provide an algorithm that achieves near-optimal performance using the same idea from RAId.

3.7 Conclusion

RAId is a new algorithm for the NP-hard informative path planning problem. We show that it computes a polylogarithmic approximation to the optimal solution in polynomial time, when the robot travels in a metric space. Furthermore, our experiments demon-

strate that RAId is effective in practice and provides good approximate solutions for several distinct robot planning tasks. We also extend RAId to handle noisy observation in *Sampled*-RAId. Unlike RAId, *Sampled*-RAId does not have any theoretical performance guarantee.

Chapter 4

Adaptive Stochastic Optimization

4.1 Introduction

Combinatorial optimization problems are hard in general. Fortunately, many real world problems have submodular and monotone objective functions which make them easy to approximate. Adaptive stochastic optimization extends deterministic combinatorial optimization to stochastic settings where we model the unknown part of the world as a random variable. Adaptive submodularity (Golovin and Krause, 2011) elegantly generalizes submodularity to stochastic settings for adaptive stochastic optimization problem on subsets but it is unclear if adaptive submodularity is sufficient for an efficient approximation algorithm to exist for the adaptive stochastic optimization problem on paths.

We propose a new condition, called the *marginal likelihood rate bound* condition for *pointwise submodular* functions, and propose an algorithm called *Recursive Adaptive Coverage* (RAC) to give near optimal solutions to adaptive stochastic optimization problems (on both subsets and paths) that optimize these functions. RAC extends RAId from Chapter 3 from identifying hypothesis to achieving general goals that can be modeled by functions. The marginal likelihood rate bound condition does not imply adaptive submodularity and vice versa. Even if we restrict our problems to adaptive stochastic optimization on subsets, the marginal likelihood rate bound enlarges the class of problems that can be efficiently approximated.

There are natural problems that do not satisfy marginal likelihood rate bound conditon and adaptive submodular. We propose a more general condition, the marginal

likelihood bound condition for these problems. In fact, all discrete finite pointwise submodular and monotone function can be made to satisfy marginal likelihood bound. We give a modified version of RAC that efficiently computes an approximation for all pointwise submodular function, where the quality of approximation depends on a problem-specific constant used to satisfy marginal likelihood bound. Figure 4.1 shows the relationship between properties of pointwise submodular functions.

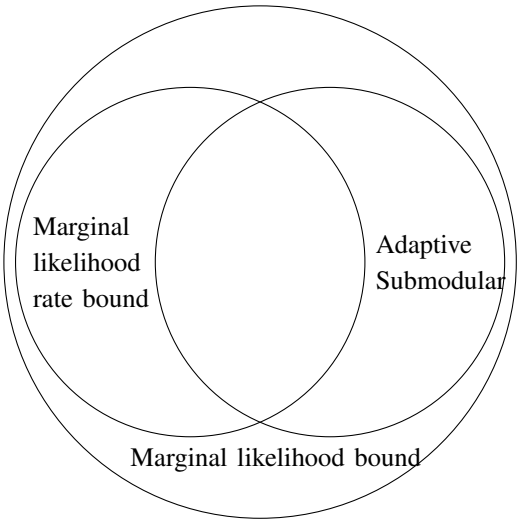

Figure 4.1: Relationship between properties of pointwise submodular functions

Informally, the marginal likelihood rate bound and the marginal likelihood bound conditions imply that the worst case objective value should increase when uncertainty decreases. Marginal likelihood rate bound is a stricter condition that restricts the worst case value to increase at the same "rate" as uncertainty decreases. We quantify uncertainty using the marginal likelihood of a history, which reflects the uncertainty over the unknown environment state. At the beginning when the history is empty, all scenarios are possible and likelihood of the empty history is 1. As we visit new locations and receive observations, the history grows and its marginal likelihood decreases. At the same time the scenarios that are inconsistent with the history becomes impossible. The sum of probabilities of scenarios that are still possible is equal to marginal likelihood of history. Hence, we reduce the possibilities of scenarios and thus uncertainty when we decrease the likelihood of history.

The previous chapter left the informative path planning problem with noisy observations open. We apply results from this chapter and obtain near-optimal solution to it.

RAC gives promising results on noisy variant of two IPP tasks from Chapter 3 when evaluated in simulation.

4.2 Related Work

Submodular set function optimization encompasses many hard combinatorial optimization problems and has many applications in operation research and decision making. A simple "greedy" heuristic was shown to give good approximation bound(Nemhauser, Wolsey, and Fisher, 1978) on submodular monotone set functions. Recent works have incorporate stochasticity to submodular optimization (Asadpour, Nazerzadeh, and Saberi, 2008; Golovin and Krause, 2011). Submodular optimization has been extend to submodular orienteering in (Calinescu and Zelikovsky, 2005).

Our work build on progress in submodular orienteering to solve the adaptive stochastic optimization problem. Our RAC algorithm has similar structure and analysis as RAId algorithm in Chapter 3 that is used to solve adaptive informative path planning (IPP) problems. Adaptive IPP is a special case of adaptive stochastic optimization problems on paths that satisfies the marginal likelihood rate bound condition. We can derive the same approximation bounds by apply the results in Section 3.4 directly. Both works are inspired by the recursive algorithm in (Gupta, Nagarajan, and Ravi, 2010) used to solve the Adaptive Traveling Salesperson (ATSP) problem. In the ATSP problem, a salesperson has to service a subset of locations with demand that is not known in advance. However, the salesperson knows the prior probabilities of the demand at each location (possibly correlated) and the goal is to find an adaptive policy to service all locations with demand.

Adaptive submodularity (Golovin and Krause, 2011) generalizes submodularity for set functions to adaptive policies for stochastic set functions. As in the classic approximation result for submodular functions (Nemhauser, Wolsey, and Fisher, 1978), a greedy heuristic was shown to give logarithmic approximation ratio with respect to the optimal adaptive policy. It was also shown that no polynomial time algorithm can compute approximate solution of adaptive stochastic optimization problems within a factor of $O(|X|^{1-\epsilon})$ unless $PH = \sum_2^p$, that is the polynomial-time hierarchy collapses to its second level (Golovin and Krause, 2011). Adaptive submodularity can be seen

as a way to circumvent the hardness. Many Bayesian active learning problems can be modeled by suitable adaptive submodular objective functions (Golovin, Krause, and Ray, 2010; Cuong, Lee, Ye, et al., 2013; Cuong, Lee, and Ye, 2014). However, (Cuong, Lee, and Ye, 2014) recently proposed a new objective function for active learning with a general loss function that is not adaptive monotone submodular. This new objective function satisfies the marginal likelihood bound condition with nontrivial constant G. Section 4.3 describes marginal likelihood bound and marginal likelihood rate bound formally and gives the relationship between these conditions.

Asadpour, Nazerzadeh, and Saberi (2008) considers a notion of stochastic set functions where the outcome of selecting each element is independent of others and shows that the adaptive gap, the ratio between optimal adaptive policy and optimal non-adaptive policy is at most $\frac{e}{e-1}$. An important difference from our work is we allow the outcome of selecting elements to be correlated. Correlations in outcome is common in many practical applications such as Bayesian active learning and informative path planning.

Interactive submodular cover (Guillory and Bilmes, 2010; Guillory and Bilmes, 2011) considers the case where one must cover an unknown submodular function from a family of submodular functions. Instead of receiving random observations from a posterior probability distribution, the observations are chosen adversarially from a set of valid observations. They prove logarithmic approximation guarantees for worst-case policy cost. In contrast, we prove results for average-case policy cost in this work.

4.3 Classes of adaptive stochastic optimization

We now examine the classes of functions covered in this chapter. We restrict ourselves to pointwise submodular and monotone functions on a finite domain. We introduce a condition called the marginal likelihood rate bound which allows efficient approximation algorithms for the adaptive stochastic optimization problems. We show that there are functions that satisfy the marginal likelihood rate bound condition but does not satisfy the previously studied adaptive submodularity condition (Golovin, Krause, and Ray, 2010), and vice versa. We also introduce a condition called the marginal likelihood bound which is satisfied by all pointwise submodular and monotone stochastic functions on a finite domain, albeit with different bounding constants. An efficient adap-

tive stochastic optimization problem algorithm for these classes of function is given in Section 4.4.

4.3.1 Adaptive Monotonicity and Submodularity

Adaptive submodularity and monotonicity generalize submodularity and monotonicity to stochastic settings where we receive random observations at each item. We define the expected marginal value of an item x given a history ψ, $\triangle(x|\psi)$ as:

$$\triangle(x|\psi) = \mathrm{E}_{\phi\sim\psi}\left[f(\mathrm{dom}(\psi)\cup\{x\},\phi) - f(\mathrm{dom}(\psi),\phi)\right]$$

A function $f : 2^X \times O^X \rightarrow \mathbb{R}$ is *adaptivity monotone* with respect to a prior distribution $p(\phi)$ if , for all ψ such that $p[\Phi \sim \psi] > 0$ and all $x \in X$, it holds that

$$\triangle(x|\psi) \geq 0$$

i.e. the expected marginal value of any fixed item is nonnegative.

Adaptive submodular and monotonicity is sufficient for a greedy policy to be near-optimal on the adaptive stochastic maximization problem that maximizes the value of a stochastic function given a budget. This is a dual form of of the adaptive stochastic optimization problem where we minimize the cost for covering the function. Strong adaptive monotonicity, a stricter condition than adaptive monotonicity, is required for greedy selection to be near-optimal for adaptive stochastic optimization problems. A function f is strongly adaptive monotone if for any item x and observation o such that $p(o|x,\psi) > 0$,

$$\mathrm{E}_{\phi\sim\psi}\left[f(\mathrm{dom}(\psi),\phi)\right] \leq \mathrm{E}_{\phi\sim\psi,\phi(x)=o}[f(\mathrm{dom}(\psi)\cup\{x\},\phi)]$$

Strong adaptive monotonicity means that the expected marginal value of selecting any fixed item and receiving an observation is nonnegative.

A function $f : 2^X \times O^X \rightarrow \mathbb{R}$ is *adaptive submodular* with respect to a prior distribution $p(\phi)$ if, for all ψ and ψ' such that $\psi' \sim \psi$ and for all $x \in X \setminus \mathrm{dom}(\psi')$, it

holds that

$$\triangle(x|\psi) \geq \triangle(x|\psi')$$

i.e. the expected marginal value of any fixed item does not increase as more items are visited and observations received.

4.3.2 Marginal likelihood rate bound

We denote $\hat{f}(S, \psi) = \min_{\phi \sim \psi} f(S, \phi)$ as the worst case value of f given a history. The marginal likelihood rate bound condition requires a function f such that: For all $\psi' \sim \psi$, if $p(\psi') \leq 0.5p(\psi)$ then

$$Q - \hat{f}(\text{dom}(\psi'), \psi') \leq \frac{1}{K}\left(Q - \hat{f}(\text{dom}(\psi), \psi)\right) \tag{4.1}$$

except for scenarios already covered, where $K > 1$ and $Q \geq \max_{\phi} f(X, \phi)$ is a constant upper bound for the maximum value of f for all scenarios.

Intuitively, this condition means that the worst case remaining objective value decreases by a constant fraction whenever the marginal likelihood of history decreases by more than half.

Example The version space reduction function $\mathcal{V}$ with arbitrary prior is adaptive submodular and monotone (Golovin and Krause, 2011) satisfies marginal likelihood rate bound. The version space reduction function $\mathcal{V}$ is defined as:

$$\mathcal{V}(S, \phi) = 1 - \sum_{\phi' \sim \phi(S)} \rho(\phi') \tag{4.2}$$

for all scenario ϕ, $S \subseteq X$ and $\phi(S)$ gives the history of visiting locations x in S when the scenario is ϕ. We present the proof of satisfying the marginal likelihood rate bound in the Appendix.

Proposition 1 (Version Space Reduction)**.** *The version space function $\mathcal{V}$ satisfies marginal likelihood rate bound.*

RAId in Chapter 3 is a special case of RAC applied to the function $\mathcal{V}$ and prior ρ_H. Each hypothesis $h \in H$ in the IPP problem has a corresponding scenario ϕ_h that is the

observation vector $\phi_h = (o_1, o_2, \ldots, o_{|X|})$, where $Z_{x_i}(h, o_{x_i}) = 1$. The prior ρ_H has the support set of $\{\phi_h\}$ for all hypothesis $h \in H$ and $\rho_H(\phi_h) = \rho(h)$.

To tease apart the relationship between condition marginal likelihood rate bound and adaptive monotone submodular problems, we construct a few examples to show the following relationship:

Proposition 2. *Adaptive monotonicity and submodularity does not imply the marginal likelihood rate bound. Furthermore, the marginal likelihood rate bound does not imply adaptive monotonicity and submodularity.*

4.3.3 Marginal likelihood bound

The marginal likelihood bound condition requires that for some constant G,

$$f(X, \phi) - \hat{f}(\text{dom}(\psi), \psi) \leq G \cdot p(\psi), \tag{4.3}$$

for all scenarios $\phi \sim \psi$. In other words, the worst remaining objective value must be less than the marginal likelihood of its history multiplied by some constant G. Our quality of solution depends on the constant G. The smaller the constant G, the better the approximation bound.

In fact, we can make any adaptive stochastic optimization problem satisfy the marginal likelihood bound with a large enough constant G. To trivially ensure the bound of marginal likelihood bound, let $Q = \max_\phi f(X, \phi)$, we set $G = Q \cdot 1/\delta$, where $\delta = \min_\phi \rho(\phi)$. Hence, $Q \leq G \cdot p(\psi)$ unless we have visited all locations and covered the function by definition.

Example The version space reduction function $\mathcal{V}$ can be interpreted as the expected $0-1$ loss of a random scenario $\phi' \sim \psi$ differing from true scenario ϕ. The loss is counted as one whenever $\phi' \neq \phi$. For example in the UAV task, a pair of scenarios that differ in only one sensor has the same loss of 1 as another pair that differs in all sensor readings. Thus, it can be useful to assign different loss to different pair of scenarios with a general loss function. Correspondingly, the generalized version space reduction

function is defined as:

$$f_L(S, \phi) = \mathrm{E}_{\phi'} \left[L(\phi, \phi') \mathbf{1}(\phi(S) \neq \phi'(S)) \right]. \tag{4.4}$$

where $L : O^X \times O^X \to \mathbb{R}_{\geq 0}$ is a general loss function that satisfies $L(\phi', \phi) = L(\phi, \phi')$ and $L(\phi, \phi') = 0$ if $\phi = \phi'$. (Cuong, Lee, and Ye, 2014) have shown that the generalized version space reduction function's average case criterion is not adaptive submodular with respect to a prior p_o such that $p_o(h) > 0$ for all h. However, the generalized version space reduction function satisfies marginal likelihood bound with a non-trivial constant $G = \max_{\phi,\phi'} L(\phi, \phi')$. On the other hand, it does not satisfy marginal likelihood rate bound.

Proposition 3. *The generalized version space reduction function f_L satisfies marginal likelihood bound with constant $G = \max_{\phi,\phi'} L(\phi, \phi')$.*

4.4 Algorithm

Adaptive planning is computationally hard due to the need to consider every possible observation after each action. RAC assumes that it always receive the most likely observation to simplify adaptive planning. RAC is a recursive algorithm that partially covers the function in each step and repeats on the residual function until the entire function is covered.

In each recursive step, RAC uses the mostly like observation assumption to transform adaptive stochastic optimization problem into a submodular orienteering problem to generate a tour and traverse it. If the assumption is true throughout the tour, then RAC achieves the required partial coverage. Otherwise, RAC receives some observation that has probability less than half (since only the most likely observation has probability at least half), the marginal likelihood of history decreases by at least half, and the marginal likelihood rate bound and marginal likelihood bound conditions ensures that substantial progress is made towards covering the function.

Submodular orienteering takes a submodular function $g : X \to \mathbb{R}$ and a metric on X and gives the minimum cost path τ that covers function g such that $g(\tau) = g(X)$. We now describe the submodular orienteering problem used in each recursive step.

Given the current history ψ, we construct a restricted set of location-observation pairs, $Z = \{(x, o) : (x, o) \notin \psi, o \text{ is the most likely observation at } x \text{ given } \psi\}$. Using ideas from (Guillory and Bilmes, 2010), we construct a submodular function $g^*_\nu : 2^Z \to \mathbb{R}$ to be used in the submodular orienteering problem. Upon completion of the recursive step, we would like the function to be either covered or have value at least ν for all scenarios consistent with $\psi \cup Z'$ where Z' is the selected subset of Z. We first restrict ϕ to a subset of scenarios Ψ that are consistent with ψ. To simplify, we transform the function so that its maximum value for all ϕ is at least ν by defining $f_\nu(S, \phi) = f(S, \phi) + (\nu - f(X, \phi))$ whenever $f(X, \phi) < \nu$ and $f_\nu(S, \phi) = f(S, \phi)$ otherwise. For $Z' \subseteq Z$, we now define $g_\nu(Z', \phi) = f_\nu(\text{dom}(\psi \cup Z'), \phi)$ if Z' is consistent with ϕ and $g_\nu(Z', \phi) = f_\nu(X, \phi)$ otherwise. Finally, we construct the submodular function $g^*_\nu(Z') = 1/|\Psi| \sum_{\phi \in \Psi} \min(\nu, g_\nu(Z', \phi))$. The constructions have the following properties that guarantees the effectiveness of the recursive steps of RAC.

Proposition 4. *Let f be a pointwise monotone submodular function. Then g_ν is pointwise monotone submodular and g^*_ν is monotone submodular. In addition $g^*_\nu(Z') \geq \nu$ if and only if f is either covered or have value at least ν for all scenarios consistent with $\psi \cup Z'$.*

We can replace g^*_ν by a simpler function if f satisfy a *minimal dependency* property where the value of function f depends only on the history, *i.e.* $f(\text{dom}(\psi), \phi') = f(\text{dom}(\psi), \phi)$ for all $\phi, \phi' \sim \psi$. We define a new submodular set function $g^m_\nu(Z') = g_\nu(Z', Z)$.

Proposition 5. *When f satisfies minimal dependency, $g^m_\nu(Z') \geq \nu$ implies $g^*_\nu(Z') \geq \nu$.*

RAC needs to guard against committing to costly plan made under the most likely observation assumption which is bound to be wrong eventually. RAC uses two different mechanisms for hedging. For marginal likelihood rate bound, instead of requiring complete coverage, we solve partial coverage using a submodular path optimization problem $g^*_{(1-1/K)Q}$ so that $f(S) \geq (1 - 1/K)Q$ for all consistent scenarios under the most likely observation assumption in each recursive step. For marginal likelihood bound, we solve submodular orienteering for complete coverage of g^*_Q but also solve for the version space reduction function with 0.5 as the target, $\mathcal{V}^*_{0.5}$, as a hedge against

over-commitment by the first tour when the function is not well aligned with the probability of observations. The cheaper tour is then traversed by RAC in each recursive step.

We define the informative observation set Ω_x for every location $x \in X$: $\Omega_x = \{ o \mid p(o|x) \leq 0.5\}$. RAC traverses the tour and adaptively terminates when it encounters an informative observation. Subsequent recursive calls work on the residual function f' and normalized prior p'. Let ψ be the history encountered so far just before the recursive call, for any set $S \supset \text{dom}(\psi)$ $f'(S, \phi) = f(S, \phi) - f(\text{dom}(\psi), \phi)$. Let η be any value such that $f(S, \phi) > Q - \eta$ implies $f(S, \phi) = f(X, \phi)$ for all $S \subseteq X$ and all scenario ϕ. The recursive step is repeated until the residual value Q' is less than η. We give the pseudocode of RAC in Algorithm 5. We give details of SUBMODULARORIENTEER procedure and prove its approximation bound in Appendix.

Algorithm 3 GenerateTour1

GenerateTour1 construct a tour for a function satisfying marginal likelihood bound

procedure GenerateTour1($X, d, \rho, O, r, f, Q, \eta$)
  $\tau_f \leftarrow$ SUBMODULARORIENTEER(X, $X \times X$, d, g^*_Q, ρ)
  if $\max_\phi p(\phi) \leq 0.5$ **then**
    $\tau_{vs} \leftarrow$ SUBMODULARORIENTEER(X, $X \times X$, d, $\mathcal{V}^*_{0.5}, \rho$)
    $\tau \leftarrow \arg\min_{\tau_f, \tau_{vs}}(W(\tau'))$
  else
    $\tau \leftarrow \tau_f$
  return τ where $\tau = (x_0, x_1, \ldots, x_t)$ and $x_0 = x_t = r$

Algorithm 4 GenerateTour2

Generate a tour for a function satisfying marginal likelihood rate bound

procedure GenerateTour2($X, d, \rho, O, r, f, Q, \eta$)
  $\tau \leftarrow$ SUBMODULARORIENTEER(X, $X \times X$, d, $g^*_{(1-1/K)Q}, \rho$)
  return τ where $\tau = (x_0, x_1, \ldots, x_t)$ and $x_0 = x_t = r$

4.5 Analysis

We first give the performance guarantees of RAC for adaptive stochastic optimization problem on paths and then specialize these results for adaptive stochastic optimization problem on subsets. There proofs are provided in Appendix.

Theorem 11. *Assume that f is a pointwise integer-valued submodular monotone function. Let η be any value such that $f(S, \phi) > f(X, \phi) - \eta$ implies $f(S, \phi) = f(X, \phi)$*

Algorithm 5 RAC

procedure recurseRAC($X, d, \rho, O, r, f, Q, \eta$)
  if $\max_{\phi \in \{\phi' | \rho(\phi') > 0\}} f(X, \phi) < \eta$ **then**
    return
  else
    $\tau \leftarrow$ GENERATETOUR($X, d, \rho, O, r, f, Q, \eta$) ▷ *Use an appropriate* GENERATETOUR1 *or* GENERATETOUR2 *procedure for the condition it satisfies*
    $(\psi, r) \leftarrow$ EXECUTEPLAN(τ, r)
    $\rho' \leftarrow \frac{p(\psi|\phi)p(\phi)}{p(\psi)}$, $f' \leftarrow f(Y, \phi) - f(\tau, \phi)$, $Q' \leftarrow Q - \min_\phi f(\tau, \phi)$ for all $\psi \sim \phi$
    recurseRAC($X, d, \rho', O, r, f', Q', \eta$)

procedure EXECUTEPLAN(τ)
  $i \leftarrow 1, \psi \leftarrow \{\}$
  repeat
    Visit location x_i and receive observation o.
    $\psi \leftarrow \psi \cup (x_i, o)$, $i \leftarrow i + 1$.
  until $o \in \Omega_{x_i}$ or $i = t$.
  Move to location $x_t = r$.
  return (ψ, r).

for all $S \subseteq X$ and all scenario ϕ. For any constant $\epsilon > 0$ and an instance of adaptive stochastic optimization problem on path satisfying marginal likelihood rate bound, RAC computes a policy π in polynomial time such that

$$C(\pi) = O((\log|X|)^{2+\epsilon} \log Q \log_K(Q/\eta))C(\pi^*)),$$

where Q and $K > 1$ are constants that satisfies Equation (4.1)*.*

Theorem 12. *Assume that the prior probability distribution ρ is represented as non-negative integers with $\sum_\phi \rho(\phi) = P$. Let η be any value such that $f(S, \phi) > f(X, \phi) - \eta$ implies $f(S, \phi) = f(X, \phi)$ for all $S \subseteq X$ and all scenario ϕ. Assume that f is a pointwise integer-valued submodular monotone function. For any constant $\epsilon > 0$ and an instance of adaptive stochastic optimization problem on path satisfying marginal likelihood bound, RAC computes a policy π for in polynomial time such that*

$$C(\pi) = O((\log|X|)^{2+\epsilon}(\log P + \log Q) \log(G/\eta))C(\pi^*),$$

where $Q = \max_\phi f(X, \phi)$.

For adaptive stochastic optimization problems on subsets, we achieve tighter ap-

proximation bounds by replacing the approximation bounds of submodular orienteering with greedy approximation of submodular set cover.

Theorem 13. *For an instance of adaptive stochastic optimization problem on subsets satisfying marginal likelihood rate bound, assuming f is pointwise integer-valued submodular and monotone, let η be any value such that $f(S,\phi) > f(X,\phi) - \eta$ implies $f(S,\phi) = f(X,\phi)$ for all $S \subseteq X$ and all scenario ϕ. RAC computes a policy π in polynomial time such that*

$$C(\pi) = 4(\ln Q + 1)(\log_K(Q/\eta) + 1)C(\pi^*),$$

where Q and $K > 1$ are constants that satisfies Equation (4.1)*.*

Theorem 14. *For an instance of adaptive stochastic optimization problem on subsets satisfying the marginal likelihood bound condition, assuming f is pointwise integer-valued submodular and monotone, let η be any value such that $f(S,\phi) > f(X,\phi) - \eta$ implies $f(S,\phi) = f(X,\phi)$ for all $S \subseteq X$ and all scenario ϕ and $\delta = \min_\phi \rho(\phi)$. RAC computes a policy π in polynomial time such that*

$$C(\pi) = 4(\ln 1/\delta + \ln Q + 2)(\log(G/\eta) + 1)C(\pi^*)),$$

where $Q = \max_\phi f(X,\phi)$.

4.6 Application: Noisy IPP

In this section, we apply RAC to solve IPP with noisy observations. First, we reduce an adaptive noisy IPP problem to an *Equivalence Class Determination* (ECD) problem. Then we apply RAC to solve ECD problem near-optimally using an objective function that satisfies marginal likelihood rate bound condition. Finally, we evaluate our approach on two IPP tasks with noisy observations.

4.6.1 Equivalence Class Determination Problem

An ECD problem consists of a set of hypotheses H that is partitioned into a set of equivalence classes $\{\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_m\}$. The goal of ECD problem is to identify which

equivalence class the true hypothesis lies in by moving to locations and making observations while minimizing the expected robot movement cost. ECD problem has been applied to noisy Bayesian active learning to achieve near-optimal performance (Golovin, Krause, and Ray, 2010). Noisy adaptive IPP problem can also be reduced to an ECD instance when it is always possible to identify the true hypothesis in the IPP problem.

The ECD problem may seems to be an easier problem compared to noiseless IPP as we do not need to determine the true hypothesis. Since we already solve noiseless IPP, a straightforward extension will be to apply the same algorithm on the ECD instance and terminate when all consistent hypotheses fall in the same equivalence class. However, this is not optimal. Consider an instance of ECD that has n hypotheses $h_1, h_2, \ldots, h_n$ with uniform prior. There are two equivalence classes, $\mathcal{H}_1 = \{h_1, \ldots h_{n-1}\}$ and $\mathcal{H}_2 = \{h_n\}$. There are n locations $x_1, x_2, \ldots x_n$ that are directly connected to the root r with unit cost. Each location x_i gives an observation 1 if h_i is the true hypothesis and 0 otherwise. The optimal policy only needs to visit location x_n but RAId will visit locations $x_1, x_2, \ldots, x_n$ in turn, resulting in an expected cost n times the optimal one.

To differentiate between the equivalence classes, we use the Gibbs error objective function (same as edge-cutting function in (Golovin, Krause, and Ray, 2010)). The idea is to consider the ambiguities between pairs of hypotheses in different equivalence class es, and to visit locations and make observations to disambiguate between them. The set of pairs of hypotheses in different classes is $\mathcal{E} = \cup_{1 \leq i < j \leq m} \{\{h', h''\} : h' \in \mathcal{H}_i, h'' \in \mathcal{H}_j\}$. We disambiguate a pair $\{h', h''\}$ when we make an observation o at a location x and either h' or h'' is inconsistent with the observation, $Z'_x(h', o) = 0$ or $Z'_x(h', o) = 0$. The set of pairs disambiguated by visiting a location x when hypothesis $h \in H'$ is true is given by $\mathcal{E}_x(h) = \{\{h', h''\} : Z'_x(h, o) = 1, Z'_x(h', o) = 0 \text{ or } Z'_x(h'', o) = 0\}$. We define a weight function $w : \mathcal{E} \to \mathbb{R}_{\geq 0}$ as $w(\{h', h''\}) = p'(h') \cdot p'(h'')$. We can now define the Gibbs error objective function:

$$f_{GE}(Y, h) = W(\cup_{x \in Y} \mathcal{E}_x(h)), \tag{4.5}$$

where $W(\mathcal{E}') = \sum_{e \in \mathcal{E}'} w(e)$, Y is the set of location visited and $h \in H'$.

Proposition 6. *The Gibbs error function f_{GE} is pointwise submodular and monotone. In addition, it satisfies condition marginal likelihood rate bound with constants $Q =$*

$W(\mathcal{E}) = 1 - \sum_{i=1}^{m}(p(\mathcal{H}_i))^2$, *the total weight of ambiguous pairs of hypotheses, and* $K = 2$.

4.6.2 Adaptive IPP with Noisy Observations

The first step to reduce adaptive noisy IPP instance $\mathcal{I}$ to ECD instance E is to create a noiseless IPP problem $\mathcal{I}' = (X, d, H', \rho', O, \mathcal{Z}', r)$ from a noisy IPP instance $\mathcal{I} = (X, d, H, \rho, O, \mathcal{Z}, r)$ using the transformation in section 3.6.1. Each hypothesis $h' \in H'$ is an observation vector $h' = (o_1, o_2, \ldots, o_{|X|})$ and the new hypothesis space H' is $H' = O^{|X|}$. For each hypothesis $h_i \in H$, we create an equivalence class $\mathcal{H}_i$ that consists of all hypotheses $h' = (o_1, o_2, \ldots, o_{|X|}) \in H'$ where the observation vector is consistent with $h_i \in H$, such that

$$\mathcal{H}_i = \left\{ (o_1, o_2, \ldots, o_{|X|}) \middle| \prod_{j=1}^{|X|} Z_{x_j}(h_i, o_j) > 0 \right\}.$$

When we can always identify the true underlying hypothesis $h \in H$, the equivalence classes is a partition on the set H', *i.e.* each observation vector associated with only one hypothesis $h \in H$. After we created the ECD instance E, we apply RAC to the corresponding Gibbs error function (eq. (4.5)). Hence, we get a near-optimal algorithm for noisy adaptive IPP problems when we apply RAC since the Gibbs error function f_{GE} satisfies marginal likelihood rate bound and RAC is near-optimal for such instances.

4.6.3 Experiment

IPP tasks

We evaluate RAC in simulation on variants of UAV search and rescue task and grasping task (see Sections 3.5.2 and 3.5.2) where we introduce noisy observations and equivalence classes to them.

In the UAV search and rescue task, we introduce noise to the high altitude sensor such that it may report a false observation with probability σ. There is a safe zone on the map where the survivor is deemed to be safe. We only need to know the exact location of the survivor if he is not in the safe zone. The equivalence classes in this task are the

locations outside of safe zone and the safe zone. The safe zone in this experiment is a 3 by 8 region on the right side of the map.

For grasping a cup task, we add noise to laser readings in the grasping task. Let x be the true distance and x' be the value reported by the laser, the probability distribution of the readings is: $p(x = x') = 0.85, p(|x-x'| = 1) = 0.05$, and $p(|x-x'| = 2) = 0.025$. Given that we identified the cup that has the handle, the robot gripper is fairly robust to estimation error of the cup handle's orientation. For each cup, we partition the cup handle orientation into regions of 20 degrees each. We only need to know the region that contains cup handle. The equivalence classes here are the regions.

After introducing noise to the task, it is still possible to for the UAV task to exactly find the location of survivor because the low altitude sensor is still noiseless. Adaptive IPP on UAV task can be reduced an ECD problem as the set of equivalence classes is a partition of the observation vectors. On the other hand, it is not always possible to identify the true region that contains the cup handle due to observation noise. However, we can still reduce to ECD problem by associating each observation vector to its most likely equivalence class.

Setup

We evaluate IG and IG-Cost, *Sampled*-RAId, and RAC with version space reduction (RAC-VSR) and Gibbs error (RAC-GE) objectives. Of these algorithms, only RAC-GE has theoretical performance guarantees for the noisy adaptive IPP problem. Even though RAC-VSR is guaranteed to perform near-optimally for the version space reduction function, it is the wrong objective for the adaptive IPP problem.

We set the termination condition of RAC to be $\eta = 10^{-5}$. The Gibbs error objective function corresponds to the exponentiated Rényi entropy (order 2) and can be interpreted as the prediction error of a Gibbs classifier. A Gibbs classifier predicts by sampling a hypothesis from the prior. For consistency, we set the other algorithms to terminate when Gibbs error of the prior is less than 10^{-5}. We run 1000 trials with the true hypothesis sampled randomly from the prior for the UAV search task and 3000 trials for the grasping task as its variance is higher. For *Sampled*-RAId, we set the number of samples to be three times the number of hypothesis. Out of the six algorithms we

compare, only RAC-GE is near-optimal for an ECD problem.

Results

To compare the performance between the algorithms, we pick a threshold γ for Gibbs error of the equivalence classes and compare the average cost incurred by each algorithm to reduce Gibbs error to below γ. We repeat this for 15 different γ, starting from $1 \times 10^{-}5$ and doubling the value each time. We plot the average cost with 95% confidence interval for UAV Search and Rescue task and the grasping task in Figures 4.5 and 4.6 respectively. For the grasping task, there may be trials where the minimum Gibbs error possible is greater than γ especially for small threshold γ due to noisy observations. We omit these trials when we compute the average cost incurred to reach γ, since the threshold γ is unreachable for these trials.

RAC-GE has the lowest average cost for both tasks at almost every threshold of Gibbs error. RAC-VSR has the second lowest average for the UAV search task but its confidence interval is outside of RAC-GE. On the other hand, RAC-GE has average cost much lower than RAC-$\mathcal{V}$ for every threshold level. The other algorithms, *Sampled*-RAId, IG-Cost and IG do not perform as well for both the UAV search and grasping task.

We plot the graphs again using Shannon's entropy of the partitions as threshold in Figures 4.2 and 4.3. We pick 10 threshold levels γ starting from 0.001 and doubling it every step. The experiment results reflect that of Gibbs error. RAC-GE achieves the best result for both UAV search and grasping task. It is worth noting that RAC-GE does not explicitly use Shannon's entropy as an optimization criteria, yet it perform the best in terms of Shannon's entropy.

4.7 Conclusion

We study approximation algorithms for adaptive stochastic optimization problem on paths and subsets. We give two conditions on pointwise monotone submodular functions that are useful for understanding the performance of approximation algorithms on these problems: the marginal likelihood bound condition and the marginal likelihood rate bound condition. Our algorithm, RAC, runs in polynomial time with an approx-

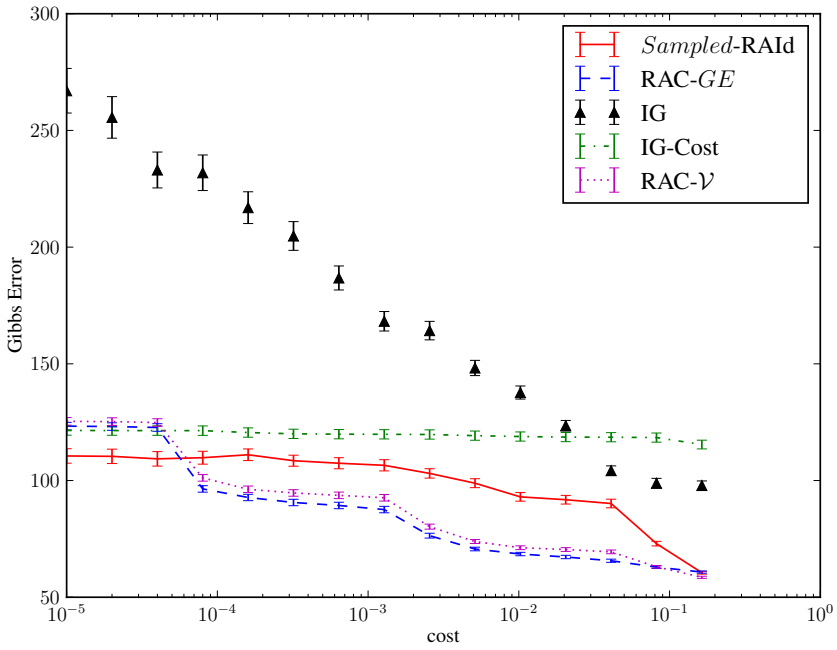

Figure 4.2: UAV Search and Rescue: Average cost vs Gibbs error

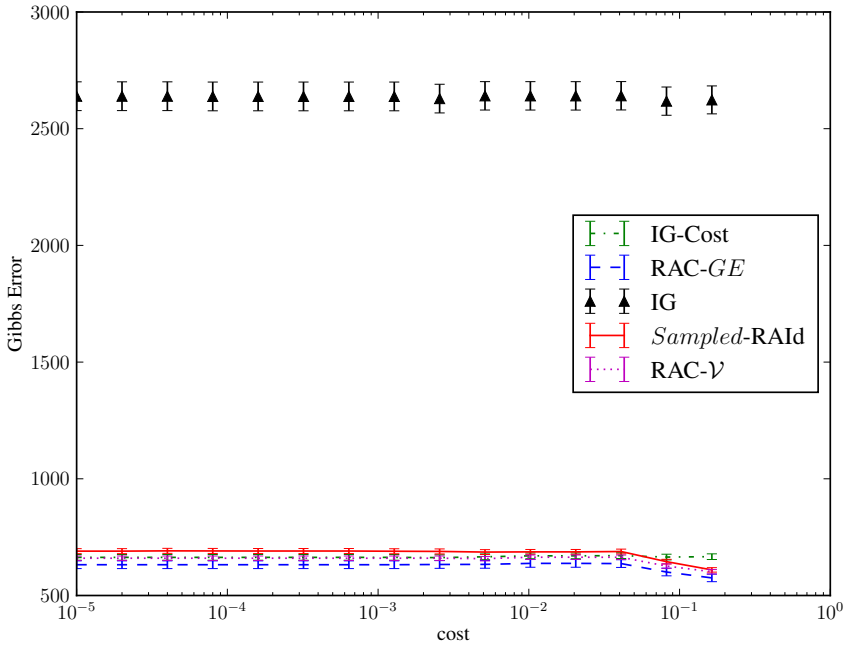

Figure 4.3: Grasping: Average cost vs Gibbs error. We zoom in on the top two algorithms in Figure 4.4

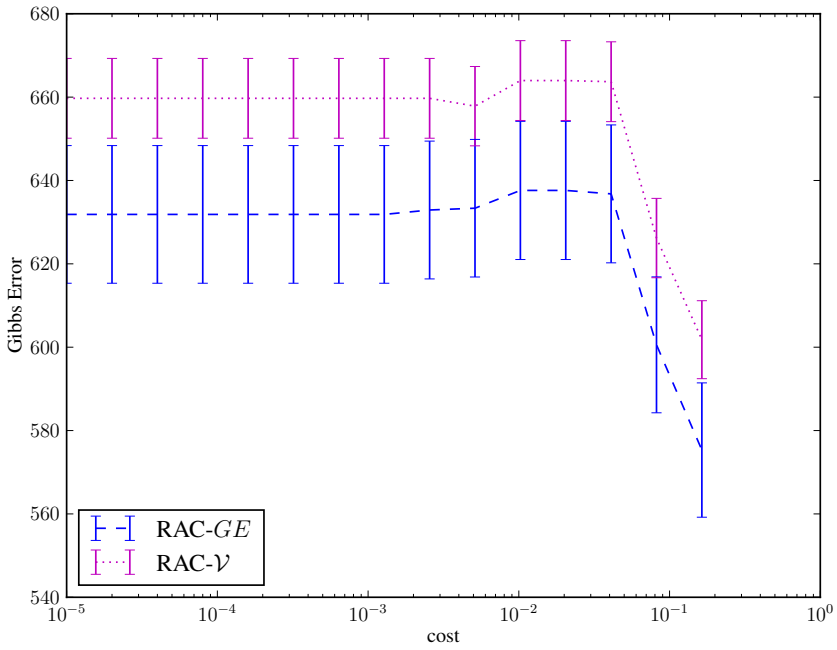

Figure 4.4: Grasping: Average cost vs Gibbs error for RAC-$\mathcal{V}$ and RAC-GE

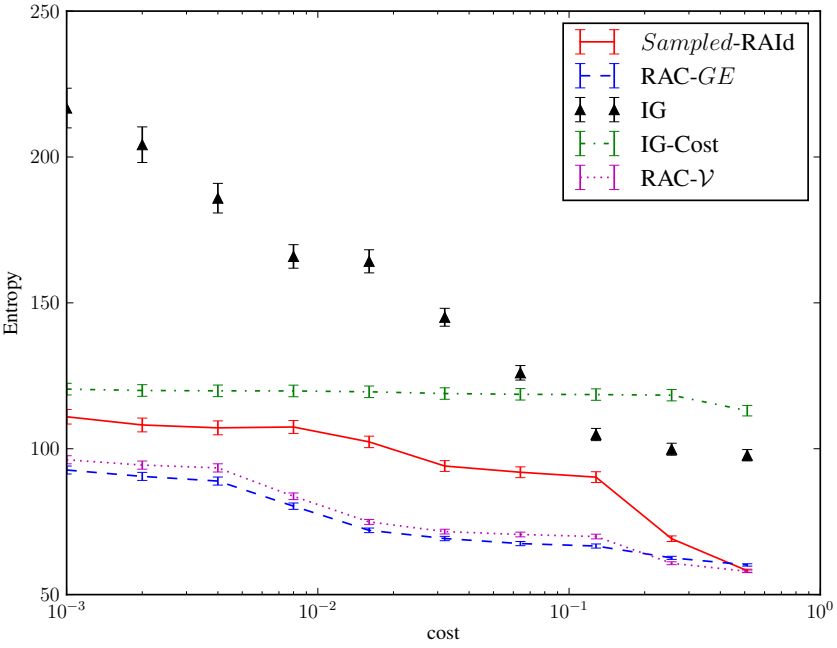

Figure 4.5: UAV Search and Rescue: Average cost vs Shannon's entropy

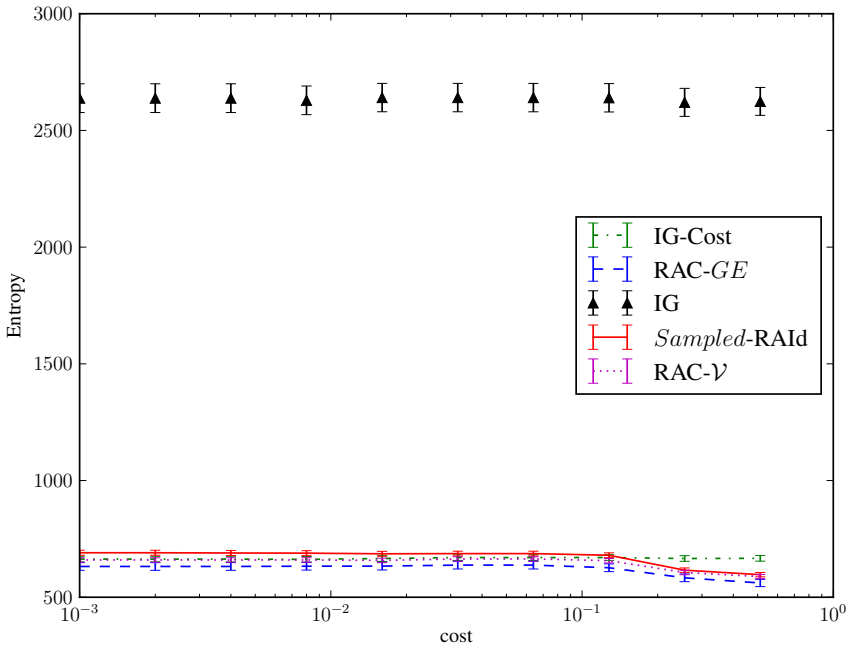

Figure 4.6: Grasping: Average cost vs Shannon's entropy. We zoom in on the top two algorithms in Figure 4.7

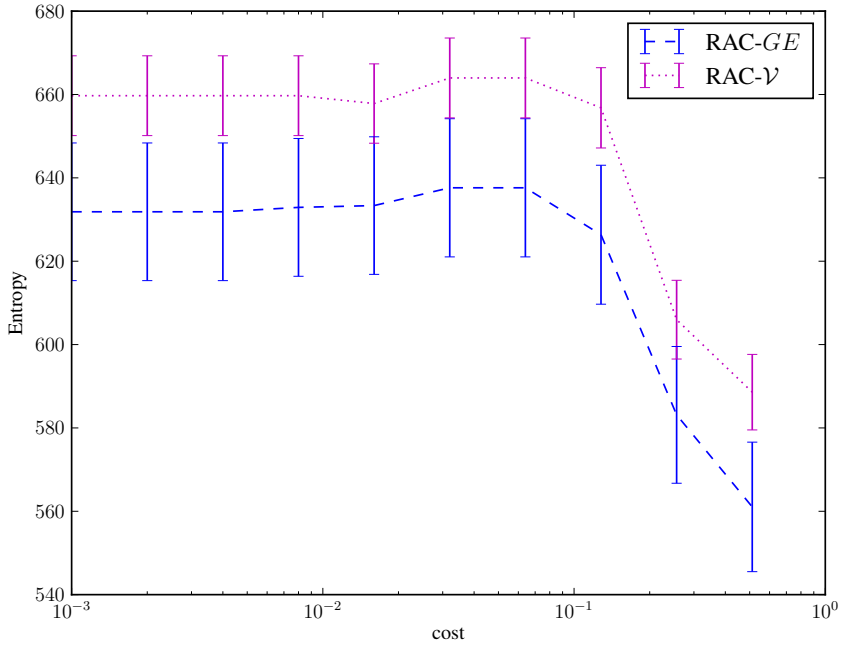

Figure 4.7: Grasping: Average cost vs Shannon's entropy for RAC-$\mathcal{V}$ and RAC-GE

imation ratio that depends on the constants characterizing these two conditions. The results extend known results for adaptive stochastic optimization on subsets to paths, and enlarges the class of functions known to be efficiently approximable for both problems. We apply the algorithm to two adaptive informative path planning applications with promising results.

Chapter 5

POMDP with Macro Actions

A robot operating in an uncertain environment needs to act appropriately given all the information available and to gather new information that is necessary to complete the assigned task. Chapters 3 and 4 gave algorithms to deal with the information gathering part of the problem. It may be tempting to assume that the robot can plan without uncertainty in mind after it has gathered the necessary information. This may work for simple tasks such as estimating the poses of an object and grasping it. However, many complex tasks and dynamic environments require the robot to interleave between acting towards achieving the goal and acting to gather information or even to do both simultaneously. Dynamic environment and uncertain actions' effects require the robot to constantly monitor its environment while trying to achieve its objective. POMDPs provides a framework to model planning under uncertainty for these complex tasks.

Despite recent successes with large state space POMDPs using point-based algorithms (see Section 5.1.1), long horizon POMDPs are still hard to solve. Heuristics may help to reduce the effective search space to mitigate effects of long horizon but they are unlikely to work well for general problems. The curse of history remains an outstanding issue in scaling up POMDP algorithms. One way to tackle issues with long horizon is to reduce its planning horizon by making use of macro actions that span more than one time step and able to do more within its duration. Macro action can be as simple as a fixed sequence of primitive actions or a complex conditional plan described by a policy or finite state controller.

However, theoretical properties of point-based algorithms do not necessarily carry

over to POMDP with macro actions. We give sufficient conditions for the good theoretical properties to be retained, transforming POMDPs into a particular type of partially observable semi-Markov decision processes (POSMDPs) in which the lengths of macro-actions are not observable.

The final part of this thesis considers POMDPs that can be well approximated by sequence of macro actions. We extend Monte-Carlo value iteration (MCVI) algorithm (Bai, Hsu, Lee, et al., 2010) to use macro actions. A major advantage of the new algorithm is its ability to abstract away the lengths of macro-actions in planning and reduce the effect of long planning horizons. Furthermore, it does not require explicit probabilistic models for macro-actions and treats them just like primitive actions in MCVI. This simplifies macro-action construction and is a major benefit in practice. Macro-MCVI can also be used to construct a hierarchy of macro-actions for planning large spaces. Experiments show that the algorithm is effective with suitably designed macro-actions.

5.1 Related Works

This section reviews prior literature for approximate POMDP algorithms. Section 5.1.1 looks at a few approaches to solve POMDPs approximately. Some of these algorithms have been successful in tackling the "curse of dimensionality" but they are mostly inadequately for long horizon POMDPs. Section 5.1.2 considers temporal abstraction and studies a few POMDP algorithms that explicitly tackle the "curse of history" via macro actions.

5.1.1 Approximate POMDP Algorithms

A number of approaches had been proposed to obtain practical working action strategies for POMDPs. This section reviews simple heuristics that can be useful for some POMDPs as well as more sophisticated approximation and policy search algorithms. These algorithms tackle intractability at various places where they arise. State-of-the-art POMDP solvers are often a combination of approaches that address various difficulties. This section reviews the key ideas behind some successful POMDP algorithms. At the end of this section, we summarize these approaches and draw connections between

them.

MDP Heuristic

A simple yet powerful way to approximate POMDP value function is to assume full observability of all states and take the value of the maximum Q-function action as follows:

$$\hat{V}(b) = \max_{a \in A} \sum_{s \in S} b(s) Q^*_{MDP}(s),$$

where

$$Q^*_{MDP}(s,a) = R(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V *_{MDP} (s')$$

and V^*_{MDP} is the optimal value function for fully observable version of the POMDP. The QMDP approximation $\hat{V}$ is piecewise linear and convex with $|A|$ vectors, each corresponding to one action. It is an upper bound of the optimal value function as it assumes more information than it has.

QMDP value function may be used directly for control and it performs well for problems that do not require many explicit exploratory actions (i.e. going off the optimal MDP path to gather information). It is an important approximation because it works well for many real world problems and is also used as heuristic for other approximation techniques shown in the subsequent sub-section. However, the QMDP heuristic only consider uncertainty of one step and is unable to do long term information gathering beyond that.

Point-based Value Iteration

As we seen in equation 2.5, the number of α-vectors required to represent the exact value function can be potentially exponential. However, it is not necessary to know the exact value function for every belief state if we know the agent starts its operation from some initial belief state. We only care about values of belief states that are reachable from the initial belief state. Based on this insight, *point-based* value iteration algorithms approximate the value function by iteratively updating the value function at a small representative set of belief points. The resulting solution is a policy that is optimal for the sampled belief points. One algorithm that applies point-based backup on a set

of belief points is the Point-based Value Iteration(PBVI) (Pineau, Gordon, and Thrun, 2003).

A point-based backup at belief state b can be done as follows:

1. Construct new α-vector for each action a and each belief b in the representative set B.

$$\Gamma_b^a = r_a + \gamma \sum_{o \in O} \arg\max_{\alpha \in V^t} T^{a,o}\alpha \cdot b \tag{5.1}$$

2. Pick the best α-vector for each belief b in the set B

$$V^{t+1} = \arg\max_{\Gamma_b^a, \forall a \in A}(\Gamma_b^a \cdot b), \forall b \in B \tag{5.2}$$

The point-based backup (5.1) differs from (2.5) in that is it only consider the cross-sum with the projected α-vector that is maximal for belief point b, $\arg\max_{\alpha \in V^t} T^{a,o}\alpha \cdot b$, instead of all projected α-vectors, $T^{a,o}\alpha, \forall \alpha \in V^t$. Out of these $|A||B|$ new α-vectors, the procedure admits only the maximal α-vector for each belief point b to the value function V^{t+1}. The number of α-vector is constant throughout the iterations. Hence, the point-based backup operation is quadratic instead of exponential compared a full backup.

An important implication of point-based backup is that the value function do not necessarily generalizes well for belief points outside the representative set. The value gradient produced by point-based backup may help generalization to beliefs close to the representative set. Hence, it is crucial to get a good representative set that is relevant to the agent's operations.

Belief Space Sampling Strategies The sampling strategy of point-based algorithms directly affects both the quality of value function approximate and its computational efficiency. Hence, it has been the focus of a number of works to produce a superior sampling strategy. (For e.g SARSOP(Kurniawati, Hsu, and Lee, 2008), HSVI(T. Smith and Simmons, 2004), HSVI2(T. Smith and Simmons, 2005), GapMin(Poupart, K. E. Kim, and D. Kim, 2011), FSVI(Shani, Brafman, and Shimony, 2007), and Perseus(Spaan and Vlassis, 2005)).

The sampling strategy used by PBVI first initialize representative set with the initial

belief. It then randomly simulates a one-step action for each action on each belief points in the set and add the new belief for each original point that is furthest away (Euclidean) from the closest point in the set to the set. The idea behind this expansion strategy is to cover as large portion of the reachable belief space as possible.

Heuristic Search Value Iteration (HSVI) (T. Smith and Simmons, 2004) maintains both an upper bound and lower bound of the value function to guide the sampling process. Upper bound of the value function can be computed using optimistic heuristics such as the QMDP. HSVI keeps a belief search tree with initial belief as the root, using a depth-first search it expand a belief by simulating the action having the highest upper bound, and choosing the observation whose resulting belief has the largest gap between the upper and lower bound. HSVI also employed a trial-based asynchronous (Gauss-Seidel) updates to update the values of belief along its path in the depth-first traversal of the belief tree. With these key ingredients, HSVI was able to significant outperform PBVI.

SARSOP (Kurniawati, Hsu, and Lee, 2008) pushes the idea of sampling reachable belief space further by sampling reachable belief space under optimal policy. Since an agent executing an optimal policy it will never encounter any beliefs outside of optimal sequence of actions, it is reasonable to approximate the value function using a set of optimally reachable belief points. The reachable belief space under optimal policy is also believed to be much smaller than the reachable belief space. As the optimal policy is not unknown in advance, SARSOP approximate the optimal reachable belief space by using machine learning techniques to predict the value of beliefs and using the prediction to guide its sampling. Through successive sampling and removing belief points found to be suboptimal, SARSOP iteratively converges to the optimal reachable belief space. SARSOP also aggressively prunes α-vectors that are dominated over the sampled optimally reachable belief points to keep the set of α-vectors small.

In a recent survey study of point-based algorithms by Shani *et al.* (Shani, Pineau, and Kaplow, 2012), they found that different sampling strategies are suited for different type of environments. The stochasticity of actions, noisiness observation, and amount of exploratory actions needed are among the factors that affect performance of a particular sampling strategy. Also, the size of the problem domain is not a good indicator of the

difficulty of the problem.

Through point-based value backup and advanced sampling of the reachable belief space, it appears that the dimensionality of the belief space does not matter anymore since we do not backup over the entire belief. The size of state space is no longer a reliable indicator of difficulty of POMDP when using point-based algorithms. (Hsu, Lee, and Rong, 2007) has shown that approximate optimal POMDP solution can be computed in time polynomial in the covering number of reachable belief space.

Monte-Carlo Methods

To tackle very large, infinite, or continuous state space, one approach is to approximate the belief b by a set of state samples $x^{[1]}, x^{[2]}, \ldots, x^{[M]}$ known as the particles. This set of particles may be updated by the *particle filter* algorithm to obtain a set of particles that approximates the next belief b_{ao} after executing action a and receiving observation o. The particle filter algorithm requires a simulative model that when given a state and action stochastically returns a next state. An important advantage for this representation is that it can approximate arbitrary state space.

MC-POMDP (Thrun, 2000) uses particle sets to represent beliefs. The algorithm maintains a set of particle sets and it gives a value function that is represented as values of this set of particle sets. The value for a new particle set is calculated using nearest neighbor interpolation over known sets.

Monte Carlo Value Iteration (MCVI) (Bai, Hsu, Lee, et al., 2010) also uses particle belief representation. It samples both an agent's state space and the corresponding belief space simultaneously, thus avoiding the prohibitive computational cost of unnecessarily processing these spaces in their entirety. It uses Monte Carlo sampling in conjunction with dynamic programming to compute a policy represented as a finite state controller. The finite state controller gives the value of the policy over the entire belief space without the need of interpolating values over belief points in MC-POMDP.

Both theoretical analysis and experiments on several robotic motion planning tasks indicate that MCVI is a promising approach for planning under uncertainty with very large state spaces, and it has already been applied successfully to compute the threat resolution logic for aircraft collision avoidance systems in 3-D space (Bai, Hsu, Kochen-

derfer, et al., 2011).

As policy evaluation is done using Monte-Carlo sampling, we only get the value of a policy node on sampled belief points without value gradient. There is no way to know if a policy node dominates another node over the belief space. Using pointwise dominance as a condition to prune to too aggressive and may remove many useful nodes. Therefore, the size of the MCVI's policy grows quickly with horizon. It is actually possible to get the α-vectors for the policy by solving the system of linear equations 2.6. But this defeats the purpose of doing Monte-Carlo sampling to handle very large and continuous state spaces as the length of α-vectors does not scales. Also due to the lack of value gradient, it is unable to quickly generalize value for belief never encountered before during policy computation.

Policy Search Using Scenarios

While the size of an optimal POMDP policy may be exponential in horizon, for many real world problems, there may exist small yet useful policies. By searching over a space of bounded size policy, we may find a policy that is good for the problem, if it exists.

Ng and Jordan shown that by transforming any POMDP (or MDP) into one that has only deterministic transition, we can turn the stochastic optimization problem into a deterministic one and apply standard search techniques to find a good policy within a class of policy.

A POMDP M can be transform into a deterministic POMDP M' by augmenting the state with an infinite sequence of random real numbers such that a state is now $(s, p_1, p_2, \ldots)$. When we take an action a in state $(s, p_1, p_2, \ldots)$, we consuming one random number from the sequence $(p_1, p_2, \ldots)$ to generate s' according to the transition distribution. If $p_1 \sim \mathrm{Uniform}[0, 1]$, then distribution of s' is the same as the original POMDP.

A initial state $(s_0, p_1, p_2, \ldots)$ of transformed POMDP M' defines a "scenario" can be interpreted as a "fixed" Monte Carlo trajectory. Thus, the value of a particular policy can be approximated by averaging over the value executing the policy in m scenarios. Since the value of m scenarios is a deterministic function, any standard optimization

method may be used. It was shown this is a uniformly good approximation with "sample complexity" bounds that have polynomial dependence on horizon time.

Using expert knowledge to design the form of policies, the solver applying this approach (PEGASUS) was able find a policy that can autonomously fly a helicopter with complex maneuvers (Ng et al., 2003).

Bounded Finite State Controller Policy Iteration

When the form of policy for a problem is not known, one simple and general class of policy to consider is the finite state controller with fixed bounded number of nodes. Gradient ascent methods (Aberdeen and Baxter, 2002) may be used to search for a bounded finite state controller, but they tend to get stuck in local optima easily.

Poupart and Boutilier (2003) proposed *b*ounded policy iteration algorithm that monotonically improves a finite state controller while keeping the number of nodes fixed. The monotonic improvement is made possible by having a finite state controller whose successor node function is stochastic.

In usual policy iteration, dynamic programming creates new nodes, and an old node is removed when it is pointwise-dominated by a new node. Its incoming edges are redirected to the dominating new node. When an old node is pointwise-dominated by a group of nodes (see Figure 5.1), it cannot be removed. Since removing it means that its incoming edges to have to be redirected to different nodes depending on the belief. However, if we allow stochastic successor function and we stochastically redirect incoming edges to the convex combination of the dominating group of nodes, then we get an improved controller (may not be as good as the usual policy iteration) even though we remove the old node.

Summary

This section reviewed several existing approaches to POMDP solution. Point-based value iteration algorithms reduce the amount of computation by doing dynamic programming updates only on the representative set of belief points instead of the entire belief space. The resultant α-vectors set can be deliberately kept small by pruning for computational efficiency. Point-based methods can be further improved by having bet-

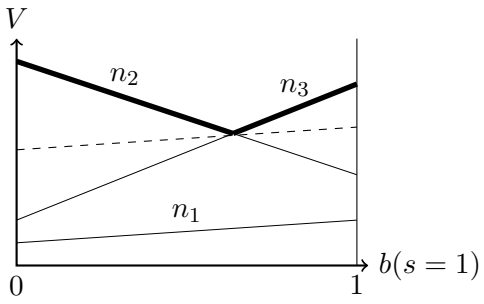

Figure 5.1: Value function of nodes in a finite state controller. n_1 is jointly dominated by n_2 and n_3. Convex combination of n_2 and n_3 shown in dashed.

ter sampling strategies using heuristics such as QMDP to sample a set of belief points that better approximate optimally reachable belief space. Given what we learned from point-based algorithms, the size of state space is no longer a reliable indicator for "hardness" of a problem. It may seem that the "curse of dimensionality" has been broken by sampling the belief space. The limitation is now the space required to represent the α-vectors and transition and observation matrices. By exploiting factored representation of POMDPs, much computational and space efficiency can be gained by processing the factored vectors and matrices. To handle even larger or continuous state space, point-based algorithms can be extended to use simulative POMDP models and Monte-Carlo sampling to sample the state space.

Another class of practical POMDP algorithms is policy search within space of finite size policy. The key idea is to restrict policy search to a small policy space and hope that a good policy can be found there. Ng and Jordan has shown POMDP can be approximated by transforming the original stochastic optimization into a deterministic one so that standard optimization tools can be applied to search within some policy space. Poupart *et al.*proposed a policy iteration algorithm that iteratively improves a finite-state-controller of a fixed size. Ng and Jordan's algorithm are able to benefit from using simulative models to represent the POMDP model, while Poupart's bounded policy iteration can only make use of factored representation of the POMDP model.

However, general POMDP approaches do not automatically generalize well for long horizon problems. Despite the huge recent advances, point-based algorithms are still susceptible to the curse of history. In general, the number of belief points needed to approximate the reachable belief space closely grows with the horizon. Long horizon

problems tend to require large policy due long action sequences required and policy search methods may not find a good policy if there are too little nodes in the controller. Policy search algorithms are also vulnerable to local optima, they may not synthesize the correct long sequence of actions required typically in long horizon problems. Specific approaches are required to address issues arising from long horizon.

5.1.2 POMDPs with Macro Actions

Macro-actions have long been used to speed up planning and learning algorithms for MDPs (see, *e.g.*, (Hauskrecht et al., 1998; Sutton, Precup, and S. Singh, 1999; Barto and Mahadevan, 2003)). There are two types of macro actions (also known as options) in MDP literature, Markov options and Semi-Markov options. Markov option has partial policy that solely depends on the state of the MDP; Semi-Markov option's policy is allowed to depend on the action-state partial history since the option is initiated. Semi-Markov options are more flexible, it allows policy such as those that terminates after some predefined number of time steps. Adding options to MDP turns the process into a Semi-Markov decision process (SMDP). Our work can be viewed as an extension of options to POMDP case. However, since states are partially observable, Markov options are inapplicable in POMDP case. We show in later section that even if we treat belief state as state, Markov options whose policy depends on the belief state can be problematic.

Similarly, macro actions have been used in offline policy computation for POMDPs. Theocharous and Kaelbling (2003) used macro actions in conjunction with Monte Carlo update to speed up computation of Q-value on a grid-based approximation of the belief space. Macro-actions allow the algorithm to experience a smaller part of the belief space resulting in faster backup making it possible to process at higher grid resolution leading to better performance.

Kurniawati, Du, et al. (2010)'s algorithm uses sequences of action and observations (similar to macro action) to sample the belief space more sparsely to cover more space while keeping the number of sampled points low. Macro actions are only used to sample belief spaces but not actually used in the actual policy execution. Dynamic programming backup are done using the primitive actions. Macro-actions can be com-

posed hierarchically to further improve scalability (Dietterich, 2000; Pineau, Roy, and Thrun, 2001). These earlier works rely on vector representations for beliefs and value functions, making it difficult to scale up to large state spaces.

Macro-actions have also been used in online search algorithms for POMDPs. In He, Brunskill, and Roy (2010), open-loop sequences of actions that try to reach high reward or high information state are being constructed on-the-fly and used during the search to help sample values further down in planning horizon. Similar to Kurniawati *et al.*'s algorithm, macro actions are not used in actual policy execution. Instead only the first action of the best sequence of actions is taken, and then the planner searches again from the posterior belief given the observation.

Macro-MCVI is related to E. Hansen and Zhou (2003) where they use programmer-defined task hierarchy to constrain space of their policy. The earlier work uses finite state controllers for policy representation and policy iteration for policy computation, but it has not yet been shown to work on large state spaces.

5.2 Planning with Macro Action

We would like to generalize POMDPs to handle macro-actions. Ideally, the generalization should retain properties of POMDPs such as piecewise linear and convex finite horizon value functions. We would also like the approximation bounds for MCVI (Bai, Hsu, Lee, et al., 2010) to hold with macro-actions.

We would like to allow our macro-actions to be as powerful as possible. A very powerful representation for a macro-action would be to allow it to be an arbitrary mapping from belief to action that will run until some termination condition is met. Unfortunately, the value function of a process with such macro-actions need not even be continuous. Consider the following simple finite horizon example, with horizon one. Assume that there are two primitive actions, both with constant rewards, regardless of state. Consider two macro-actions, one which selects the poorer primitive action all the time while the other which selects the better primitive action for some beliefs. Clearly, the second macro-action dominates the first macro-action over the entire belief space. The reward for the second macro-action takes two possible values depending on which action is selected for the belief. The reward function also forms the optimal value func-

tion of the process and need not even be continuous as the macro-action can be an arbitrary mapping from belief to action.

Next, we give sufficient conditions for the process to retain piecewise linearity and convexity of the value function. We do this by constructing a type of partially observable semi-Markov decision process (POSMDP) with the desired property. The POSMDP does not need to have the length of the macro-action observed, a property that can be practically very useful as it allows the branching factor for search to be significantly smaller. Furthermore, the process is a strict generalization of a POMDP as it reduces to a POMDP when all the macro-actions have length one.

5.3 Partially Observable Semi-Markov Decision Process

Finite-horizon (undiscounted) POSMDP were studied in White (1976). Here, we focus on a type of infinite-horizon discounted POSMDPs whose transition intervals are not observable. Our POSMDP is formally defined as a tuple $(S, \mathcal{A}, \mathcal{O}, \mathbf{T}, \mathbf{R}, \gamma)$, where S is a state space, $\mathcal{A}$ is a macro-action space, $\mathcal{O}$ is a macro-observation space, $\mathbf{T}$ is a joint transition and observation function, $\mathbf{R}$ is a reward function, and $\gamma \in (0, 1)$ is a discount factor. If we apply a macro-action $\mathbf{a}$ with start state s_i, $\mathbf{T} = p(s_j, \mathbf{o}, k|s_i, \mathbf{a})$ encodes the joint conditional probability of the end state s_j, macro-observation $\mathbf{o}$, and the number of time steps k that it takes for $\mathbf{a}$ to reach s_j from s_i. We could decompose $\mathbf{T}$ into a state-transition function and an observation function, but avoid doing so here to remain general and simplify the notation. The reward function $\mathbf{R}$ gives the discounted cumulative reward for a macro-action $\mathbf{a}$ that starts at state s: $\mathbf{R}(s, \mathbf{a}) = \sum_{t=0}^{\infty} \gamma^t \mathrm{E}(r_t|s, \mathbf{a})$, where $\mathrm{E}(r_t|s, \mathbf{a})$ is the expected reward at step t. Here we assume that the reward is 0 once a macro-action terminates.

For convenience, we will work with reweighted beliefs, instead of beliefs. Assuming that the number of states is n, a reweighted belief (like a belief) is a vector of n non-negative numbers that sums to one. By assuming that the POSMDP process will stop with probability $1 - \gamma$ at each time step, we can interpret the reweighted belief as the conditional probability of a state given that the process has not stopped. This gives an interpretation of the reweighted belief in terms of the discount factor. Given a reweighted belief, we compute the next reweighted belief given macro action $\mathbf{a}$ and

observation $\mathbf{o}$, $b' = \tau(b, \mathbf{a}, \mathbf{o})$, as follows:

$$b'(s) = \frac{\sum_{k=1}^{\infty} \gamma^{k-1} \sum_{i=1}^{n} p(s, \mathbf{o}, k | s_i, \mathbf{a}) b(s_i)}{\sum_{k=1}^{\infty} \gamma^{k-1} \sum_{j=0}^{n} \sum_{i=1}^{n} p(s_j, \mathbf{o}, k | s_i, \mathbf{a}) b(s_i)}. \tag{5.3}$$

We will simply refer to the reweighted belief as a belief from here on. We denote the denominator $\sum_{k=1}^{\infty} \gamma^{k-1} \sum_{j=0}^{n} \sum_{i=1}^{n} p(s_j, \mathbf{o}, k | s_i, \mathbf{a}) b(s_i)$ by $p_\gamma(\mathbf{o}|\mathbf{a}, b)$. The value of $\gamma p_\gamma(\mathbf{o}|\mathbf{a}, b)$ can be interpreted as the probability that observation $\mathbf{o}$ is received and the POSMDP has not stopped. Note that $\sum_{\mathbf{o}} p_\gamma(\mathbf{o}|\mathbf{a}, b)$ may sum to less than 1 due to discounting.

A policy π is a mapping from a belief to a macro-action. Let $\mathbf{R}(b, \mathbf{a}) = \sum_s b(s)\mathbf{R}(s, \mathbf{a})$. The value of a policy π can be defined recursively as

$$V_\pi(b) = \mathbf{R}(b, \pi(b)) + \gamma \sum_{\mathbf{o}} p_\gamma(\mathbf{o}|\pi(b), b) V_\pi(\tau(b, \pi(b), \mathbf{o})).$$

Note that the policy operates on the belief and may not know the number of steps taken by the macro-actions. If knowledge of the number of steps is important, it can be added into the observation function in the modeling process.

We now define the backup operator H that operates on a value function V_m and returns V_{m+1}

$$HV(b) = \max_{\mathbf{a}} \big(R(b, \mathbf{a}) + \gamma \sum_{\mathbf{o} \in \mathcal{O}} p_\gamma(\mathbf{o}|\mathbf{a}, b) V(\tau(b, \mathbf{a}, \mathbf{o})) \big). \tag{5.4}$$

The backup operator is a contractive mapping[1].

Lemma 15 (Contraction)**.** *Given value functions U and V, $||HU - HV||_\infty \leq \gamma ||U - V||_\infty$.*

Let the value of an optimal policy, π^*, be V^*. The following theorem is a consequence of the Banach fixed point theorem and Lemma 15.

Theorem 16. *V^* is the unique fixed point of H and satisfies the Bellman equation $V^* = HV^*$.*

We call a policy an m-step policy if the number of times the macro-actions is applied is m. For m-step policies, V^* can be approximated by a finite set of linear functions;

[1] Proofs of the results in this section are in appendix

the weight vectors of these linear functions are called the α-vectors.

Theorem 17 (Piecewise Linearity and Convex)**.** *The value function for an m-step policy is piecewise linear and convex and can be represented as*

$$V_m(b) = \max_{\alpha \in \Gamma_m} \sum_{s \in S} \alpha(s) b(s) \tag{5.5}$$

where Γ_m is a finite collection of α-vectors.

As V_m is convex and converges to V^*, V^* is also convex.

5.3.1 Macro-action Construction

We would like to construct macro-actions from primitive actions of a POMDP in order to use temporal abstraction to help solve difficult POMDP problems. A partially observable Markov decision process (POMDP) is defined by finite state space S, finite action space A, a reward function $R(s, a)$, an observation space O, and a discount $\gamma \in (0, 1)$.

We discuss some of the issues here. They are further illustrated in the Section 5.5 where we show how the use of appropriately constructed macro-actions can improve the performance of MCVI.

In our POSMDP, the probability function $p(s_j, \mathbf{o}, k | s_i, \mathbf{a})$ for a macro-action must be independent of the history given the current state s_i; hence the selection of primitive actions and termination conditions within the macro-action cannot depend on the belief. We examine some allowable dependencies here. Due to partial observability, it is often not possible to allow the primitive action and the termination condition to be functions of the initial state. Dependence on the portion of history that occurs after the macro-action has started is, however, allowed. In some POMDPs, a subset of the state variables is always observed and can be used to decide the next action. In fact, we may sometimes explicitly construct observed variables to remember relevant parts of the history prior to the start of macro-action (see Section 5.5); these can be considered as parameters that are passed on to the macro-action. Hence, one way to construct the next action in a macro-action is to make it a function of the history since the macro-action started, $x_k, a_k, o_{k+1}, \ldots, x_{t-1}, a_{t-1}, o_t, x_t$, where x_i is the fully observable subset of state variables at time i, and k is the starting time of the macro-action.

Similarly, when the termination criterion and the observation function of the macro-action depends only on the history $x_k, a_k, o_{k+1}, \ldots, x_{t-1}, a_{t-1}, o_t, x_t$, the macro-action can retain a transition function that is independent of the history given the initial state. Note that the observation to be passed on to the POSMDP to create the POSMDP observation space, $\mathcal{O}$, is part of the design trade off - usually it is desirable to reduce the number of observations in order to reduce complexity without degrading the value of the POSMDP too much. In particular, we may not wish to include the execution length of the macro-action if it does not contribute much towards obtaining a good policy.

5.4 Monte Carlo Value Iteration with Macro-Actions

We have shown that if the action space $\mathcal{A}$ and the observation space $\mathcal{O}$ of a POSMDP are discrete, then the optimal value function V^* can be approximated arbitrarily closely by a piecewise-linear, convex function. Unfortunately, when S is very high-dimensional (or continuous), a vector representation is no longer effective. In this section, we show how the Monte Carlo Value Iteration (MCVI) algorithm (Bai, Hsu, Lee, et al., 2010), which has been designed for POMDPs with very large or infinite state spaces, can be extended to POSMDP.

Instead of α-vectors, MCVI uses an alternative policy representation called a *policy graph* G. A policy graph is a directed graph with labeled nodes and edges. Each node of G is labeled with a macro-action $\mathbf{a}$ and each edge of G is labeled with an observation $\mathbf{o}$. To execute a policy π_G, it is treated as a finite state controller whose states are the nodes of G. Given an initial belief b, a starting node v of G is selected and its associated macro-action $\mathbf{a}_v$ is performed. The controller then transitions from v to a new node v' by following the edge (v, v') labeled with the observation received, $\mathbf{o}$. The process then repeats with the new controller node v'.

Let $\pi_{G,v}$ denote a policy represented by G, when the controller always starts in node v of G. We define the value $\alpha_v(s)$ to be the expected total reward of executing $\pi_{G,v}$ with initial state s. Hence

$$V_G(b) = \max_{v \in G} \sum_{s \in S} \alpha_v(s) b(s). \tag{5.6}$$

V_G is completely determined by the α-functions associated with the nodes of G.

Algorithm 6 MC-Backup of a policy graph G at a belief $b \in \mathcal{B}$ with N samples.

MC-BACKUP(G, b, N)

For each action $\mathbf{a} \in \mathcal{A}$, $R_{\mathbf{a}} \leftarrow 0$.
For each action $\mathbf{a} \in \mathcal{A}$, each observation $\mathbf{o} \in \mathcal{O}$, and each node $v \in G$, $V_{\mathbf{a},\mathbf{o},v} \leftarrow 0$.
for each action $\mathbf{a} \in \mathcal{A}$ **do**
    for $i = 1$ to N **do**
        Sample a state s_i with probability $b(s_i)$.
        Simulate taking macro-action $\mathbf{a}$ in state s_i. Generate a new state s_i', observation $\mathbf{o}_i$, and discounted reward $R'(s_i, \mathbf{a})$ by sampling from $p(s_j, \mathbf{o}, k|s_i, \mathbf{a})$.
        $R_{\mathbf{a}} \leftarrow R_{\mathbf{a}} + R'(s_i, \mathbf{a})$.
        for each node $v \in G$ **do**
            Set V' to be the expected total reward of simulating the policy represented by G, with initial controller state v and initial state s_i'.
            $V_{\mathbf{a},\mathbf{o}_i,v} \leftarrow V_{\mathbf{a},\mathbf{o}_i,v} + V'$.
    for each observation $\mathbf{o} \in \mathcal{O}$ **do**
        $V_{\mathbf{a},\mathbf{o}} \leftarrow \max_{v \in G} V_{\mathbf{a},\mathbf{o},v}$.
        $v_{\mathbf{a},\mathbf{o}} \leftarrow \text{argmax}_{v \in G} V_{\mathbf{a},\mathbf{o},v}$.
    $V_{\mathbf{a}} \leftarrow (R_{\mathbf{a}} + \gamma \sum_{\mathbf{o} \in \mathcal{O}} V_{\mathbf{a},\mathbf{o}})/N$.
$V^* \leftarrow \max_{\mathbf{a} \in \mathcal{A}} V_{\mathbf{a}}$.
$\mathbf{a}^* \leftarrow \text{argmax}_{\mathbf{a} \in \mathcal{A}} V_{\mathbf{a}}$.
Create a new policy graph G' by adding a new node u to G. Label u with $\mathbf{a}^*$. For each $\mathbf{o} \in \mathcal{O}$, add the edge $(u, v_{\mathbf{a}^*,\mathbf{o}})$ and label it with $\mathbf{o}$. **return** G'.

5.4.1 MC-Backup

One way to approximate the value function is to repeatedly run the backup operator H starting from an arbitrary value function until it is close to convergence. This algorithm is called *value iteration* (VI). Value iteration can be carried out on policy graphs as well, as it provides an implicit representation of a value function. Let V_G be the value function for a policy graph G. Substituting (5.6) into (5.4), we get

$$HV_G(b) = \max_{\mathbf{a} \in \mathcal{A}} \Big\{ \sum_{s \in S} \mathbf{R}(s, a) b(s) + \sum_{\mathbf{o} \in \mathcal{O}} p_\gamma(\mathbf{o}|\mathbf{a}, b) \max_{v \in G} \sum_{s \in S} \alpha_v(s) b'(s) \Big\}. \tag{5.7}$$

It is possible to then evaluate the right-hand side of (5.7) via sampling and Monte Carlo simulation at a belief b. The outcome is a new policy graph G' with value function $\hat{H}_b V_G$. This is called MC-backup of G at b (Algorithm 6) (Bai, Hsu, Lee, et al., 2010).

There are $|\mathcal{A}||G|^{|O|}$ possible ways to generate a new policy graph G' which has one new node compared to the old policy graph node. Algorithm 6 computes an estimate of the best new policy graph at b using only $N|\mathcal{A}||G|$ samples. Furthermore, we can show that MC-backup approximates the standard VI backup (equation (5.7)) well at b, with error decreasing at the rate $O(1/\sqrt{N})$. Let $R_{\max}$ be the largest absolute value of the

reward, $|r_t|$, at any time step.

Theorem 18. *Given a policy graph G and a point $b \in B$,* MC-BACKUP(G, b, N) *produces an improved policy graph such that*

$$|\hat{H}_b V_G(b) - HV_G(b)| \leq \frac{2R_{\max}}{1-\gamma}\sqrt{\frac{2\big(|\mathcal{O}|\ln|G| + \ln(2|\mathcal{A}|) + \ln(1/\tau)\big)}{N}},$$

with probability at least $1-\tau$.

The proof uses Hoeffding bound together with union bound. Details can be found in (Bai, Hsu, Lee, et al., 2010).

MC-backup can be combined with point-based POMDP planning, which samples the belief space $\mathcal{B}$. Point-based POMDP algorithms use a set B of points sampled from $\mathcal{B}$ as an approximate representation of $\mathcal{B}$. In contrast to the standard VI backup operator H, which performs backup at every point in $\mathcal{B}$, the operator $\hat{H}_B$ applies MC-BACKUP(G_m, b, N) on a policy graph G_m at every point in B. This results in $|B|$ new policy graph nodes. $\hat{H}_B$ then produces a new policy graph G_{m+1} by adding the new policy graph nodes to the previous policy graph G_m.

Let $\delta_B = \sup_{b\in\mathcal{B}} \min_{b'\in B} \|b - b'\|_1$ be the maximum L_1 distance from any point in $\mathcal{B}$ to the closest point in B. Let V_0 be value function for some initial policy graph and $V_{m+1} = \hat{H}_B V_m$. The theorem below bounds the approximation error between V_m and the optimal value function V^*.

Theorem 19. *For every $b \in B$,*

$$|V^*(b) - V_m(b)| \leq \frac{2R_{\max}}{(1-\gamma)^2}\sqrt{\frac{2\big(|\mathcal{O}|\ln(|B|m) + \ln(2|\mathcal{A}|) + \ln(|B|m/\tau)\big)}{N}} + \frac{2R_{\max}}{(1-\gamma)^2}\delta_B + \frac{2\gamma^m R_{\max}}{(1-\gamma)},$$

with probability at least $1-\tau$.

The proof requires the contraction property and a Lipschitz property that can be derived from the piece-wise linearity of the value function. Having established those results in Section 5.3, the rest of the proof follows from the proof in (Bai, Hsu, Lee, et al., 2010). The first term in the bound in Theorem 19 comes from Theorem 18, showing that the error from sampling decays at the rate $O(1/\sqrt{N})$ and can be reduced by taking a large enough sample size. The second term depends on how well the set B covers $\mathcal{B}$

and can be reduced by sampling a larger number of beliefs. The last term depends on the number of MC-backup iterations and decays exponentially with m.

5.4.2 Algorithm

Theorem 19 bounds the performance of the algorithm when given a set of beliefs. Macro-MCVI, like MCVI, samples beliefs incrementally in practice and performs backup at the sampled beliefs. Branch and bound is used to avoid sampling unimportant parts of the belief space. See (Bai, Hsu, Lee, et al., 2010) for details.

The other important component in a practical algorithm is the generation of next belief; Macro-MCVI uses a particle filter for that. Given the macro-action construction as described in Section 5.3.1 a simple particle filter is easily implemented to approximate the next belief function in equation (5.3): sample a set of states from the current belief; from each sampled state, simulate the current macro-action until termination, keeping track of its path length, t; if the observation at termination matches the desired observation, keep the particle; the set of particles that are kept are weighted by γ^t and then renormalized to form the next belief[2]. Similarly, MC-backup is performed by simply running simulations of the macro-actions - there is no need to store additional transition and observation matrices, allowing the method to run for very large state spaces.

We give a short description of the algorithm for completeness.

Let $\mathcal{R} \subseteq \mathcal{B}$ be a subset of beliefs reachable from a given initial belief $b_0 \in \mathcal{B}$ under arbitrary sequences of macro-actions and observations. MCVI samples from this set rather than the whole space to get a more relevant set of beliefs. The sampled beliefs can be structured as a tree $T_{\mathcal{R}}$, where the root of $T_{\mathcal{R}}$ is the initial belief b_0. If b is a node of $T_{\mathcal{R}}$ and b' is a child of b in $T_{\mathcal{R}}$, then $b' = \tau(b, \mathbf{a}, \mathbf{o})$ for some $\mathbf{a} \in \mathcal{A}$ and $\mathbf{o} \in \mathcal{O}$.

MCVI maintains both upper and lower bounds on $V^*(b)$ each node b of $T_{\mathcal{R}}$. To sample a new belief, it starts from the root of $T_{\mathcal{R}}$ and traverse a single path down until reaching a leaf of $T_{\mathcal{R}}$. At a node b along the path, it chooses the action $\mathbf{a}$ with the highest upper bound and the observation $\mathbf{o}$ that has the largest weighted gap between the upper and lower bounds. New beliefs are constructed using particle filtering and sampling terminates when the gap between the upper and lower bounds is sufficiently

[2]More sophisticated approximation of the belief can be constructed but may require more knowledge of the underlying POMDP and more computation.

small. MC-backup is then performed on all the nodes along this path to improve the lower bound estimate. At the same time, a sampled approximation of (5.4) is used to improve the upper bounds of the same nodes. This is repeated until the gap between upper and lower bounds at the root of $T_{\mathcal{R}}$ reaches the desired value.

To initialize, G is given a set of fixed macro-action policies representing prior knowledge of the problem; if no knowledge is available, each macro-actions $\mathcal{A}$ may be used to create a simple policy for the initial set by by looping back to itself after every observation. For upper bound, a heuristic upper bound is used to initialize new beliefs; if no knowledge is available $R_{\max}/(1-\gamma)$ can be used.

5.5 Experiments

We now illustrate the use of macro-actions for temporal abstraction in three POMDPs of varying complexity. Their state spaces range from relatively small to very large. Correspondingly, the macro-actions range from relatively simple ones to much more complex ones forming a hierarchy.

5.5.1 Underwater Navigation:

The underwater navigation task was introduced in Kurniawati, Hsu, and Lee (2008). In this task, an autonomous underwater vehicle (AUV) navigates in an environment modeled as 51 x 52 grid map (see Figure 5.2). The AUV needs to move from the left border to the right border while avoiding the rocks scattered near its destination. The AUV has six actions: move north, move south, move east, move north-east, move south-east or stay in the same location. Due to poor visibility, the AUV can only localize itself along the top or bottom borders where there are beacon signals.

This problem has several interesting characteristics. First, the relatively small state space size of 2653 means that solvers that use α-vectors, such as SARSOP (Kurniawati, Hsu, and Lee, 2008) can be used. Second, the dynamics of the robot is actually noise-less; hence the main difficulty is actually localization from the robot's initially unknown location.

We use 5 macro-actions that move in a direction (north, south, east, north-east, or south-east) until either a beacon signal or the destination is reached. We also define an

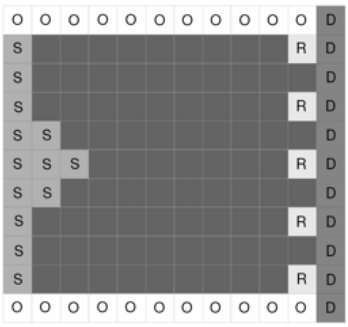

Figure 5.2: Underwater Navigation: A reduced map with a 11 $\times$ 12 grid is shown with "S" marking the possible initial positions, "D" marking the destinations, "R" marking the rocks and "O" marking the locations where the robot can localize completely.

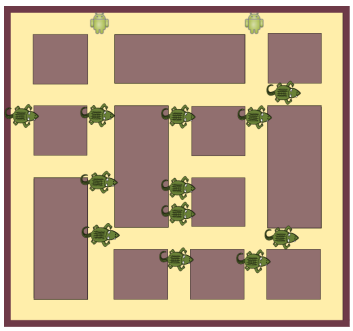

Figure 5.3: Collaborative search and capture: Two robotic agents catching 12 escaped crocodiles in a 21 $\times$ 21 grid.

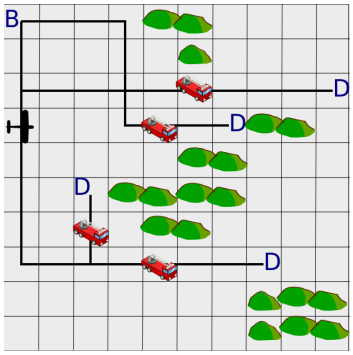

Figure 5.4: Vehicular ad-hoc networking: An UAV maintains ad-hoc network over four ground vehicles in a 10 $\times$ 10 grid with "B" marking the base and "D" the destinations.

additional macro-action that: navigates to the nearest goal location if the AUV position is known, or simply stays in the same location if the AUV position is not known. To enable proper behavior of the last macro-action, we augment the state space with a fully observable state variable that indicates the current AUV location. The variable is initialized to a value denoting "unknown" but takes the value of the current AUV location after the beacon signal is received. This gives a simple example where the original state space is augmented with a fully observable state variable to allow more sophisticated macro-action behavior.

5.5.2 Collaborative Search and Capture:

In this problem, a group of crocodiles had escaped from its enclosure into the environment and two robotic agents have to collaborate to hunt down and capture the crocodiles (see Figure 5.3). Both agents are centrally controlled and each agent can make a one step move in one of the four directions (north, south, east and west) or stay still at each time instance. There are twelve crocodiles in the environment. At every time instance, each crocodile moves to a location furthest from the agent that is nearest to it with a probability $1 - p$ ($p = 0.05$ in the experiments). With a probability p, the crocodile moves randomly. A crocodile is captured when it is at the same location as an agent. The agents do not know the exact location of the crocodiles, but each agent knows the number of crocodiles in the top left, top right, bottom left and bottom right quadrants around itself from the noise made by the crocodiles. Each captured crocodile gives a reward of 10, while movement is free.

We define twenty-five macro actions where each agent moves (north, south, east, west, or stay) along a passage way until one of them reaches an intersection. In addition, the macro-actions only return the observation it makes at the point when the macro-action terminates, reducing the complexity of the problem, possibly at a cost of some suboptimality. In this problem, the macro-actions are simple, but the state space is extremely large (approximately 179^{14}).

5.5.3 Vehicular Ad-hoc Network:

In a post disaster search and rescue scenario, a group of rescue vehicles are deployed for operation work in an area where communication infrastructure has been destroyed. The rescue units need high-bandwidth network to relay images of ground situations. An Unmanned Aerial Vehicle (UAV) can be deployed to maintain WiFi network communication between the ground units. The UAV needs to visit each vehicle as often as possible to pick up and deliver data packets (Sivakumar and Tan, 2010) (see Figure 5.4).

In this task, 4 rescue vehicles and 1 UAV navigates in a terrain modeled as a 10 x 10 grid map. There are obstacles on the terrain that are impassable to ground vehicle but passable to UAV. The UAV can move in one of the four directions (north, south, east, and west) or stay in the same location at every time step. The vehicles set off from the same base and move along some predefined path towards their pre-assigned destinations where they will start their operations, randomly stopping along the way. Upon reaching its destination, the vehicle may roam around the environment randomly while carrying out its mission. The UAV knows its own location on the map and can observe the location of a vehicle if they are in the same grid square. To elicit a policy with low network latency, there is a penalty of $-0.1\times$ number of time steps since last visit of a vehicle for each time step for each vehicle. There is a reward of 10 for each time a vehicle is visited by the UAV. The state space consists of the vehicles' locations, UAV location in the grid map and the number of time steps since each vehicle is last seen (for computing the reward).

We abstract the movements of UAV to search and visit a single vehicle as macro actions. There are two kinds of search macro actions for each vehicle: search for a vehicle along its predefined path and search for a vehicle that has started to roam randomly. To enable the macro-actions to work effectively, the state space is also augmented with the previous seen location of each vehicle. Each macro-action is in turn hierarchically constructed by solving the simplified POMDP task of searching for a single vehicle on the same map using basic actions and some simple macro-actions that move along the paths. This problem has both complex hierarchically constructed macro-actions and very large state space.

5.5.4 Experimental setup

We applied Macro-MCVI to the above tasks and compared its performance with the original MCVI algorithm. We also compared with a state-of-the-art off-line POMDP solver, SARSOP (Kurniawati, Hsu, and Lee, 2008), on the underwater navigation task. SARSOP could not run on the other two tasks, due to their large state space sizes. For each task, we ran Macro-MCVI until the average total reward stabilized. We then ran the competing algorithms for at least the same amount of time. The exact running times are difficult to control because of our implementation limitations. To confirm the comparison results, we also ran the competing algorithms 100 times longer when possible. All experiments were conducted on a 16 core Intel Xeon 2.4Ghz computer server.

Neither MCVI nor SARSOP uses macro-actions. We are not aware of other efficient off-line macro-action POMDP solvers that have been demonstrated on very large state space problems. Some online search algorithms, such as PUMA (He, Brunskill, and Roy, 2010), use macro-actions and have shown strong results. Online search algorithms do not generate a policy, making a fair comparison difficult. Despite that, they are useful as baseline references; we implement a variant of PUMA as a one such reference. In our experiments, we simply gave the online search algorithms as much or more time than Macro-MCVI and report the results here. PUMA uses open-loop macro-actions. As a baseline reference for online solvers with closed-loop macro-actions, we also created an online search variant of Macro-MCVI by removing the MC-backup component. We refer to this variant as *Online-Macro*. It is similar to other recent online POMDP algorithms (Ross et al., 2008), but uses the same closed-loop macro-actions as MCVI does.

5.5.5 Results

The performance of the different algorithms is shown in Figure 5.1 with 95% confidence intervals.

The underwater navigation task consists of two phases: the localization phase and navigate to goal phase. Macro-MCVI's policy takes one macro-action, "moving north-east until reaching the border", to localize and another macro-action, "navigating to the

Table 5.1: Performance comparison.

	Reward	Time(s)
Underwater Navigation		
Macro-MCVI	**749.30** $\pm$ 0.28	1
MCVI	678.05 $\pm$ 0.48	4
	725.28 $\pm$ 0.38	100
SARSOP	710.71 $\pm$ 4.52	1
	730.83 $\pm$ 0.75	100
PUMA	697.47 $\pm$ 4.58	1
Online-Macro	746.10 $\pm$ 2.37	1
Collaborative Search & Capture		
Macro-MCVI	**17.04** $\pm$ 0.03	120
MCVI	13.14 $\pm$ 0.04	120
	16.38 $\pm$ 0.05	12000
PUMA	1.04 $\pm$ 0.91	144
Online-Macro	0	3657
Vehicular Ad-Hoc Network		
Macro-MCVI	**-323.55** $\pm$ 3.79	29255
MCVI	-1232.57 $\pm$ 2.24	29300
Greedy	-422.26 $\pm$ 3.98	28800

goal", to reach the goal. In contrast, both MCVI and SARSOP fail to match the performance of Macro-MCVI even when they are run 100 times longer. Online-Macro does well, as the planning horizon is short with the use of macro-actions. PUMA, however, does not do as well, as it uses the less powerful open-loop macro-actions, which move in the same direction for a fixed number of time steps.

For the collaborative search & capture task, MCVI fails to match the performance of Macro-MCVI even when it is run for 100 times longer. PUMA and Online-Macro do badly as they fail to search deep enough and do not have the benefit of reusing sub-policies obtained from the backup operation. To confirm that it is the backup operation and not the shorter per macro-action time that is responsible for the performance difference, we ran Online-Macro for a much longer time and found the result unchanged.

The vehicular ad-hoc network task was solved hierarchically in two stages. We first used Macro-MCVI to solve for the policy that finds a single vehicle. This stage took roughly 8 hours of computation time. We then used the single-vehicle policy as a macro-action and solved for the higher-level policy that plans over the macro-actions. Although it took substantial computation time, Macro-MCVI generated a reasonable policy in the end. In contrast, MCVI, without macro-actions, fails badly for this task.

Due to the long running time involved, we did not run MCVI 100 times longer. To confirm that that the policy computed by Macro-MCVI at the higher level of the hierarchy is also effective, we manually crafted a greedy policy over the single-vehicle macro-actions. This greedy policy always searches for the vehicle that has not been visited for the longest duration. The experimental results indicate that the higher-level policy computed by Macro-MCVI is more effective than the greedy policy. We did not apply online algorithms to this task, as we are not aware of any simple way to hierarchically construct macro-actions online.

5.6 Conclusion

We have successfully extended MCVI, an algorithm for solving very large state space POMDPs, to include macro-actions. This allows MCVI to use temporal abstraction to help solve difficult POMDP problems. The method inherits the good theoretical properties of MCVI and is easy to apply in practice. Experiments show that it can substantially improve the performance of MCVI when used with appropriately chosen macro-actions.

Chapter 6

Conclusion

Partially observable Markov decision process (POMDP) is sufficiently general for a wide range of problems that require planning under uncertainty. However, it is hard to obtain useful approximate solution to POMDPs except for very small problems. This thesis identifies a few subclasses of POMDPs and proposes efficient approximation algorithms to solve them.

6.1 Informative Path Planning

The IPP problem optimizes a path for a robot to move around to sense and gather information. While it is a subclass of POMDPs, it captures the information gathering aspect of POMDPs where the state information of interest is static.

Recursive Adaptive Identification is a new polynomial time algorithm that solves the adaptive IPP problem with polylogarithmic approximation bound when the observations are noiseless and the robot move in metric spaces. The key strategy of this algorithm is to construct tour of "informative" locations such that at least half the of probability of hypotheses is eliminated by the end of the tour. This is repeated recursively on the remaining hypotheses until only one hypothesis is left. RAId makes use approximate group Steiner tree algorithm to trade off movement cost with information gain in polynomial time. RAId was shown to work well on a few information gathering tasks in simulation.

We also give a simple extension, *Sampled*-RAId to allow RAId to work on IPP problems with noisy observations. *Sampled*-RAId transform a noisy IPP problem to

a noiseless one by sampling the vector of "realized" observations at every location as hypotheses in the noiseless IPP problem. *Sampled*-RAId then applies RAId to solve the transformed problem. *Sampled*-RAId gives satisfactory results in experiments but it lacks theoretical guarantees. Although this may work in practice, there is no theoretical guarantee for *Sampled*-RAId because it is optimizing the wrong objective. It seeks to differentiate the sampled observation vector.

6.2 Adaptive Stochastic Optimization

We propose two novel conditions, the marginal likelihood bound and the marginal likelihood rate bound conditions for pointwise submodular monotone functions. The marginal likelihood bound and the marginal likelihood rate bound tie marginal likelihood a history to the worst case objective value of the function. While adaptive stochastic optimization is NP-hard in general, these conditions characterize classes of adaptive stochastic optimization where there can be efficient approximate solution

Our algorithm, *Recursive Adaptive Coverage* (RAC), runs in polynomial time with an approximation ratio that depends on the constants characterizing these two conditions. The results extend known results for adaptive stochastic optimization prbolems on subsets to adaptive stochastic optimization problems on paths, and enlarges the class of functions known to be efficiently approximable for both type of problems.

We use RAC to solve the noisy adaptive IPP problem. We first reduce the IPP problem to equivalence class determination problem and then apply RAC to a Gibbs error objective function to solve the ECD problem near-optimally. The Gibbs error function satisfies marginal likelihood rate bound and hence it is near-optimal for noisy adaptive IPP problem. Empirically, RAC with Gibbs error function gives good results when evaluated on two noisy IPP tasks in simulation.

6.3 Temporal Abstraction with Macro Actions

A POMDP model can optimize a plan that is robust to effect uncertainty and imperfect state information. Despite recent advances in POMDP algorithms, it remains hard to scale up due to "curse of dimensionality" and "curse of history".

Macro actions are temporally extended actions that can take more than one time step. We develop theoretical understanding of macro actions and apply it to a practical algorithm to tackle the "curse of history" of POMDPs.

Adding macro actions to POMDP results in a Partially Observable Semi-Markov Decision Process. In general, good theoretical properties of POMDP such as piecewise linearity and convexity of its finite horizon value function do not carry over to POSMDP. We give sufficient conditions for macro actions to retain these properties.

Macro Monte-Carlo Value Iteration is a new algorithm that exploits temporal abstraction in POMDPs using macro actions. Macro-MCVI extends Monte-Carlo Value Iteration, a point-based POMDP algorithm that can scale up to large and continuous state spaces to use macro actions. Macro-MCVI only needs a generative model for macro actions, making it easy to specify macro actions. Using suitably constructed macro actions that satisfies our sufficient conditions, Macro-MCVI retains the performance guarantees of MCVI. Experiment shows significant performance improvement over MCVI. We also demonstrated hierarchical composing of macro actions in one experiment, where the macro actions are policies obtained using Macro-MCVI.

6.4 Future Work

The work in this thesis can be expanded in a few directions. First, we can expand the applications of the algorithms developed in this thesis. As marginal likelihood bound and marginal likelihood rate bound are novel conditions, we need to discover new applications where their objective functions satisfy the conditions and apply RAC to approximate them. We also intend to apply RAId and RAC to real data from robot sensors to evaluate its performance in real world systems.

Second, the algorithms developed in this thesis can be extended to new problem domains such as Bayesian reinforcement learning. In Bayesian reinforcement learning, an agent explores a world modeled by an unknown MDP and aims to act optimally given its knowledge of the world. Assuming the model parameters are discrete, Bayesian reinforcement learning can be cast as a noisy IPP problem where each parameter value is a hypothesis and there is a prior over them. We further transform MDP into a deterministic process by assuming a state will always transit to its most likely next state. This

is similar to the mostly likely outcome assumption used in RAC to simplify adaptive planning. Exploration is implicit in this method. If the state transits to some state other than the most likely one, then the agent learns more about the true MDP model in the same way as the agent eliminates more than half of probabilities of hypotheses when it receives an informative observation in an IPP problem. With some mild assumptions, we can try to prove that applying RAC strategy results in a Bayesian reinforcement learning policy that has cost competitive with an optimal exploration policy.

Finally, we aim to develop new algorithms using the algorithmic structure of RAId and RAC for other stochastic partially observable problems. The algorithmic structure of RAId and RAC: planning using the most likely outcome assumption, exploiting information gain if the most likely outcome does not occur, and hedging against over-commitment, is general enough to be applicable to a large range of learning and planning applications. We can discover new conditions that work well with the most likely outcome assumption and exploit them for theoretical guarantees.

Bibliography

Aberdeen, D. and J. Baxter (2002). "Scaling internal-state policy-gradient methods for POMDPs". In: *Proc. Int. Conf. on Machine Learning*, pp. 3–10.

Asadpour, A., H. Nazerzadeh, and A. Saberi (2008). "Stochastic Submodular Maximization". English. In: *Internet and Network Economics*, pp. 477–489.

Bai, H., D. Hsu, M. Kochenderfer, et al. (2011). "Unmanned Aircraft Collision Avoidance using Continuous-State POMDPs". In: *Proc. Robotics: Science and Systems*.

Bai, H., D. Hsu, W. S. Lee, et al. (2010). "Monte Carlo Value Iteration for Continuous-State POMDPs". In: *Algorithmic Foundations of Robotics IX—Proc. Int. Workshop o n the Algorithmic Foundations of Robotics (WAFR)*. Springer, pp. 175–191.

Bandyopadhyay, T. et al. (2013). "Intention-aware motion planning". In: *Algorithmic Foundations of Robotics X*. Springer, pp. 475–491.

Barto, A. G. and S. Mahadevan (2003). "Recent Advances in Hierarchical Reinforcement Learning". In: *Discrete Event Dynamic Systems* 13, p. 2003.

Calinescu, G. and A. Zelikovsky (2005). "The polymatroid steiner problems". In: *J. Combinatorial Optimization* 9.3, pp. 281–294.

Caselton, W. F. and J. V. Zidek (1984). "Optimal monitoring network designs". In: *Statistics & Probability Letters* 2.4, pp. 223–227.

Cassandra, A., M. L. Littman, and N. L. Zhang (1997). "Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes". In: *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, pp. 54–61.

Chakaravarthy, V. et al. (2007). "Decision trees for entity identification: approximation algorithms and hardness results". In: *Proc. ACM Symp. on Principles of Database Systems*.

Chekuri, C. and M. Pal (2005). “A recursive greedy algorithm for walks in directed graphs”. In: *Proc. IEEE Symp. on Foundations of Computer Science*, pp. 245–253.

Christofides, N. (1976). *Worst-case analysis of a new heuristic for the travelling salesman problem*. Tech. rep. 388. Graduate School of Industrial Administration, Carnegie Mellon University.

Cuong, N. V., W. S. Lee, and N. Ye (2014). “Near-optimal Adaptive Pool-based Active Learning with General Loss”. In: *Proc. Uncertainty in Artificial Intelligence*.

Cuong, N. V., W. S. Lee, N. Ye, et al. (2013). “Active Learning for Probabilistic Hypotheses Using the Maximum Gibbs Error Criterion”. In: *Advances in Neural Information Processing Systems (NIPS)*, pp. 1457–1465.

Dean, B., M. Goemans, and J. Vondrdk (2004). “Approximating the stochastic knapsack problem: The benefit of adaptivity”. In: *Proc. IEEE Symp. on Foundations of Computer Science*, pp. 208–217.

Dietterich, T. G. (2000). “Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition”. In: *J. Artificial Intelligence Research* 13, pp. 227–303.

Fakcharoenphol, J., S. Rao, and K. Talwar (2003). “A Tight Bound on Approximating Arbitrary Metrics by Tree Metrics”. In: *Proceedings of the Thirty-fifth Annual ACM Symposium on Theory of Computing*. STOC ’03. New York, NY, USA: ACM, pp. 448–455. ISBN: 1-58113-674-9. DOI: 10.1145/780542.780608.

Feder, H., J. Leonard, and C. Smith (1999). “Adaptive mobile robot navigation and mapping”. In: *Int. J. Robotics Research* 18.7, pp. 650–668.

Fox, D., W. Burgard, and S. Thrun (1998). “Active Markov localization for mobile robots”. In: *Robotics & Autonomous Systems* 25.3, pp. 195–207.

Golovin, D. and A. Krause (2011). “Adaptive Submodularity: Theory and Applications in Active Learning and Stochastic Optimization”. In: *J. Artificial Intelligence Research* 42.1, pp. 427–486.

Golovin, D., A. Krause, and D. Ray (2010). “Near-Optimal Bayesian Active Learning with Noisy Observations”. In: *Advances in Neural Information Processing Systems (NIPS)*, pp. 766–774.

Guillory, A. and J. Bilmes (2010). “Interactive Submodular Set Cover”. In: *International Conference on Machine Learning (ICML)*. Haifa, Israel.

— (2011). “Simultaneous learning and covering with adversarial noise”. In: *Proc. Int. Conf. on Machine Learning*, pp. 369–376.

Gupta, A., V. Nagarajan, and R. Ravi (2010). “Approximation Algorithms for Optimal Decision Trees and Adaptive TSP Problems”. In: *Automata, Languages and Programming*. Ed. by S. Abramsky et al. Lecture Notes in Computer Science 6198. Springer Berlin Heidelberg, pp. 690–701. ISBN: 978-3-642-14164-5, 978-3-642-14165-2.

Hansen, E. A. (1998). “An improved policy iteration algorithm for partially observable MDPs”. In: *Advances in Neural Information Processing Systems*, pp. 1015–1021.

Hansen, E. and R. Zhou (2003). “Synthesis of hierarchical finite-state controllers for POMDPs”. In: *Proc. Int. Conf. on Automated Planning and Scheduling*.

Hauskrecht, M. et al. (1998). “Hierarchical solution of Markov decision processes using macro-actions”. In: *Proc. Conf. on Uncertainty in Artificial Intelligence*. Citeseer, pp. 220–229.

He, R., E. Brunskill, and N. Roy (2010). “PUMA: Planning under uncertainty with macro-actions”. In: *Proc. AAAI Conf. on Artificial Intelligence*.

Hoey, J. et al. (2007). “Assisting persons with dementia during handwashing using a partially observable Markov decision process”. In: *Proc. Int. Conf. on Vision Systems*. Vol. 65, p. 66.

Hollinger, G., S. Singh, et al. (2009). “Efficient Multi-robot Search for a Moving Target”. In: *Int. J. Robotics Research* 28.2, pp. 201–219.

Hollinger, G. and G. Sukhatme (2013). “Sampling-based Motion Planning for Robotic Information Gathering.” In: *Proc. Robotics: Science and Systems*.

Hollinger, G., U. Mitra, and G. Sukhatme (2011). “Active classification: Theory and application to underwater inspection”. In: *Proc. Int. Symp. on Robotics Research*. Springer.

Hollinger, G. et al. (2012). “Uncertainty-driven view planning for underwater inspection”. In: *Proc. IEEE Int. Conf. on Robotics & Automation*.

Hollinger, G. et al. (2013). “Active planning for underwater inspection and the benefit of adaptivity”. In: *Int. J. Robotics Research* 32.1, pp. 3–18.

Hsiao, K. (2009). “Relatively robust grasping”. PhD thesis. Citeseer.

Hsu, D., W. S. Lee, and N. Rong (2007). “What makes some POMDP problems easy to approximate”. In: *Advances in Neural Information Processing Systems (NIPS)*, p. 28.

Javdani, S., M. Klingensmith, et al. (2013). “Efficient touch based localization through submodularity”. In: *Proc. IEEE Int. Conf. on Robotics & Automation*.

Javdani, S., Y. Chen, et al. (2014). “Near Optimal Bayesian Active Learning for Decision Making”. In: *Proc. Int. Conf. on Artificial Intelligence and Statistics*.

Kaelbling, L. P., M. L. Littman, and A. R. Cassandra (1998). “Planning and acting in partially observable stochastic domains”. In: *Artificial Intelligence* 101, pp. 99–134.

Krause, A. and C. Guestrin (2009). “Optimal Value of Information in Graphical Models”. In: *J. Artificial Intelligence Research* 35.1, pp. 557–591.

Kurniawati, H., Y. Du, et al. (2010). “Motion planning under uncertainty for robotic tasks with long time horizons”. In: *Int. J. Robotics Research* 30.3, pp. 308–323.

Kurniawati, H., D. Hsu, and W. S. Lee (2008). “SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces”. In: *Proc. Robotics: Science and Systems*.

Low, K. H., J. M. Dolan, and P. K. Khosla (2009). “Information-Theoretic Approach to Efficient Adaptive Path Planning for Mobile Robotic Environmental Sensing.” In: *ICAPS*.

Monahan, G. E. (1982). “A survey of partially observable Markov decision processes: Theory, models, and algorithms”. In: *Management Science*, pp. 1–16.

Nemhauser, G. L., L. A. Wolsey, and M. L. Fisher (1978). “An analysis of approximations for maximizing submodular set functions—I”. In: *Mathematical Programming* 14.1, pp. 265–294.

Ng, A. Y. et al. (2003). “Autonomous helicopter flight via Reinforcement Learning”. In: *Advances in Neural Information Processing Systems (NIPS)*.

Papadimitriou, C.H and Tsitsiklis, J.N. (1987). “The complexity of Markov decision processes”. In: *Mathematics of operations research*, pp. 441–450.

Pineau, J., G. Gordon, and S. Thrun (2003). “Point-based value iteration: An anytime algorithm for POMDPs”. In: *Proc. Int. Jnt. Conf. on Artificial Intelligence*, pp. 477–484.

Pineau, J., N. Roy, and S. Thrun (2001). “A hierarchical approach to POMDP planning and execution”. In: *Workshop on Hierarchy & Memory in Reinforcement Learning (ICML)*. Vol. 156.

Platt Jr, R. et al. (2011). “Simultaneous Localization and Grasping as a Belief Space Control Problem”. In: *Proc. Int. Symp. on Robotics Research*.

Poupart, P. and C. Boutilier (2003). “Bounded finite state controllers”. In: *Advances in neural information processing systems* 16, pp. 823–830.

Poupart, P., K. E. Kim, and D. Kim (2011). “Closing the Gap: Improved Bounds on Optimal POMDP Solutions”. In: *Twenty-First International Conference on Automated Planning and Scheduling*.

Rasmussen, C. E. (2006). *Gaussian processes for machine learning*. MIT Press.

Ross, S. et al. (2008). “Online planning algorithms for POMDPs”. In: *Journal of Artificial Intelligence Research* 32.1, pp. 663–704.

Shani, G., R. I. Brafman, and S. E. Shimony (2007). “Forward search value iteration for POMDPs”. In: *Proc. Int. Jnt. Conf. on Artificial Intelligence*.

Shani, G., J. Pineau, and R. Kaplow (2012). “A survey of point-based POMDP solvers”. In: *Autonomous Agents and Multi-Agent Systems*.

Singh, A., A. Krause, C. Guestrin, et al. (2009). “Efficient informative sensing using multiple robots”. In: *J. Artificial Intelligence Research* 34.2, pp. 707–755.

Singh, A., A. Krause, and W. Kaiser (2009). “Nonmyopic Adaptive Informative Path Planning for Multiple Robots.” In: *Proc. Int. Jnt. Conf. on Artificial Intelligence*.

Sivakumar, A. and C. Tan (2010). “UAV swarm coordination using cooperative control for establishing a wireless communications backbone”. In: *Proc. Int. Conf. on Autonomous Agents & Multiagent Systems*, pp. 1157–1164.

Smallwood, R. D. and E. J. Sondik (1973). “The optimal control of partially observable Markov processes over a finite horizon”. In: *Operations Research* 21.5, pp. 1071–1088.

Smith, T. and R. Simmons (2004). “Heuristic search value iteration for POMDPs”. In: *Proc. Conf. on Uncertainty in Artificial Intelligence*. AUAI Press, pp. 520–527.

— (2005). “Point-Based POMDP Algorithms: Improved Analysis and Implementation”. In: *Proc. Uncertainty in Artificial Intelligence*.

Spaan, M. T. and N. Vlassis (2005). “Perseus: Randomized point-based value iteration for POMDPs”. In: *Journal of Artificial Intelligence Research* 24.1, pp. 195–220.

Sutton, R., D. Precup, and S. Singh (1999). “Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning”. In: *Artificial Intelligence* 112.1, pp. 181–211.

Theocharous, G. and L. P. Kaelbling (2003). “Approximate planning in POMDPs with macro-actions”. In: *Advances in Neural Information Processing Systems (NIPS)*. Vol. 17.

Thrun, S. (2000). “Monte carlo pomdps”. In: *Advances in neural information processing systems* 12, pp. 1064–1070.

Tong, S. and D. Koller (2002). “Support Vector Machine Active Learning with Applications to Text Classification”. In: *J. Mach. Learn. Res.* 2, pp. 45–66. ISSN: 1532-4435. DOI: 10.1162/153244302760185243.

White, C. (1976). “Procedures for the solution of a finite-horizon, partially observed, semi-Markov optimization problem”. In: *Operations Research* 24.2, pp. 348–358.

Williams, J. D. and S. Young (2007). “Partially observable Markov decision processes for spoken dialog systems”. In: *Computer Speech & Language* 21.2, pp. 393–422.

Wolsey, L. A. (1982). “An analysis of the greedy algorithm for the submodular set covering problem”. In: *Combinatorica* 2.4, pp. 385–393.

Zheng, A. X., I. Rish, and A. Beygelzimer (2005). “Efficient Test Selection in Active Diagnosis via Entropy Approximation”. In: *Proc. Uncertainty in Artificial Intelligence*.

Appendix A

Proofs

A.1 Adaptive Stochastic Optimization

A.1.1 Proofs for examples of adaptive stochastic optimization problem

Proposition 1 (Version Space Reduction)**.** *The version space function $\mathcal{V}$ satisfies marginal likelihood rate bound.*

Proof. We need to show

$$Q - \min_{\phi' \sim \psi'} \mathcal{V}(\text{dom}(\psi'), \phi') \leq 0.5 \left(Q - \min_{\phi \sim \psi} (\mathcal{V}(\text{dom}(\psi), \phi)) \right), \tag{A.1}$$

for any pair of history ψ', ψ such that $\psi' \sim \psi$ and $p(\psi') \leq 0.5p(\psi)$. The relationship becomes obvious when we observe that Equation (4.2) can be written as $\mathcal{V}(S, \phi) = 1 - \sum_{\phi' \sim \phi(S)} \rho(\phi') = 1 - p(\psi)$, for all $\phi \sim \psi$ and choosing $Q = 1$. Hence,

$$\begin{aligned} LHS &= 1 - \min_{\phi' \sim \psi'} \big(1 - p(\psi')\big) \\ &= p(\psi') \\ &\leq 0.5p(\psi) \\ &= RHS \end{aligned}$$

□

Proposition 3. *The generalized version space reduction function f_L satisfies marginal likelihood bound with constant $G = \max_{\phi,\phi'} L(\phi, \phi')$.*

Proof. The generalized version space reduction can be written as:

$$f_L(S,\phi) = \sum_{\phi'} \rho(\phi')L(\phi,\phi') - \sum_{\phi' \sim \phi(S)} \rho(\phi')L(\phi,\phi').$$

We also have

$$f_L(X,\phi) = \sum_{\phi'} \rho(\phi')L(\phi,\phi')$$

Let $G = \max_{\phi,\phi'} L(\phi,\phi')$. For any history ψ,

$$\begin{aligned} f_L(X,\phi) - f_L(\text{dom}(\psi),\phi) &= \sum_{\phi' \sim \phi(\text{dom}(\psi))} \rho(\phi')L(\phi,\phi') \\ &\leq \sum_{\phi' \sim \phi(\text{dom}(\psi))} p(\phi') \cdot G \\ &\leq G \cdot p(\psi) \end{aligned}$$

and hence satisfies condition of marginal likelihood bound with constant $G = \max_{\phi,\phi'} L(\phi,\phi')$.

□

Proposition 6. *The Gibbs error function f_{GE} is pointwise submodular and monotone. In addition, it satisfies condition marginal likelihood rate bound with constants $Q = W(\mathcal{E}) = 1 - \sum_{i=1}^{m}(p(\mathcal{H}_i))^2$, the total weight of ambiguous pairs of hypotheses, and $K = 2$.*

Proof. First, we show f_{GE} is pointwise submodular and monotone. For a fixed hypothesis $h \in H'$, the function f_{GE} is monotone because it is the total weight of disambiguated pairs of hypotheses and the weight of a pair of hypotheses is nonnegative.

For a fixed hypothesis $h \in H'$, sets of location A, B, a location $y \notin B$, and $A \subseteq B$,

$$\begin{aligned} f_{GE}(A \cup \{y\}, h) - f_{GE}(A,h) &= W(\cup_{x \in A}\mathcal{E}_x(h) \cup \mathcal{E}_y(h)) - W(\cup_{x \in A}\mathcal{E}_x(h)) \\ &= W(\mathcal{E}_y(h) \setminus \cup_{x \in A}\mathcal{E}_x(h)) \\ &\geq W(\mathcal{E}_y(h) \setminus \cup_{x \in B}\mathcal{E}_x(h)) \\ &= f_{GE}(B \cup \{y\}, h) - f_{GE}(B,h) \end{aligned}$$

Hence f_{GE} is submodular.

Now, we note that $Q - f_{GE}(\text{dom}(\psi), h) = p(\psi)^2 - \sum_i p(\psi, \mathcal{H}_i)^2$. Given $p(\psi)$, the largest value for $\sum_i p(\psi, \mathcal{H}_i)^2$ occurs when there are only two equal valued probabilities $p(\psi, \mathcal{H}_1) = p(\psi, \mathcal{H}_2) = p(\psi)/2$ giving the value of $\sum_i p(\psi, \mathcal{H}_i)^2 = p(\psi)^2/2$ and $Q - f_{GE}(\text{dom}(\psi), h) \geq p(\psi)^2/2$. When $p(\psi') \leq p(\psi)/2$, we have $p(\psi')^2 \leq p(\psi)^2/4$ and $Q - f_{GE}(\text{dom}(\psi'), h) \leq p(\psi)^2/4$. Hence $Q - f_{GE}(\text{dom}(\psi'), h) \leq p(\psi)^2/4 \leq (Q - f_{GE}(\text{dom}(\psi), h))/2$ giving $K = 2$. □

Proposition 2. *Adaptive monotonicity and submodularity does not imply the marginal likelihood rate bound. Furthermore, the marginal likelihood rate bound does not imply adaptive monotonicity and submodularity.*

Proof. We prove the proposition using two counter examples.

Example 1. *Consider an adaptive stochastic optimization problem with two items $X = \{a, b\}$ and two observations $O = \{0, 1\}$. There are four possible scenarios where both observations are possible at both locations and the prior over them is uniform. The function f is defined such that $f(S, \phi) = |S \cap \{a\}|$ for all scenarios ϕ. This example is trivially adaptive monotone submodular as f does not depend on the scenario.*

However, it is does not satisfy marginal likelihood rate bound. Let history $\psi = \{\}$ and $\psi' = \{(b, 1)\}$. Hence, $p(\psi') \leq 0.5p(\psi)$. But $\hat{f}(dom(\psi), \psi) = \hat{f}(dom(\psi'), \psi') = 0$. Hence, there is no constant fraction $K > 1$ that fulfil Equation (4.1)*.*

Example 2. *Consider an adaptive stochastic optimization problem with two items $X = \{a, b\}$ and two observations $O = \{0, 1\}$, and maximum value $Q = 1$. The prior and function f is defined in Table A.1 This problem is pointwise monotone submodular.*

Table A.1: ρ and f for Example 2

$\rho\ (\phi)$	ϕ	$\{\}$	$\{a\}$	$\{b\}$	$\{a, b\}$
0.6	(a,1) (b,0)	0	1	0	1
0.4	(a,0) (b,0)	0	0.5	1	1

There are two pair of histories where $p(\psi') \leq 0.5p(\psi)$ and they are $\psi' = \{(a, 0)\}, \psi = \{\}$ and $\psi' = \{(a, 0), (b, 0)\}, \psi = \{(b, 0)\}$. For both pair histories, we can verify that they satisfy eq. (4.1) *with upperbound $Q = 1$ and $K = 2$. Hence, this problem satisfies marginal likelihood rate bound. On the other hand, $0.4 = \triangle(b|\{\}) < \triangle(b|\{(a, 0)\}) = 0.5$, it is not adaptive submodular.*

□

We now give the proofs for performance guarantees of RAC. For clarity, we refer to adaptive stochastic optimization problem on paths simply as adaptive stochastic optimization problem. Our proofs hold for both adaptive stochastic optimization problem on paths and on subsets unless we specifically specialize it to subsets at the end.

Proposition 4. *Let f be a pointwise monotone submodular function. Then g_ν is pointwise monotone submodular and g^*_ν is monotone submodular. In addition $g^*_\nu(Z') \geq \nu$ if and only if f is either covered or have value at least ν for all scenarios consistent with $\psi \cup Z'$.*

Proof. First note that the operations of adding a constant to a monotone submodular function, adding together one or more monotone submodular function and setting a ceiling to a monotone submodular function (taking the minimum of a function and a constant) all result in monotone submodular functions. Similarly, if $f_\nu(S, \phi)$ is monotone submodular for X, modifying it by setting $f_\nu(S, \phi) = f_\nu(X, \phi)$ if S contains $x \in X$ preserves monotonicity and submodularity. To see this, note that $f_\nu(X, \phi)$ is the maximum value of the function and setting the function to its maximum later has less gain for a monotone function.

Note that $\min(\nu, g_\nu(Z', \phi))$, $g^*_\nu(Z') \geq \nu$ if and only if $g_\nu(Z', \phi) \geq \nu$ for all ϕ. Finally, note that $g_\nu(Z', \phi) \geq \nu$ exactly when Z' is inconsistent with ϕ, or when it is consistent and $f(\text{dom}(\psi \cup Z'), \phi)$ is covered, or when it is consistent and $f(\text{dom}(\psi \cup Z'), \phi) \geq \nu$ as required.

□

Proposition 5. *When f satisfies minimal dependency, $g^m_\nu(Z') \geq \nu$ implies $g^*_\nu(Z') \geq \nu$.*

Proof. By definition, $g^m_\nu(Z') = g_\nu(Z', Z)$. As f satisfies minimal dependency, g_ν also satisfies minimal dependency. Hence, if $g_\nu(Z', Z) \geq \nu$, we also have $g_\nu(Z', \phi) \geq \nu$ for all ϕ, implying $g^*_\nu(Z') \geq \nu$

□

A.1.2 Adaptive Stochastic Optimization on Paths

We begin by analyzing a variant of adaptive stochastic optimization problem where the agent has to return to the starting location r in the end. We assume that we can compute an optimal submodular orienteering solution, and then relax this assumption to use polynomial time approximation later. This subsection can be divided into three parts. First, we analyze RAC on problems satisfying the marginal likelihood bound condition (Lemma 20 to Lemma 25). Next, we complete the analysis for problems satisfying condition the marginal likelihood rate bound condition (Lemma 26 to Lemma 28). Finally, we relax the assumptions of computing optimal submodular orienteering solution and of going back to the starting location. We derive the final approximation bounds for the non-rooted adaptive stochastic optimization problems satisfying the marginal likelihood bound condition and for those satisfying the marginal likelihood rate bound condition (Lemma 29 to Theorem 11).

The main strategy of this analysis is to establish the post conditions upon termination of the adaptive plan in each recursive step. There are two components to prove in the post conditions; progress made in covering the function and distance traveled by the agent.

In the following (Lemmas 20 and 21), we show that each adaptive plan reduce likelihood of history by half except when it is the last recursive step where it completes the coverage.

Lemma 20. *Let τ be the solution to a submodular orienteering problem g^*_ν in* GENERATETOUR1. *Let ψ be the history experienced by the agent after we call* EXECUTEPLAN *with tour τ. Either $p(\psi) < 0.5$ or $g^*_\nu(\psi) = \nu$.*

Proof. During the execution of EXECUTEPLAN, if the agent receives an observation $o' \in \Omega_x$ at some location x' on τ, then the agent returns to r immediately with history $\psi = ((x_1, o_1), \ldots, (x', o'))$. The probability of this history is $p(\psi) = \prod_{(x,o)\in\psi} p(o|x) \leq p(o'|x')$. From the definition of $\Omega_{x'}$, we have $p(\psi) \leq p(o'|x') < 0.5$.

Otherwise, the agent visits every location x on τ and receives at every x an observation $o^*_x \notin \Omega_x$ and has history $\psi = \psi^*(\tau)$, *i.e.* the agent always receive the most likely observation throughout the tour and $g^*_\nu(\psi) = \nu$. □

Lemma 21. *Let ψ be the history after a recursive call of RAC. After each recursive call, either likelihood of history is reduced by half, $p(\psi) < 0.5$ or we have completely covered the function f.*

Proof. RAC calls EXECUTEPLAN with either τ_f or τ_{vs}, which solves the submodular orienteering problem g_Q^* and $\mathcal{V}_{0.5}^*$ respectively. If RAC uses τ_f, Lemma 20 tells us that EXECUTEPLAN either reduces the likelihood of history by at least half or completely covers the function g_Q^*, which implies that we have completely covered the function f.

Otherwise, RAC uses τ_{vs} and reduces the version space (and equivalently $p(\psi)$) by at least a half.

Finally, we prove the lemma by combining the outcomes from using τ_f or τ_{vs}. □

We want to bound the distance traveled in each recursive call by comparing the length of the submodular orienteering tour to a path in the optimal policy. This path always exist and is traversed with probability more than half by the optimal policy. Hence, we can bound the length of our tour by twice the expected cost of optimal policy.

Lemma 22. *Let π^* be an optimal policy tree for a rooted adaptive stochastic optimization problem $\mathcal{I}$. There is a subpath σ' of π^* such that π^* traverses σ' with probability at least 0.5. Furthermore, one of the following conditions must hold: (1) the probability of most likely history on this path $p(\psi^*(\sigma')) \geq 0.5$ and $\psi^*(\sigma')$ covers f, or (2) $p(\psi^*(\sigma')) < 0.5$ and $p(\psi^*(\sigma'_{-1})) \geq 0.5$, where $\psi^*(\sigma'_{-1})$ is the most likely history without the final observation.*

Proof. We give the construction for such a subpath σ'. First, we extracts a path σ from an optimal policy π^* tree by following the most likely observation edge from the root. Let $\sigma = (r, x_1, x_2, \ldots, x_s, r)$ be a path in the optimal policy tree π^* such that every edge following a node x_i in the path is labeled with the most likely observation $o_{x_i}^* = \arg\max_{o \in O} p(o|x)$ up to the last node x_s and then return to the root r. Thus, the history from traversing σ is $\psi^*(\sigma)$.

Next, we need to ensure that π^* traverses its subpath σ' with probability at least 0.5. Let $p(\sigma_i|\pi^*)$ be the probability of reaching the node x_i on the path σ under the optimal policy π^*. It is equal the probability of traversing the path σ and observing the most

likely observation at every location in σ up to x_{i-1} and go on to x_i (without making an observation at x_i) *i.e.*

$$\begin{aligned} p(\sigma_i|\pi^*) &= p((r,(x_1,o^*_{x_1}),\ldots,(x_{i-1},o^*_{x_{i-1}}),x_i)) \\ &= p(\psi^*(\sigma_{i-1})) \end{aligned}$$

If $p(\sigma_s|\pi^*) < 0.5$, we truncate the path σ_s from the end at a location x_q such that $p(\sigma_q|\pi^*) > 0.5$. In other words, σ_q is the longest subpath of σ where $p(\sigma_q|\pi^*) > 0.5$. We set $\sigma' = (\sigma_q, r)$. That is, we return to the root r after traversing σ_q. Otherwise $p(\sigma_s|\pi^*) \geq 0.5$, and we simply set $\sigma' = (\sigma_s, r) = \sigma$.

π^* traverses σ' with probability at least 0.5 by construction. If $\sigma' = \sigma$, it is a complete path along the most likely outcome branch from the root to the leaf of the optimal policy π^*. Thus, $f(\sigma', \phi) = f(X, \phi)$ for all scenarios $\phi \sim \psi^*(\sigma')$.

Otherwise, it is the truncated path $\sigma' = (\sigma_q, r)$. After receiving the most likely observation $o^*_{x_q}$ at x_q, we get $p((r,(x_1,o^*_{x_1}),\ldots,(x_q,o^*_{x_q}))) \leq 0.5$ because σ_q is the longest subpath that is $p(\sigma_q|\pi^*) \geq 0.5$. Thus, $p(\psi^*(\sigma_q)) \leq 0.5$. □

Lemma 23. *Assuming we compute the optimal solution to the submodular orienteering problems, the agent travels at most $2C(\pi^*)$ for each recursive step of RAC.*

Proof. Using Lemma 22, we show that there is a subpath σ' from the optimal policy π^* that is a feasible solution to either the submodular orienteering problem g^*_Q or $\mathcal{V}^*_{0.5}$.

Let σ' a subpath from Lemma 22. If the first case of Lemma 22 is true , then σ' is a feasible solution to the submodular orienteering problem g^*_Q. Otherwise the second case $p(\psi^*(\sigma')) < 0.5$ and $p(\psi^*(\sigma'_{-1})) \geq 0.5$, is true. Then σ' is feasible solution to the problem of $\mathcal{V}^*_{0.5}$ because $\mathcal{V}_{0.5}(\sigma', \phi) = \min(0.5, 1 - p(\psi^*(\sigma')) < 0.5$ for all scenario $\phi \in \Phi_{\sigma'}$.

Let W^*_f and W^*_{vs} be the total edge-weight of optimal submodular orienteering tour τ_f and τ_{vs} respectively. Let the total edge-weight of the tour used in each recursive step be $W^* = \min(W^*_f, W^*_{vs})$. If it is the first case, then $W^* \leq W^*_f \leq W(\sigma')$. Otherwise,

$W^* \leq W^*_{vs} \leq W(\sigma')$. As σ' is traversed with probability at least 0.5,

$$\begin{aligned} C(\pi^*) &\geq \sum_{\phi \sim \psi^*(\sigma)} \rho(\phi) w(\sigma') \\ &\geq 0.5 w(\sigma') \geq 0.5 W^* \\ W^* &\leq 2C(\pi^*), \end{aligned}$$

where $w(\sigma')$ is the total edge-weight of tour σ'.

In EXECUTEPLAN, the agent travels on a path bounded by W^*. Hence, the agent travels at most $2C(\pi^*)$. □

Lemma 24. *Suppose that π^* is an optimal policy for a rooted adaptive stochastic optimization problem $\mathcal{I}$ with prior probability distribution ρ. Let $\{\Phi_1, \Phi_2, \ldots, \Phi_n\}$ be a partition of the scenarios O^X, and let π^*_i be an optimal policy for the subproblem $\mathcal{I}_i$ with prior probability distribution ρ_i:*

$$\rho_i(\phi) = \begin{cases} \rho(\phi)/\rho(\Phi_i) & \text{if } \phi \in \Phi_i \\ 0 & \text{otherwise} \end{cases}$$

where $\rho(\Phi_i) = \sum_{\phi \in \Phi_i} p(\phi)$ Then we have

$$\sum_{i=1}^{n} \rho(\Phi_i) C(\pi^*_i) \leq C(\pi^*).$$

Proof. For each subproblem $\mathcal{I}_i$, we can construct a feasible policy π_i for $\mathcal{I}_i$ from the optimal policy π^* for $\mathcal{I}$. Consider the policy tree π^*. Every scenario ϕ must has a path σ from root to the leaf in the optimal tree π^* that covers the scenario because the optimal policy covers all scenarios. So we choose the policy tree π_i as the subtree of π^* that consists of all the paths that cover scenarios in Φ_i. Clearly π_i is feasible, as every

scenario in Φ_i has a path in π_i that covers it. Then,

$$\begin{aligned}\sum_{i=1}^{n} \rho(\Phi_i) C(\pi_i^*) &\leq \sum_{i=1}^{n} \rho(\Phi_i) C(\pi_i) \\ &\leq \sum_{i=1}^{n} \rho(\Phi_i) \sum_{\phi \in \Phi_i} \frac{\rho(\phi)}{\rho(\Phi_i)} \cdot C(\pi_i, \phi) \\ &= \sum_{\phi \in \Phi_i} \rho(\phi) C(\pi^*, \phi) = C(\pi^*).\end{aligned}$$

□

For functions satisfying the marginal likelihood bound, the remaining objective value to cover is bounded by marginal likelihood of history multiplied by G. Every recursive call either reduces marginal likelihood of history by half or completely covers the function f and thus bounding the remaining function to cover at the same time. The algorithms is repeated at most a logarithmic number of times and we can obtain an approximation bound.

Lemma 25. *Let π denote the policy that RAC computes for a rooted adaptive stochastic optimization problem on paths. Let η be any value such that $f(S, \phi) > f(X, \phi) - \eta$ implies $f(S, \phi) = f(X, \phi)$. If RAC computes an* optimal *submodular coverage tour in each step, then for an instance of adaptive stochastic optimization satisfying marginal likelihood bound*

$$C(\pi) \leq 2 \left(\log(G/\eta) + 1\right) C(\pi^*),$$

where $C(\pi)$ is the expected cost of RAC.

Proof. Let ψ be the entire history experienced by the agent from the start of RAC. If a recursive call picks tour τ_f, traverses the entire tour, and receive most likely observation throughout the tour, then $f(\text{dom}(\psi), \phi) = f(X, \phi)$ for all scenario $\phi \sim \psi$ and we have fully covered f. Otherwise, we repeat the recursive call until $f(X, \phi) - f(\text{dom}(\psi), \phi) < \eta$, for all $\phi \sim \psi$. The marginal likelihood bound condition gives us $f(X, \phi) - f(\text{dom}(\psi), \psi) \leq G \cdot p(\psi)$ for all $\phi \sim \psi$. Hence, we derive from Lemma 21 the number of recursive steps required for any scenario is at most $\log\left(\frac{G}{\eta}\right) + 1$.

We now complete the proof by induction on the number of recursive calls to RAC. For the base case of $k = 1$ call, $C(\pi) \leq 2C(\pi^*)$ by Lemma 23. Assume that $C(\pi) \leq$

$2(k-1)C(\pi^*)$ when there are at most $k-1$ recursive calls. Now consider the induction step of k calls. The first recursive call partitions the scearios into a collection of mutually exclusive subsets, $\Phi_1, \Phi_2, \ldots, \Phi_n$. Let $\mathcal{I}_i$ be the subproblem with scenario set Φ_i and optimal policy π_i^*, for $i = 1, 2, \ldots, n$. After the first recursive call, it takes at most $k-1$ additional calls for each $\mathcal{I}_i$. In the first call, the agent incurs a cost at most $2C(\pi^*)$ by Lemma 23. For each $\mathcal{I}_i$, the agent incurs a cost at most $2(k-1)C(\pi_i^*)$ in the remaining $k-1$ calls, by the induction hypothesis. Putting together this with Lemma 24, we conclude that the agent incurs a total cost of at most $2kC(\pi^*)$ when there are k calls. □

The marginal likelihood rate bound condition (Equation (4.1)) tells us that we reduce the remaining function to cover by a fraction whenever the remaining version space is halved. Next, we show that the remaining function to cover is reduced by a fraction upon termination of each adaptive plan.

Lemma 26. *Let τ be the tour generated in a recursive and ψ be the history after a recursive call of RAC. By the end of each recursive call, for each scenario $\phi \sim \psi$, $f(dom(\psi), \phi) \geq (1 - 1/K)Q$ unless $f(X, \phi) < (1 - 1/K)Q$. In that case, $f(dom(\psi), \phi) = f(X, \phi)$.*

Proof. The procedure EXECUTEPLAN is called with tour τ that is a solution to submodular orienteering problem $g^*_{(1-1/K)Q}$. From Lemma 20, if EXECUTEPLAN terminates with $p(\psi) \leq 0.5$, we know from marginal likelihood rate bound (Equation (4.1)) that $f(\text{dom}(\psi), \phi) \geq (1 - 1/K)Q$ for all $\phi \sim \psi$. Otherwise, EXECUTEPLAN terminates with $g^*_{(1-1/K)Q}(\tau, \psi) = (1 - 1/K)Q$. In that case, from Proposition 4, $f(\text{dom}(\psi), \phi) \geq (1 - 1/K)Q$ or $f(X, \phi) < (1 - 1/K)Q$ and f is already covered for ϕ.

□

Lemma 27. *Assuming we compute the optimal solution to the submodular orienteering problems, the agent travels at most $2C(\pi^*)$ for each recursive step of RAC.*

Proof. From Lemma 22 and marginal likelihood rate bound, the subpath σ' is feasible solution to the submodular orienteering problem of $g^*_{(1-1/K)Q}$. Let W^* be the total

edge-weight of the tour used in a recursive call of RAC. Then, $W^* \leq W(\sigma')$ because W^* is the value of an optimal solution. Since σ' is traversed with probability at least 0.5,

$$\begin{aligned} C(\pi^*) &\geq \sum_{\phi \sim \psi^*(\sigma)} \rho(\phi) w(\sigma') \\ &\geq 0.5 w(\sigma') \geq 0.5 W^* \\ W^* &\leq 2C(\pi^*), \end{aligned}$$

where $w(\sigma')$ is the total edge-weight of tour σ'.

In EXECUTEPLAN, the agent travels on a path bounded by W^*. Hence, the agent travels at most $2C(\pi^*)$. □

Lemma 28. *Let π denote the policy that RAC computes for a rooted adaptive stochastic optimization problem on paths. Let η be any value such that $f(S, \phi) > f(X, \phi) - \eta$ implies $f(S, \phi) = f(X, \phi)$. If RAC computes an* optimal *submodular coverage tour in each step, then for an instance of adaptive stochastic optimization satisfying marginal likelihood rate bound*

$$C(\pi) \leq 2 \left(\log_K(Q/\eta) + 1\right) C(\pi^*),$$

where $C(\pi)$ is the expected cost of RAC, and $K > 1$ and $Q \geq \max_\phi f(X, \phi)$ are the constants that satisfy Equation (4.1)*.*

Proof. We need to repeat the recursive call until $f(X, \phi) - f(\text{dom}(\psi), \phi) \leq \eta$ for all $\phi \sim \psi$. From marginal likelihood rate bound and Lemma 26, the number of recursive steps required for any scenario is at most $\log_K \left(\frac{Q}{\eta}\right) + 1$.

We now complete the proof by induction on the number of recursive calls to RAC. For the base case of $k = 1$ call, $C(\pi) \leq 2C(\pi^*)$ by Lemma 27. Assume that $C(\pi) \leq 2(k-1)C(\pi^*)$ when there are at most $k-1$ recursive calls. Now consider the induction step of k calls. The first recursive call partitions the scenarios into a collection of mutually exclusive subsets, $\Phi_1, \Phi_2, \ldots, \Phi_n$. Let $\mathcal{I}_i$ be the subproblem with scenario set Φ_i and optimal policy π_i^*, for $i = 1, 2, \ldots, n$. After the first recursive call, it takes at most $k-1$ additional calls for each $\mathcal{I}_i$. In the first call, the agent incurs a cost at

most $2C(\pi^*)$ by Lemma 27. For each $\mathcal{I}_i$, the agent incurs a cost at most $2(k-1)C(\pi_i^*)$ in the remaining $k-1$ calls, by the induction hypothesis. Putting together this with Lemma 24, we conclude that the agent incurs a total cost of at most $2kC(\pi^*)$ when there are k calls. Hence, we obtain our approximation bounds. □

Now, we relax the optimal submodular orienteering assumption and replace it with our polynomial time approximation procedure.

Lemma 29. *An α-approximation algorithm for rooted adaptive stochastic optimization problem on paths is a 2α-approximation algorithm for adaptive stochastic optimization.*

Proof. Let C^* and C_{r}^* be the expected cost of an optimal policy for an adaptive stochastic optimization problem and for a corresponding rooted adaptive stochastic optimization problem, respectively. As any policy for non-rooted problem can be turned into a policy for the root version by retracing the solution path back to the start location, we have $C_{\mathrm{r}}^* \leq 2C^*$. An α-approximation algorithm for rooted adaptive stochastic optimization computes a policy π for $\mathcal{I}_{\mathrm{r}}$ with expected cost $C_{\mathrm{r}}(\pi) \leq \alpha C_{\mathrm{r}}^*$. It then follows that $C_{\mathrm{r}}(\pi) \leq \alpha C_{\mathrm{r}}^* \leq 2\alpha C^*$ and this algorithm provides a 2α-approximation to the optimal solution of the non-rooted problem. □

Theorem 11. *Assume that f is a pointwise integer-valued submodular monotone function. Let η be any value such that $f(S,\phi) > f(X,\phi) - \eta$ implies $f(S,\phi) = f(X,\phi)$ for all $S \subseteq X$ and all scenario ϕ. For any constant $\epsilon > 0$ and an instance of adaptive stochastic optimization problem on path satisfying marginal likelihood rate bound, RAC computes a policy π in polynomial time such that*

$$C(\pi) = O((\log|X|)^{2+\epsilon} \log Q \log_K(Q/\eta))C(\pi^*)),$$

where Q and $K > 1$ are constants that satisfies Equation (4.1)*.*

Proof. The distance traveled in each recursive step is at most $\alpha W^* \leq O(\alpha)C(\pi^*)$. From Lemma 2 , the approximation factor for the submodular orienteering problem solved in RAC is
$\alpha = O((\log|X|)^{2+\epsilon} \log Q)$. Putting this together with Lemma 28 and Lemma 29,

we get the desired approximation bound. The algorithm clearly runs in polynomial time. □

Theorem 12. *Assume that the prior probability distribution ρ is represented as non-negative integers with $\sum_\phi \rho(\phi) = P$. Let η be any value such that $f(S,\phi) > f(X,\phi) - \eta$ implies $f(S,\phi) = f(X,\phi)$ for all $S \subseteq X$ and all scenario ϕ. Assume that f is a pointwise integer-valued submodular monotone function. For any constant $\epsilon > 0$ and an instance of adaptive stochastic optimization problem on path satisfying marginal likelihood bound, RAC computes a policy π for in polynomial time such that*

$$C(\pi) = O((\log|X|)^{2+\epsilon}(\log P + \log Q)\log(G/\eta))C(\pi^*),$$

where $Q = \max_\phi f(X,\phi)$.

Proof. Let α_1 and α_2 be the approximation factors when we compute the submodular orienteering tours τ_f and τ_{VS} respectively in one recursive call of RAC. Let the length of the tour chosen be W, Let the length of the tour chosen be W,

$$\begin{aligned} W &= \min(\alpha_1 W_f^*, \alpha_2 W_{VS}^*) \\ &\leq (\alpha_1 + \alpha_2) W^* \\ &\leq 2(\alpha_f + \alpha_{VS}) C(\pi^*) \end{aligned}$$

The last inequality is due to Lemma 23. Hence, the distance traveled in each recursive step is at most $2(\alpha_f + \alpha_{VS})C(\pi^*)$. Lemma 2 tells us that $\alpha_1 \in O((\log|X|)^{2+\epsilon}\log Q)$ and $\alpha_2 \in O((\log|X|)^{2+\epsilon}\log P)$. Putting this together with Lemma 25 and Lemma 29, we get the desired approximation bound. The algorithm clearly runs in polynomial time. □

A.1.3 Adaptive Stochastic Optimization on Sets

Adaptive stochastic minimum cost cover on sets (without path constraints) is a special case where the metric is a star graph where all elements are connected to a root node. In the special case of sets, the submodular orienteering problems that RAC solves become submodular set coverage problems. At the same time, the submodular orienteering pro-

cedure in RAC becomes a greedy selection policy where we always choose the element with highest value to cost ratio, *i.e.*$\max_{x \in X \setminus \text{dom}(\psi)} \frac{\triangle(x|\psi)}{c(x)}$.

Lemma 30. *Given a submodular set function $g : X \to \mathbb{R}$, let π^G be the greedy selection policy. We have,*

$$C(\pi^G) \leq \left(1 + \ln \frac{f(X) - f(\emptyset)}{f(X) - f(S^{T-1})}\right) C(\pi^*)$$

where the subset S^{T-1} is the set of elements selected before the last step of the greedy policy (Wolsey, 1982).

Using Lemma 30, we can get tighter approximation bounds for stochastic sets functions and drop the integer representation assumption on the prior ρ.

Theorem 13. *For an instance of adaptive stochastic optimization problem on subsets satisfying marginal likelihood rate bound, assuming f is pointwise integer-valued submodular and monotone, let η be any value such that $f(S, \phi) > f(X, \phi) - \eta$ implies $f(S, \phi) = f(X, \phi)$ for all $S \subseteq X$ and all scenario ϕ. RAC computes a policy π in polynomial time such that*

$$C(\pi) = 4(\ln Q + 1)(\log_K(Q/\eta) + 1)C(\pi^*),$$

where Q and $K > 1$ are constants that satisfies Equation (4.1)*.*

Proof. The distance traveled in each recursive step is at most $\alpha W^* \leq 4\alpha C(\pi^*)$. From Lemma 30, the approximation factor for the submodular set cover problem solved in RAC is $\alpha = \log Q$. Putting this together with Lemma 28 and Lemma 29, we get the desired approximation bound. The algorithm clearly runs in polynomial time. □

Theorem 14. *For an instance of adaptive stochastic optimization problem on subsets satisfying the marginal likelihood bound condition, assuming f is pointwise integer-valued submodular and monotone, let η be any value such that $f(S, \phi) > f(X, \phi) - \eta$ implies $f(S, \phi) = f(X, \phi)$ for all $S \subseteq X$ and all scenario ϕ and $\delta = \min_\phi \rho(\phi)$. RAC computes a policy π in polynomial time such that*

$$C(\pi) = 4(\ln 1/\delta + \ln Q + 2)(\log(G/\eta) + 1)C(\pi^*)),$$

where $Q = \max_\phi f(X, \phi)$.

Proof. Let α_1, α_2 be the approximation factors when we compute the submodular set cover τ_f and τ_{VS} respectively. Let the cost of the set of elements chosen be W,

$$\begin{aligned} W &= \min(\alpha_1 W_f^*, \alpha_2 W_{VS}^*) \\ &\leq (\alpha_1 + \alpha_2) W^* \\ &\leq 2(\alpha_f + \alpha_{VS}) C(\pi^*) \end{aligned}$$

The last inequality is due to Lemma 23. Hence, the distance traveled in each recursive step is at most $4(\alpha_f + \alpha_{VS})C(\pi^*)$. From Lemma 30, the approximation factors for the submodular set cover problems are $\alpha_1 = \ln 1/\delta + 1$ and $\alpha_2 = \ln Q + 1$. Putting this together with Lemma 25 and Lemma 29, we get the desired approximation bound. The algorithm clearly runs in polynomial time. □

A.2 POMDP with Macro Actions

Lemma 15 (Contraction)**.** *Given value functions U and V, $||HU - HV||_\infty \leq \gamma ||U - V||_\infty$.*

Proof. Let b be an arbitrary belief and assume that $HV(b) \leq HU(b)$ holds. Let $\mathbf{a}^*$ be the optimal macro action for $HU(b)$. Then

$$\begin{aligned} 0 &\leq HU(b) - HV(b) \\ &\leq \mathbf{R}(b, \mathbf{a}^*) + \gamma \sum_{\mathbf{o} \in \mathcal{O}} p_\gamma(\mathbf{o}|\mathbf{a}^*, b) U(\tau(b, \mathbf{o}, \mathbf{a}^*)) - \mathbf{R}(b, \mathbf{a}^*) - \gamma \sum_{\mathbf{o} \in \mathcal{O}} p_\gamma(\mathbf{o}|\mathbf{a}^*, b) V(\tau(b, \mathbf{o}, \mathbf{a}^*)) \\ &= \gamma \sum_{\mathbf{o} \in \mathcal{O}} p_\gamma(\mathbf{o}|\mathbf{a}^*, b) [U(\tau(b, o, \mathbf{a}^*) - V(\tau(b, o, \mathbf{a}^*))] \\ &\leq \gamma \sum_{\mathbf{o} \in \mathcal{O}} p_\gamma(\mathbf{o}|\mathbf{a}^*, b) ||U - V||_\infty \\ &\leq \gamma ||U - V||_\infty. \end{aligned}$$

Since $||\cdot||_\infty$ is symmetrical, the result is the same for the case of $HU(b) \leq HV(b)$.

By taking $||\cdot||_\infty$ over all weighted belief, we get

$$||HU - HV||_\infty \leq \gamma||U - V||_\infty.$$

Thus, H is a contractive mapping. □

Theorem 17 (Piecewise Linearity and Convex)**.** *The value function for an m-step policy is piecewise linear and convex and can be represented as*

$$V_m(b) = \max_{\alpha \in \Gamma_m} \sum_{s \in S} \alpha(s)b(s) \tag{5.5}$$

where Γ_m is a finite collection of α-vectors.

Proof. We prove this property by induction. When $m = 1$, the initial value function V_1 is the best expected reward and can be written as

$$V_1(b) = \max_{\mathbf{a}} \mathbf{R}(b, \mathbf{a}) = \max_{\mathbf{a}} \sum_{s \in S} \mathbf{R}(s, \mathbf{a})b(s).$$

This has the same form as $V_m(b) = \max_{\alpha_m \in \Gamma_m} \sum_{s \in S} \alpha_m(s)b(s)$ where there is one linear α-vector for each macro action. $V_1(b)$ can therefore be represented as a finite collection of α-vectors.

Assuming the optimal value function for any b_{i-1} is represented using a finite set of α-vector $\Gamma_{i-1} = \{\alpha_{i-1}^0, \alpha_{i-1}^1, ...\}$ and

$$V_{i-1}(b_{i-1}) = \max_{\alpha_{i-1} \in \Gamma_{i-1}} \sum_{s \in S} b_{i-1}(s)\alpha_{i-1}(s) \tag{A.2}$$

Substituting

$$b_{i-1}(s) = \sum_{j=1}^{\infty} \gamma^{j-1} \sum_{s'} p(s, \mathbf{o}, j|s', \mathbf{a})b_i(s')/p_\gamma(\mathbf{o}|\mathbf{a}, b_i)$$

into (A.2), we get

$$V_{i-1}(b_{i-1}) = \max_{\alpha_{i-1} \in \Gamma_{i-1}} \sum_{s \in S} \frac{\sum_{j=1}^{\infty} \gamma^{j-1} \sum_{s'} p(s, \mathbf{o}, j|s', \mathbf{a})b_i(s')}{p_\gamma(\mathbf{o}|\mathbf{a}, b_i)} \alpha_{i-1}(s).$$

Substituting it into the backup equation gives

$$
\begin{aligned}
V_i(b_i) &= \max_{\mathbf{a}}\big(\mathbf{R}(b_i,\mathbf{a}) + \gamma \sum_{\mathbf{o}\in\mathcal{O}} p_\gamma(\mathbf{o}|\mathbf{a},b_i) \max_{\alpha_{i-1}\in\Gamma_{i-1}} \sum_{s\in S} \frac{\sum_{j=1}^{\infty}\gamma^{j-1}\sum_{s'} p(s,\mathbf{o},j|s',\mathbf{a})b_i(s')}{p_\gamma(\mathbf{o}|\mathbf{a},b_i)}\alpha_{i-1}(s)\big) \\
&= \max_{\mathbf{a}}\big(\mathbf{R}(b_i,\mathbf{a}) + \gamma \sum_{\mathbf{o}\in\mathcal{O}} \max_{\alpha_{i-1}\in\Gamma_{i-1}} \sum_{s\in S}\sum_{j=1}^{\infty}\gamma^{j-1}\sum_{s'} p(s,\mathbf{o},j|s',\mathbf{a})b_i(s')\alpha_{i-1}(s)\big) \\
&= \max_{\mathbf{a}} \max_{\alpha_{i-1}^1\in\Gamma_{i-1},\ldots,\alpha_{i-1}^{|\mathcal{O}|}} \sum_{s'\in S} b_i(s') \left[\mathbf{R}(s',\mathbf{a}) + \gamma \sum_{\mathbf{o}\in\mathcal{O}}\sum_{s\in S}\sum_{j=1}^{\infty}\gamma^{j-1} p(s,\mathbf{o},j|s',\mathbf{a})\alpha_{i-1}^{\mathbf{o}}(s)\right]
\end{aligned}
$$

The expression in the square bracket can evaluate to $|\mathcal{A}||\Gamma_{i-1}|^{|\mathcal{O}|}$ different vectors.

We can rewrite $V_i(b_i)$ as:

$$V_i(b_i) = \max_{\alpha_i\in\Gamma_i} \sum_{s\in S} \alpha_i(s)b_i(s).$$

Hence $V_i(b_i)$ can be represented by a finite set of α-vector. □