Semi-conditional Planners for Efficient Planning under Uncertainty with Macro-actions

by

Ruijie He

B.S. Massachusetts Institute of Technology (2007)
M.S. Massachusetts Institute of Technology (2008)

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

Doctor of Philosophy in Aeronautics and Astronautics

at the

Massachusetts Institute of Technology

June 2010

Author .....
Department of Aeronautics and Astronautics
May 20, 2010

Certified by .....
Nicholas Roy
Associate Professor of Aeronautics and Astronautics
Thesis Supervisor

Certified by .....
Emilio Frazzoli
Associate Professor of Aeronautics and Astronautics
Thesis Committee

Certified by .....
Jonathan How
Professor of Aeronautics and Astronautics
Thesis Committee

Certified by .....
David Hsu
Associate Professor of Computer Science
National University of Singapore
Thesis Committee

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

Semi-conditional Planners for Efficient Planning under Uncertainty with Macro-actions

by

Ruijie He

Submitted to the Department of Aeronautics and Astronautics
on May 20, 2010, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Aeronautics and Astronautics

Abstract

Planning in large, partially observable domains is challenging, especially when good performance requires considering situations far in the future. Existing planners typically construct a policy by performing fully-conditional planning, where each future action is conditioned on a set of possible observations that could be obtained at every timestep. Unfortunately, fully-conditional planning can be computationally expensive, and state-of-the-art solvers are either limited in the size of problems that can be solved, or can only plan out to a limited horizon.

We propose that for a large class of real-world, planning under uncertainty problems, it is necessary to perform far-lookahead decision-making, but unnecessary to construct policies that condition all actions on observations obtained at the previous timestep. Instead, these problems can be solved by performing *semi-conditional* planning, where the constructed policy only conditions actions on observations at certain key points. Between these key points, the policy assumes that a macro-action — a temporally-extended, fixed-length, open-loop action sequence, comprising a series of primitive actions, is executed. These macro-actions are evaluated within a forward-search framework, which only considers beliefs that are reachable from the agent's current belief under different actions and observations; a belief summarizes an agent's past history of actions and observations. Together, semi-conditional planning in a forward search manner restricts the policy space in exchange for conditional planning out to a longer-horizon.

Two technical challenges have to be overcome in order to perform semi-conditional planning efficiently — how the macro-actions can be automatically generated, as well as how to efficiently incorporate the macro-action into the forward search framework. We propose an algorithm which automatically constructs the macro-actions that are evaluated within a forward search planning framework, iteratively refining the macro-actions as more computation time is made available for planning. In addition, we show that for a subset of problem domains, it is possible to analytically compute the distribution over posterior beliefs that result from a single macro-action. This ability to directly compute a distribution over posterior beliefs enables us to enjoy computational savings when performing macro-action forward search.

Performance and computational analysis for the algorithms proposed in this thesis are presented, as well as simulation experiments that demonstrate superior performance relative to existing state-of-the-art solvers on large planning under uncertainty domains. We also demonstrate our planning under uncertainty algorithms on target-tracking applications for an actual autonomous helicopter, highlighting the practical potential for planning in real-world, long-horizon, partially observable domains.

Thesis Supervisor: Nicholas Roy
Title: Associate Professor of Aeronautics and Astronautics

Acknowledgements

Stepping through the doors of MIT's building 33 as an undergraduate for the first time, never would I have imagined having the opportunity to write these paragraphs today. It has been quite an experience, and like the soldier that stands ready on the battlefield, this journey could not have been trodden alone. Many people deserve much more than a word of thanks.

My advisor Nick Roy not only offered a UROP to this untested undergraduate, but also made the incredulous suggestion that I stay beyond my Master's. Thanks Nick for guiding me through this accelerated, often ridiculous adventure, and for never failing to leave me dumb-struck every time I step out of the office. Your open-door policy also enabled us to bother you with countless impractical ideas and outrageous requests; hopefully now the lab will be just that bit more peaceful.

Emilio Frazzoli, Jon How and David Hsu make up the rest of my thesis committee, providing me invaluable feedback that has been instrumental to the thesis. I will fondly remember Emilio's ability to frame and contextualize my latest ideas, Jon's efforts to force me to crystallize and sharpen my research arguments, as well as David's willingness to share his latest theoretical insights, algorithms and research toolkits. Thanks also to my two thesis readers, Leslie Kaelbling and Tomas Lorenzo-Perez, for insightful comments and perspectives that have been grounded in years of experience.

The MAV team has been the most enjoyable bunch of colleagues I could ever have imagined doing research with. Abe Bachrach, Sam Prentice and I ostensibly traveled the world for research, but also spent countless late nights and early mornings crashing, and repairing, helicopters in Stata. Never was research dull around them. Thanks also to Adam Bry, Anton de Winter and Garrett Hemann for their technical assistance, be it with helicopter hardware, video editing or safety piloting. Finally, the team from Ascending Technologies, especially Daniel Gurdan, Jan Stumpf, and the Achtelik brothers, provided invaluable hardware support and snorkeling gear.

The Robust Robotics Group has been a great source of research ideas and intellectual stimulation over the years. Special thanks to Emma Brunskill for numerous productive collaborations, especially given that most of the discussions had to be done remotely. Among other things, her finesse with language has been critical for articulating many of the research ideas presented here. In addition, Finale Doshi, Alborz Geramifard, Albert Huang, Josh Joseph, Tom Kollar, Matt Walter, and Javier Velez not only patiently sat through countless practice presentations over the years, but have also been amazing bouncing boards for my latest research ideas.

Outside of the group, I had the fortune of interacting with some of the most talented graduate students; they have regularly inspired me with their creativity and passion. I am thankful to colleagues such as Brett Bethke, Han-Lim Choi, Rick Cory, Marek Donic, Brandon Luders, Sertac Karaman, Sho Sato, Stephen Smith, Aditya Urdunti and many others for insightful discussions, invaluable friendship and gym companionship.

As challenging as it was, MIT would have been impossible without the dedicated assistance of the ad-

ministrative and support staff with MIT's Aero/Astro department. My heartfelt gratitude to Barbara Lechner, Marie Stuppard and Beth Morris from the Aero/Astro Student Services office for their expertise in MIT's administrative policies and their assistance with countless petitions over the years. This adventure would have ended pre-maturely had it not been for their unwavering support. Thanks also to Brian O'Conaill, Kathryn Fischer, Edward Gazarian, Alison Pita and Bryt Bradley for their administrative assistance.

I am indebted to the Singapore Armed Forces not only for sponsoring my undergraduate and Master's education, but also for making the exception and granting me extra time to complete this degree. Special thanks to COL Tan Chong Lee, COL Lim Tuang Liang, LTC Frederick Teo, LTC Lawrence Lim, LTC Clifford Lim, LTC Kevin Goh and MAJ Jasper Tan for believing that spending another year sitting behind a soldering iron would somehow be relevant to the military, as well as for their enthusiasm, encouragement and efforts in support of my request. MAJ Lim Lit Lam and CPT Goh Yanxun have also been most accomodating during my final months completing the thesis.

For the past few years, the Singapore community in Boston have provided a home away from home. In particular, I knew that I could always count on friends such as Guo-liang, Shirleen, Wenxian, Hongyi, Kah Seng, Serene, Kah Keng, Sarah, Tzu-liang, Rika, Shiming, Wee Lee and Jianlong to hunt for that quick fix of dim sum and sambal chili around Boston, or simply to trade a few sentences of *Singlish* with each other.

In addition, the brothers of Sigma Phi Epsilon fraternity MA Delta chapter have proven once again that strong fraternal bonds need not be forged over alcohol. I am thankful to brothers such as Andrew Clare, Jules Walter, Chris Tostado and Jamie Edwards for their inspiration to be better men, and their dedicated passion to make the world a better place.

Mum, dad and Ruimin have provided the best possible environment that I could imagine growing up in, and have continued to shower me with love and support despite being half the world away. They have been my biggest role models, and this thesis is as much theirs as it is mine.

Finally, to Amelia. For teaching me to discover beauty all over again.

Dedication

To mum, dad and Ruimin,
for believing what's possible
before anyone dared to dream.

Contents

List of Figures

List of Tables

List of Symbols

$\mathcal{S}$:	Set of states
s_t:	True underlying state at time t
$\mathcal{A}$:	Set of actions
a_t:	Action executed at time t
a^i:	The i-th action in $\mathcal{A}$
$\mathcal{Z}$:	Set of observations
z_t:	Observation obtained at time t
$T(s, a, s')$:	Transition function
$O(s', a, z)$:	Observation function
b_t:	Belief at time t
τ:	Belief update operator
R_B:	Reward function defined over the belief space
R_S:	Reward function defined over the state space
γ:	Discount factor
$V_{h,\pi}$:	Value function associated with policy π for planning horizon h
V_h^*:	Value function of the optimal policy π_i^* for planning horizon h
Q_h^*:	Q-value function of the optimal policy π_i^* for planning horizon h
H:	Planning horizon
A, B:	State-transition parameters under stochastic control model
C:	Observation parameters under stochastic control model
F:	Function mapping noise to state space under stochastic control model
G:	Function mapping noise to observation space under stochastic control model
ϵ, δ:	Un-modeled noise under stochastic control model
K:	Kalman gain
$\tilde{\mathcal{A}}$:	Set of macro-actions
$\tilde{a}^i$:	The ith macro-action in $\tilde{\mathcal{A}}$
$\tilde{H}$:	Macro-action planning horizon, or macro-action search depth.
L:	Macro-action length

D:	Number of independent dimensions in the state space
g:	Number of discrete states per dimension
N_z:	Number of observation sequences sampled for each macro-action
N_s:	Number of posterior belief samples for each macro-action
b_{dist}:	Distribution of posterior beliefs

Chapter 1

Introduction

Imagine a hostage-rescue scenario, where a bank building has been taken hostage by terrorists, and commandos 1km away need to be guided across a field to reach the building. There are certain well-marked paths (Figure 1-2a) that ground agents can travel along, and an enemy guard vehicle moves along these paths to guard the building (Figure 1-2b). In addition, some of the routes are blocked by unknown obstacles and terrain, while others are seeded with mines. Once detected and geo-located, the mines can be deactivated using an explosive ordinance disposal vehicle. A Micro-Aerial vehicle (MAV) (Figure 1-1) is tasked with performing aerial surveillance to 1) geo-locate the mines and obstacles, 2) decide which mines need to be deactivated, 3) keep track of the guard vehicle's position, 4) guide the commandos to the bank building in 40 minutes.

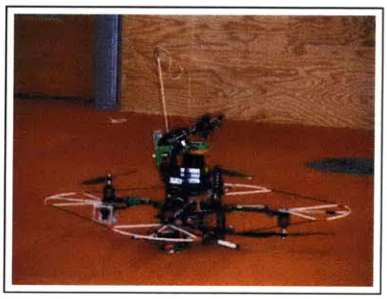
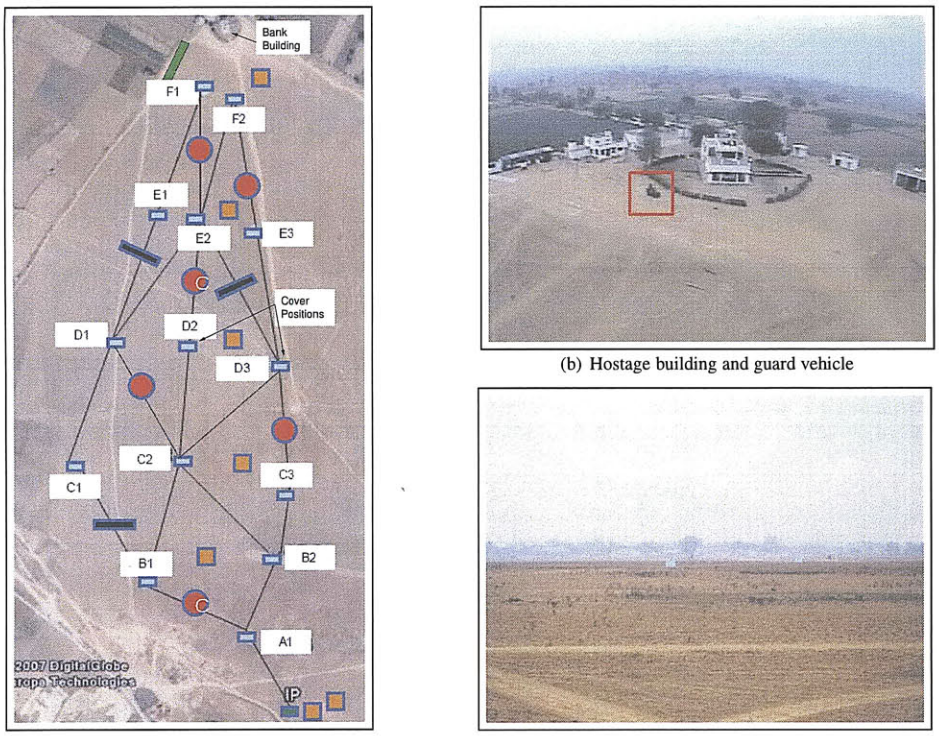


Figure 1-1: Our quadrotor helicopter

This hostage-rescue scenario was the challenge presented to competitors of the 1st US-Asian Demonstration and Assessment of Micro Aerial Vehicle (MAV) and Unmanned Ground Vehicle (UGV) Technology (MAV'08 competition), which we participated in (He et al., 2010). Jointly organized by India's National Aerospace Laboratories, India's Aerial Delivery Research and Development Establishment, and the US Army RDECOM, this scenario models some of the missions that armed forces and other security agencies around the world are currently seeking solutions for.

Executing this mission autonomously requires decision-making in uncertain, partially observable domains, which is a common challenge for artificial agents operating in real-world environments. At every timestep, the MAV seeks to maximize the likelihood of mission success by planning actions using only information that is currently available. Even if the world were fully observable, e.g., the correct guard position was known at the current time, the MAV does not know the guard vehicle's future motions perfectly. Since



(a) Prior Map

(b) Hostage building and guard vehicle

(c) View from the ingress point

Figure 1-2: (a) Bird's eye-view of the MAV08 environment, from the ingress point (IP) to the hostage building. The red dots and black boxes are potential mine locations and terrain obstacles, respectively. These positions are not known *a priori*, and have to be detected by the MAV. (b) View of the hostage building from the on-board MAV camera. The vehicle guarding the building can also be seen (red box). (c) View of the hostage building from the ingress point, 1km away.

the MAV is uncertain where the guard vehicle will be at subsequent timesteps, it has to consider how different guard motions would have different effects on its own actions. Without explicitly reasoning about the uncertain guard vehicle motion, the MAV may be unwittingly detected by the guard vehicle, severely jeopardizing the mission.

Unfortunately, the uncertainty in how the world changes over time is further complicated by the fact that the agent does not have perfect knowledge of the world state, such as the locations of the guard vehicles and the explosive content of the mines. Instead, the agent has to rely on observations that it obtains from its onboard sensors to infer the state of the world. These observations provide incomplete and imperfect information that rarely allow the agent to infer the current state uniquely and correctly. For example, the MAV may be too far away to observe the guard vehicle's motion, and even in close proximity, the observations from the onboard camera sensor may be noisy (Figure 1-2b) and subject to the pitching and rolling of the MAV. Therefore, the MAV must not only plan for a variety of possible future guard vehicle locations, but must also find ways to obtain accurate observations that will enable the MAV to disambiguate the guard vehicle's

location.

In addition, this planning problem is particularly challenging because it requires the agent to plan far ahead in order to select good actions. For example, the agent may have to travel a long distance before it gets useful information about a vehicle that is located far away. Traditionally, a planner constructs plans that condition the agent's future actions on the prior actions taken and observations received. The MAV is receiving observations at some sensor-dependent frequency, typically 10-30 Hz. Choosing new actions after every observation allows the planner to plan its subsequent actions based on the latest available information. Unfortunately, the space of conditional plans for an exact solver also grows in a doubly-exponential fashion in relation to the number of sequential observations, or the number of timesteps, that the agent has to plan ahead for, making it a challenge to plan for long-horizon problems.

This thesis presents three novel algorithms that address the problem of efficient planning under uncertainty for large, partially observable domains. Specifically, we are interested in problem domains where it is necessary to consider situations far into the future in order to choose immediate actions that will result in good performance. We adopt a probabilistic approach to model the uncertainties inherent in the problem domains, and use decision theory to compute policies that incorporate the effects of the probabilistic state estimates and action outcomes. In addition to presenting algorithms that enable long-horizon planning under uncertainty, we also demonstrate that our techniques are scalable to real-world applications on actual autonomous robotic systems.

1.1 Planning under uncertainty in partially observable domains

In partially observable, stochastic problem domains such as the MAV08 scenario presented above, not only are the state transitions at every timestep non-deterministic, but the agent also has to plan its next actions even though the states of the world are not directly observable. The agent only receives observations that provide incomplete and imperfect information about the world, and at every timestep, the agent has to use these observations and its history of prior actions to select an action at the next timestep, so as to achieve mission success, which can also be expressed as maximizing a given reward function.

When the environment, actions and observations consist of a discrete set of possible values, the Partially Observable Markov Decision Process (POMDP), originally from the operations research community (Sondik, 1971), provides a general model for sequential decision making in partially observable environments.[1] Under the POMDP model, the history of prior actions executed and prior observations received can be completely summarized by a probability distribution over world states, also known as a "belief" state. Popular POMDP approaches seek to compute a (near) globally optimal (in)finite-horizon control policy in advance of acting (Kaelbling, Littman, & Cassandra, 1998), mapping belief states to actions for any belief in the belief space. Within the controls community, for the special case of quadratic costs (negative rewards) and linear

[1] Recently, techniques have been proposed to address POMDP problems where the state space is continuous.

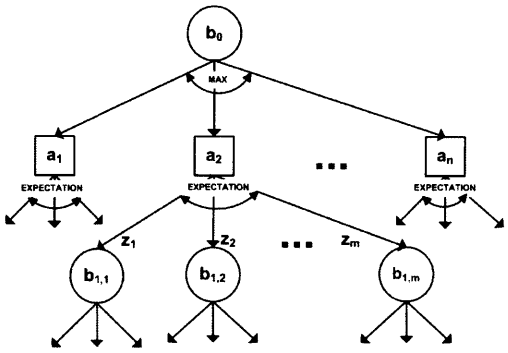


Figure 1-3: A forward search tree. The planner obtains the set of beliefs that are reachable within a limited number of timesteps, and selects the next action to execute by computing the immediate expected rewards at each leaf node and propagating the values up the tree to the root node.

Gaussian transition dynamics and observation models, an optimal infinite-horizon analytic strategy exists, known as the Linear Quadratic Gaussian (LQG) controller (Athans, 1971). More generally, the stochastic and robust control subfields allow for uncertainties in the parameters of the plant model, as well as disturbances that are inherent in both the system and the controller (Dorato & Vlack, 1987; Stengel & Ryan, 1991). Researchers in these subfields place much emphasis on the development of controllers that can account for these random deviations. In this thesis, we propose planning algorithms that are applicable across the various research domains.

1.1.1 Online forward search in partially observable domains

In planning problems with a large number of possible states, actions and observations, it is computationally intractable to compute an optimal strategy for every possible belief. For example, in the MAV08 scenario, the belief state consists of a joint probability distribution over the guard vehicle's location and the explosive content at each mine location. Instead of the standard approach to solving POMDPs by computing an optimal strategy for every possible belief (Sondik, 1971), an increasingly popular technique, known as forward search (Ross, Pineau, Paquet, & Chaib-Draa, 2008) in the POMDP community and the Model Predictive Control/Receding Horizon Control (MPC/RHC) approach from the controls community (Mayne, Rawlings, Rao, & Scokaert, 2000; Kuwata & How, 2004), focuses computational effort only towards belief states that are reachable from the current belief within a limited number of timesteps. For example, the set of reachable beliefs would include possible beliefs of the guard vehicle's location a few timesteps into the future, given the possible observations that the MAV could obtain during those timesteps. Since we are primarily interested in choosing, at every timestep, the next action for the agent, it is sufficient to consider only the reachable beliefs that the MAV could obtain in future timesteps. This condition assumes that the planner can replan fast enough every timestep, after incorporating the latest information available.

Starting from the current belief state, a forward search tree is constructed by considering all possible

actions that could be taken from that belief state, followed by all observations that could be received after an action is taken (Figure 1-3). After repeating the process out to some fixed horizon, the expected values of the posterior beliefs at the leaves are then carried back up the tree in order to select the best action from the root belief. Propagating the values up the tree involves taking the maximum expected value over the actions at each intermediate belief, and an expectation over the observations that could be obtained after each action. The best action from the agent's current belief is executed, and the forward search process is repeated after updating the agent's belief given the action executed and the observation received. Using this forward search approach, impressive results have been demonstrated on a sub-problem of a large RoboCupRescue domain (Paquet, Tobin, & Chaib-draa, 2005).

In addition, online, forward search techniques have the advantage of being applicable in environments where the world models are non-static (Paquet et al., 2005). For example, in the MAV08 scenario above, when new data suggests that the guard vehicle's dynamics have changed, a forward search planner can easily incorporate these changes and replan accordingly, since it re-generates the entire forward search tree at every step.[2]

Forward search approaches can also leverage factored dynamics and sensor models (Paquet et al., 2005). For the MAV08 scenario, the belief state is easily factored into the guard vehicle's location and the explosive content of each of the mines. Factored models often allow belief updating, which is the process of incorporating new information into the belief state, to be performed very efficiently. Finally, forward search can also be used easily in domains where the reward model is either a function of the underlying world state or a direct function of the belief state.

By focusing computational effort only on the reachable beliefs, forward search techniques have been able to solve problems that have larger state, action and observation spaces than those solvable by other state-of-the-art POMDP solvers. Nevertheless, these large problems also typically require planning to a long horizon. For example, the MAV may be required to execute many actions successively in order to reach the bank building. Unfortunately, existing POMDP forward search techniques typically scale poorly with the horizon length, and are unable to search out to longer horizons for a broad class of sensor, measurement and cost/reward models. The reward function rarely contains enough information to accurately evaluate short-horizon plans. To overcome this limitation, existing techniques use a heuristic to estimate the rewards of a plan beyond the fixed horizon. This is similar to the MPC/RHC approaches from the controls community that optimize their conditional plans for a short, finite horizon, before using a heuristic to estimate the cost for the longer horizon. In general, these techniques construct a conditional plan out to a fixed planning horizon that always conditions the next action based on the observation received at the previous timestep, but does not condition the agent's actions on the observations received beyond the planning horizon; we call these techniques *fully-conditional* planners. If the MAV planner branches on possible observations of the guard vehicle after *every* action is taken, the computational complexity of such an approach is an exponential

[2]However, as forward search requires the dynamics models to roll out potential future observations and the result of future actions, the models must be assumed either to be static for the length of the forward search planning horizon or to change in a predictable way.

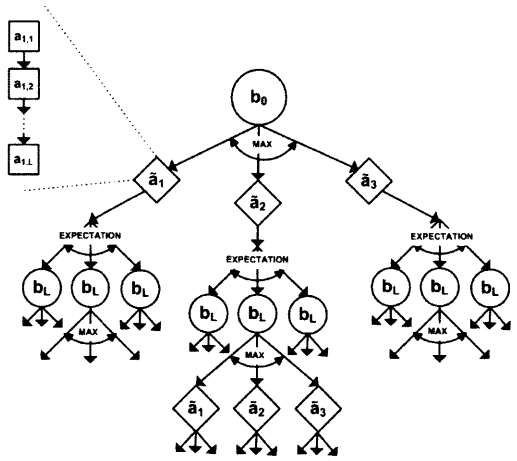


Figure 1-4: Macro-action forward search tree. Macro-actions are conditioned on received observation sequences only at the end of each macro-action.

function of the action and observation branching factors, making it difficult to construct a forward search tree beyond a very limited planning horizon. Therefore, though forward search techniques can be substantially faster than computing a policy over the entire belief space, fully-conditional forward search planners will still find long-horizon problems challenging.

1.2 Semi-conditional planning with macro-actions

Rather than determine the agent's next best actions by generating conditional plans that branch on observations after *every* action, we hypothesize that good plans can be generated if actions are chosen based on the received observations only at the end of a fixed-length action sequence. We call this class of solution techniques *semi-conditional* planners, since the conditional plan constructed only conditions actions on the received observations at certain key points (Figure 1-4). Between these key points, we assume that a temporally-extended, fixed-length action sequence, which we define as a "macro-action", is executed. We apply these open-loop action sequences in a forward-search manner, where we only consider beliefs that are reachable from the agent's current belief under different macro-actions and the resulting observation sequences. By only conditioning on the observations received at the end of each macro-action, the planner has the option of not branching on possible observations and subsequent actions after every evaluated action. In addition, if the planner only considers a subset of the admissible macro-actions and/or observation sequences, it effectively restricts the space of policies considered and devotes the computational savings towards planning to a longer horizon.

Macro-actions have been considered in depth within the fully observable Markov decision process community, and are a subset of the class of temporally-extended action policies known as "options" (Sutton,

Precup, & Singh, 1999). Often posed as part of a semi-Markov decision process (Mahadevan & Khaleeli, 1999; Puterman, 1994), the options used can themselves be mini-policies that condition on a subset of the state variables describing the environment, or mini-policies that are restricted from the full set of policies. The termination of a mini-policy can also be conditioned on the arrival at a particular state. However, such definitions are no longer appropriate in partially observable environments, since in general the agent will only have a probability distribution over whether the agent has reached the terminal state, resulting in a terminal condition that is not well-defined. While the options concept can be extended to partially-observable domains by defining the termination and initiation conditions over the belief space, it will be non-trivial for a domain expert to define these "partially-observable" options, let alone have them generated automatically. Instead, in this thesis we define macro-actions as fixed-length unconditional sequences of primitive actions. By making the action sequence unconditional, that is, independent of observations received during the sequence, the policy space is significantly reduced, making the planning problem simpler. In addition, the fixed length ensures that the termination condition is well defined in partially observable environments.

Although some problem domains do require fully-conditional planning for good performance, there exist problems where it is only necessary to condition at key choice points in order to obtain good performance. Returning to the MAV08 scenario, the MAV may plan a long path to the bank building to make observations of the guard vehicle's position, but it only needs to condition its next actions based on the observations it receives near the bank building, while the observations received during the path to the bank building are likely to be less important for choosing future actions. More generally, semi-conditional planning can be useful for reward models where similar rewards will be obtained for taking a particular action from any state, except for a few states where the agent will receive a drastically different reward from taking that same action. In domains with this kind of reward model, the agent only needs to condition its actions on the observations for those beliefs where the expected reward for the best action has high variance. Problem domains that exhibit such structure in the reward function, such as goal-oriented motion-planning problems, often have a small set of sub-goals that need to be accomplished, but require a large time lapse before subsequent sub-goals can be accomplished. In contrast, problem domains that require careful balancing of risks and rewards at every timestep, such as partially observable games and control problems with large process noise, are unlikely to find macro-actions useful.

There currently exist several offline partially observable planning approaches that use the notion of a temporally-extended action sequence. Some have used a hierarchical approach to solve discrete-state POMDPs (Pineau, Gordon, & Thrun, 2003b), while others have proposed to plan with hierarchical controllers that are either user-specified (Hansen & Zhou, 2003) or automatically generated (Charlin, Poupart, & Shioda, 2008). However, none of these approaches have demonstrated the ability to extend these techniques to large-scale POMDP problems that are similar in size to those presented in this thesis. Offline solvers have used macro-actions to guide the sampling of belief points in larger domains, but still performed fully-conditional planning when performing value-function approximation (Theocharous & Kaelbling, 2003) or

standard value iteration (Kurniawati, Du, Hsu, & Lee, 2009). These techniques were also restricted to the goal-directed robot-navigation domain. Finally, Hsiao et al. (2008) used a form of macro-actions in an on-line, forward search manner, but their focus was on hand-coding a very small set of action trajectories out to short horizon to address the specific task of robust manipulation under uncertainty. To our knowledge, macro-actions have not been used in a domain-independent fashion using an online, forward search approach for decision-making under uncertainty in partially observable environments.

1.2.1 Challenges of semi-conditional planning

Two technical challenges have to be addressed to perform semi-conditional planning efficiently. First, semi-conditional planning will only result in good performance if good macro-actions are constructed. Semi-conditional planners restrict the plan space of the search process by only evaluating a subset of the admissible macro-actions. While this restriction enables the planner to search much farther in the future, poor performance may result if the plan space is overly restrictive. In addition, the planner will only choose good actions for the agent if good choice points are chosen along the macro-action forward search tree to incorporate the observations, so that the expected values of the different root actions are more accurately computed. For example, if the semi-conditional planner fails to condition the MAV's actions on the observations it will receive at the bank building, the MAV will likely choose a sub-optimal action. It is therefore crucial to generate macro-actions that will lead to good performance, i.e., the macro-actions that are anticipated to be part of a good plan.

The second technical challenge concerns the set of possible posterior beliefs that can be obtained after a macro-action is executed. The expected immediate reward received during a macro-action, and the expected future reward received after a macro-action, are both functions of the set of posterior beliefs that could result after a macro-action is executed. A question that naturally follows is how to obtain the set of posterior beliefs that could result after a single macro-action. Each belief is a function of both the action sequence and some observation sequence. Exhaustive enumeration of the possible posterior beliefs would therefore be intractable when the set of possible observations is large, or when the macro-action consists of a long sequence of primitive actions. Sampling the observation sequences can still be computationally intensive due to the number of calculations that must occur at each step of each sampled observation sequence.

In this thesis, we seek to address these two technical challenges and develop algorithms that enable efficient semi-conditional planning.

1.3 Thesis statement

The central thesis of this document is that for many planning problems in partially observable, stochastic domains, good performance can be obtained by only conditioning the agent's actions on the observations obtained at key points along a long horizon. We will demonstrate how to perform semi-conditional planning

efficiently and in a domain-independent fashion on a variety of long-horizon, large-scale, real-world planning under uncertainty problems.

1.4 Thesis contributions

The main contributions of this thesis are three semi-conditional planners that perform efficient planning under uncertainty with macro-actions, as well as hardware demonstrations of the algorithms using a robotic helicopter. These algorithms together address the two key technical challenges necessary for efficient planning under uncertainty with macro-actions — namely, how macro-actions can be automatically generated, and how to efficiently obtain the set of possible posterior beliefs after a macro-action.

Automatic macro-action generation (PUMA)

Addressing the first technical challenge of constructing the set of macro-actions automatically, we present a domain-independent algorithm for *planning under uncertainty with macro-actions* (PUMA). Borrowing the notion of sub-goal states from the fully-observable planning literature (McGovern, 1998; Stolle & Precup, 2002), we propose the heuristic that macro-actions can be designed to take the agent, under the fully-observable model, from a possible start state under the current belief to a sub-goal state that will provide either a large immediate reward or a large gain in information under the partially observable model. These finite-length, open-loop macro-actions are automatically constructed by formulating and solving a stochastic shortest path problem, which will generate the policy of reaching the sub-goal state from every state within a fully-observable context. Assuming that the sub-goal state is reachable from the start state, we can then generate a macro-action by selecting actions from the policy starting from the start state and assuming that the maximum likelihood state is reached at every step.

However, some of these temporally-extended macro-actions may become exceedingly long if they only terminate at the sub-goals; instead, it may be more beneficial for the planner to condition on the associated observation sequences at certain choice points before the sub-goal is reached. Good choice points at which it is important to condition in order to obtain good performance are not typically known in advance, and for certain problems, good performance may even require computing a fully conditional policy. We therefore extend the planner to allow the algorithm to converge to an ϵ-optimal forward search tree for any problem in an anytime manner. If additional computational time is available for planning, we iteratively refine macro-action branches in the forward search tree, progressively increasing the amount of conditional branching in the tree. Together, the PUMA algorithm enables the planner to obtain good performance in many real-world planning problems quickly by searching to a long planning horizon, but guarantees convergence to the optimal policy as more planning computational time is made available.

Experimental results demonstrate that our approach outperforms existing state-of-the art solvers on two large POMDP problems where long horizon look-ahead is necessary — a scientific exploration problem,

as well as a target-tracking and monitoring problem. The experimental results also show that anytime improvement allows PUMA to achieve good performance in domains that require fully conditional planning, suggesting that PUMA has significant potential as a generic planner for large POMDP problems.

Computing the posterior belief distribution (PBD)

We next address the second technical challenge of computing the set of possible posterior beliefs after a macro-action in an efficient manner. Rather than sampling observation trajectories to obtain a particle approximation of the true posterior distribution of beliefs, we present an analytic method for computing the distribution over the set of posterior beliefs that could result from a single macro-action for a subset of problem domains. This distribution captures any observation sequence that could occur during the macro-action. Computing the distribution over beliefs analytically enables us to avoid the exponential computational dependence on the planning horizon length, and also allows us to avoid the costly step of performing belief updates along each observation trajectory associated with the macro-action.

Specifically, when the agent's belief of the world can be represented as a Gaussian distribution, and the transition and observation models are linear Gaussian, then all posterior beliefs in this distribution have the same covariance, and all posterior means are normally distributed over the continuous state space. When the linear Gaussian assumption does not hold, although the exact distribution of posterior beliefs may not be available analytically, an approximate version of the analytical belief update exists for problems with observation models that belong to the exponential family of distributions.

We present a semi-conditional planner, the posterior belief distribution (PBD) algorithm, which takes advantage of the analytic computation of the posterior distribution of beliefs for efficient planning under uncertainty. A formal analysis of our approach is presented, before benchmarking the PBD algorithm's performance on two simulation experiments: the same scientific exploration domain mentioned in the earlier section, and an extension of the earlier target-tracking problem to continuous states and a 3D environment. These experiments respectively model the mine-clearing and guard-tracking missions from the MAV08 competition. We also demonstrate our algorithm on actual hardware for the target-tracking problem.

Posterior belief distribution for multi-modal Gaussian beliefs (MMPBD)

The PBD algorithm assumes that the agent's belief can be represented as a uni-modal Gaussian distribution, especially when the transition and observation models are linear Gaussian. Unfortunately, this belief representation assumption may not always hold, as in the case of the MAV tracking commandos traveling along the ground paths, as shown in Figure 1-2a. Instead, we assume a more general belief representation by representing the agent's belief as a multi-modal Gaussian distribution, and show that we can still approximate the distribution of posterior beliefs at the end of a macro-action.

Multi-modal Gaussian distributions can be parameterized with a set of mean-covariance pairs and a set

of weights. Each mean-covariance pair represents the statistics of each Gaussian mode, while the weights represent the probability that each mode corresponds to the true state of the world. In addition, we represent the problem domain's transition model as a switching state-space dynamics model, while the original linear Gaussian observation model is augmented with observations that contain negative information.

To obtain the distribution of posterior beliefs after a macro-action, we require the joint distribution over both the mode statistics and the posterior weights of each mode. We introduce a distance-varying covariance function that is a Gaussian function over the state space, so as to model the bimodal characteristic of the observation model. For the target-tracking problem, the Gaussian function models the property that the observation noise typically increases with greater distance between the agent and the target.

This covariance function also enables us to use the Fisher information associated with the observation model to update the respective weights of the modes of the agent's belief. The Fisher information associated with the observation model measures the accuracy of a state estimate due to the measurement data alone, and is not a function of the actual measurement itself; instead, it is dependent only on the parameters of the observation model (Maybeck & Siouris, 1980). Since we have a noise covariance value that varies over the state space, the observation model parameters of each mode capture the likelihood that the agent will receive a positive observation, and allows us to use the relative Fisher information of each mode to update their respective posterior probabilities.

We present the *multi-modal* variant of the posterior belief distribution (MMPBD) algorithm. We demonstrate this algorithm on another variant of the target-tracking problem, restricting both the targets' and agent's motion to move along a road network. Simulation results are presented comparing our algorithm to alternative strategies for a helicopter tracking multiple targets on a road network, and we also demonstrate our algorithm on actual hardware.

Helicopter experiments

It is critical that the planning algorithms that we develop are applicable on actual systems in the real-world, capable of accomplishing real-world missions similar to those presented during the MAV08 competition. In this thesis, we demonstrate the applicability of our algorithms on an actual autonomous robotic helicopter (Figure 1-1). In previous work (Bachrach, He, & Roy, 2009; He et al., 2010; He, Prentice, & Roy, 2008), we have developed quadrotor helicopters that are capable of autonomous flight in unstructured and unknown indoor and outdoor environments. In particular, our helicopter is equipped with a laser rangefinder and uses the laser data to estimate its position and velocity in GPS-denied indoor environments.

The helicopter is capable of executing flight plans or controlled hover-in-place tasks completely autonomously, enabling us to implement the higher-level planning algorithms that have been developed in this thesis. Using a downward-pointing color camera for target detection, we apply the PBD algorithm to a live instantiation of a target-monitoring experiment. This experiment is similar to the guard-tracking mission in the MAV08 competition, and our algorithm enables the autonomous helicopter to obtain superior

performance relative to existing techniques. We also implement the MMPBD algorithm on the helicopter to perform target-tracking along a road-constrained network, simulating the MAV's task of monitoring the commandos progress along the marked paths during the MAV08 competition. These experimental validations suggest that our approaches have practical potential for planning in real-world, long-horizon, partially observable domains.

1.5 Document outline

In Chapter 2, we present the general problem of planning under uncertainty, as posed by different research communities. We also examine state-of-the-art approaches for planning under uncertainty in partially observable domains. Chapter 3 presents the forward search planning approach and demonstrate how macro-actions can be incorporated.

In Chapter 4, we present the planning under uncertainty with macro-actions (PUMA) algorithm, where the macro-actions are automatically generated and iteratively refined as more computation planning time is available.

In Chapter 5, we develop the Posterior Belief Distribution (PBD) algorithm, building on the insight that when the agent's beliefs are representable as Gaussian distribution, the distribution of posterior beliefs can be calculated directly. Chapter 6 presents experimental results for the PBD algorithm, both in simulation and on an actual robotic helicopter.

Chapter 7 presents the Multi-modal Posterior Belief Distribution (MMPBD) algorithm, relaxing the unimodal Gaussian assumption for a road-network constrained target-tracking application. An actual hardware demonstration of the MMPBD algorithm is also presented.

Finally, we summarize the thesis and discuss interesting directions for future research, as well as some of the unresolved problems of planning under uncertainty in partially observable domains.

Chapter 2

Background

Planning under uncertainty in partially observable domains is a challenging task that has been of interest to researchers from multiple research communities. We begin the technical content of this thesis by describing the state of the planning under uncertainty literature. Due to the multi-disciplinary interest in this problem, planning under uncertainty in partially observable domains has been addressed under a variety of planning formalisms, each of which belongs to a different research community.

In this chapter, we formalize the planning under uncertainty problem in a manner that is slightly more general than the models used by the different research communities. Key differences amongst the different planning formalisms are highlighted; these differences also help explain the variance in solution techniques amongst the different research communities. Algorithms from the literature are then discussed. Unsurprisingly, many algorithms that address the challenge of planning under uncertainty in partially observable environments exist, and we do not attempt to summarize all the algorithms that have been proposed. Instead, we focus on those techniques that are most relevant to the algorithms that are proposed in this thesis, with particular emphasis on each technique's ability to solve real-world, planning under uncertainty problems. Real-world problem domains not only have a large number of states, actions and observations, but also require the agent to search far into the future to obtain good performance. Examining state-of-the-art techniques from the literature provides motivation for the approach we have adopted in this thesis, as we seek to address the shortcomings of earlier approaches.

2.1 Planning under uncertainty in partially observable environments

Sequential decision-making in stochastic, uncertain domains requires an autonomous agent that is operating under action and state uncertainty to execute an action every timestep, in order to maximize a given reward function. Optimizing the agent's course of action in these domains is challenging because transitions in the state of the world are not deterministic, and the agent does not have perfect knowledge of the world at every

timestep.

A wide range of real-world planning problems, such as those associated with robotics applications, require planning under uncertainty in partially observable environments. Seldom does an autonomous agent have perfect motors and actuators that allow it to predict its resultant state after each action without incorporating any feedback from its sensors. Even if the agent is initially perfectly aware of the state of the world, it would be unable to predict the resultant state after actions with absolute certainty. An agent also frequently operates with other entities in the environment — these entities may operate independently from the agent, and hence the agent may not have control over them. Therefore, even if the agent had perfect control over its own state variables, it could still be uncertain of how the world changes during an action.

In addition, a robotic agent often finds itself operating in partially observable environments. Rather than have access to omniscient information of the world, an agent is typically equipped with local sensors that only provide observations of the agent's immediate environment, and has to make inferences of the state of the world solely based on the observations it obtains from these sensors. Furthermore, these observations could be noisy, which would mis-inform the agent if it assumes that the observations are perfectly accurate. Some robotics researchers who are primarily interested in estimating the state of their agents have circumvented this issue by operating inside motion capture systems or assume that Global Positioning Systems (GPS) provide sufficiently accurate state knowledge. Nevertheless, if autonomous agents are to be able to operate in generic environments and interact with other agents in the world, algorithms that enable agents to plan under action uncertainty in real-world, partially observable environments have to be developed.

In this thesis, we focus on planning problems where the dynamics, observation and reward models are known but the state of the world is only partially observable. The techniques developed are therefore model-based, rather than the model-free approaches such as neuro-dynamic programming (Bertsekas & Tsitsiklis, 1996) that directly optimize a policy by simulation, or the reinforcement learning frameworks (Sutton & Barto, 1999) that learn the transition and reward models as the agent interacts with the environment. These alternate techniques do not require an explicit model of the environment, but explore the world by performing repeated simulation and rationalize about the rewards obtained. Instead, we focus on problem domains where the transition, observation and reward models are known *a priori*.

Beyond robotics, other application domains that require sequential decision-making in stochastic, uncertain environments include assistive technologies (Hoey et al., 2005), spoken-dialog systems (Roy, Pineau, & Thrun, 2000) and gesture recognition (Darrell & Pentland, 1996). As a result, addressing the problem of planning under uncertainty in partially observable environments has been of great interest to researchers from many research communities.

In this chapter, we first present a planning under uncertainty model that will be used throughout this thesis, before highlighting the key differences between this model and existing formalisms that have been presented in the literature. Our planning model follows most closely from and is slightly more general than the discrete-time Partially Observable Markov Decision process, which we also describe in greater detail

below.

2.1.1 Planning under uncertainty model

Formally, we assume that our planning under uncertainty problem consists of the following known components:

- $\mathcal{S}$ is a set of states $s \in \mathcal{S}$, which can be either discrete or continuous
- $\mathcal{A}$ is a set of actions (controls) $a \in \mathcal{A}$, which can be either discrete or continuous
- $\mathcal{Z}$ is a set of observations $z \in \mathcal{Z}$, which can be either discrete or continuous
- T is the transition function, and is also known as the dynamics model. $T(s, a, s') = p(s'|s, a)$ encodes the probability of transitioning to state s' after taking action a from state s. The probability of the next state is a function only of the previous state and action (this class of transition function is often referred to as the Markov assumption), and makes the future states conditionally independent of previous states given the current state and action. In a model with discrete states, this function is specified by a multinomial distribution. In a linear Gaussian model, it is specified by a linear function with additive Gaussian noise.
- O is the observation function,and is also known as the measurement or sensor model. $O(s', a, z) = p(z|s', a)$ encodes the probability of receiving observation z after taking action a and transitioning to state s'. Similar to the transition function, the observation function is a multinomial distribution when the states are discrete, and is a linear function with additive Gaussian noise for a linear Gaussian model.
- $R_B(b, a)$ is a reward (or cost) function that describes the utility the agent receives for taking action a when it has a belief b, which is a distribution over states (b will be defined more formally below). If the reward is expressed in terms of the state, the reward function is then expressed as $R_S(s, a)$, and $R_B(b, a)$ is an expectation over the state, i.e., when the states are continuous, $R_B(b, a) = \int_s p(s) R_S(s, a) ds$
- γ is a discount factor that determines the relative weights of immediate utilities to utilities that will be received at a later timestep.

The states $\mathcal{S}$ are not fully observable. Instead, at every timestep, the agent receives an observation z after taking an action a. The agent must therefore make decisions based on the prior history of observations it has received, $z_{1:t}$, and actions it has taken, $a_{1:t}$, up to time t.

Unfortunately, over time, the number of action-observation pairs accumulated grows quickly. Nevertheless, it is possible to summarize the previous history of actions and observations using a probability distribution over the state space $\mathcal{S}$, also known as a belief state (see Figure 2-1) (Astrom, 1965). A belief $b_t(s)$

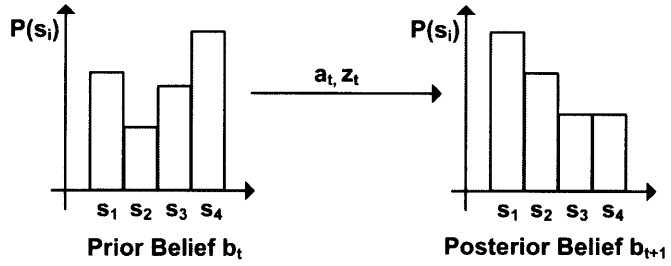


Figure 2-1: A belief update after taking action a_t and obtaining observation z_t. The belief b_t is a probability distribution over the state space.

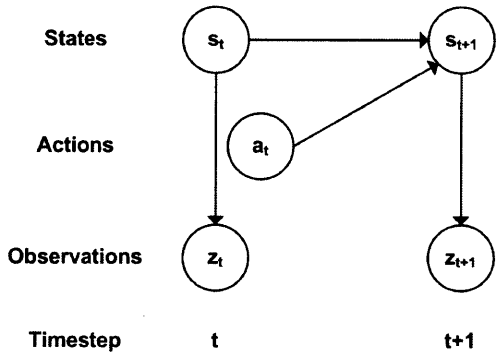


Figure 2-2: Planning under uncertainty graphical model.

summarizes the probability of the world being in each state given its past history,

$$b_t(s) = p(s_t = s|b_0, , a_0, z_1, \ldots, z_{t-1}, a_{t-1}, z_t) \tag{2.1}$$

When an action a_t is executed from the prior belief b_t, and after an observation z_{t+1} is obtained, a new belief b_{t+1} is constructed. This belief state encapsulates all the information necessary to determine how the world is expected to change at the next timestep for every action a that is taken. The agent can therefore select actions based only on the current belief state, rather than on all past actions and observations (Smallwood & Sondik, 1973).

Figure 2-2 shows the graphical model for belief updating in partially observable environments. The agent updates its belief at each step, after taking an action a and receiving an observation z using the Bayes filter,

$$b_{t+1}(s') = \tau(b_t, a, z) = \frac{O(s', a, z) \int_{s \in \mathcal{S}} T(s, a, s') b_t(s) ds}{\int_{s''} O(s'', a, z) \int_{s \in \mathcal{S}} T(s, a, s'') b_t(s) ds ds''} \tag{2.2}$$

$$= \eta\, O(s', a, z) \int_{s \in \mathcal{S}} T(s, a, s') b_t(s) ds \tag{2.3}$$

where $\tau(b_t, a, z)$ represents the belief update function and η is a normalization constant to ensure that the posterior belief is a proper probability distribution over the state space. Without loss of generality, in this section we assume that the state and observation spaces are continuous.

The goal of the planning under uncertainty problem is to choose an action a_t for the agent to execute based on its belief state b_t at time t, so as to maximize the expected sum of discounted utilities over H timesteps, which is given by:

$$\sum_{t=0}^{H} \gamma^t E[R_B(b_t, a_t)] \tag{2.4}$$

where $E[R_B(b_t, a_t)]$ denotes the expected reward at timestep t given the action a_t and possible observations received.

The action a_t is determined by the agent's policy $\pi : b \rightarrow a$, which is a mapping from belief states to actions. Each policy π is also associated with a value function $V_\pi : b \rightarrow \mathcal{R}$, specifying the expected total discounted reward of executing policy π starting from b.

This value function can be calculated using the Bellman equation (Bellman, 1957), and can be separated into an expectation of the immediate reward and the sum of future expected rewards

$$V_\pi(b) = R_B(b, a) + \gamma \int_{z \in \mathcal{Z}} p(z|b, a) V_\pi(\tau(b, a, z)) dz, \tag{2.5}$$

where $p(z|b, a)$ is the probability of obtaining observation z after executing action a from belief b, and can be computed according to

$$p(z|b, a) = \int_{s' \in \mathcal{S}} O(s', a, z) \int_{s \in \mathcal{S}} T(s, a, s') b(s) ds ds' \tag{2.6}$$

The optimization problem is to compute the optimal policy π^*, which represents the action that the agent should execute from every belief in order to maximize V_π.

The value function V^* of the optimal policy π^* is the fixed point of Bellman's equation $V^* = \overline{H} V^*$:

$$V^*(b) = \max_{a \in A} \left[R_B(b, a) + \gamma \int_{z \in Z} p(z|b, a) V^*(\tau(b, a, z)) dz \right] \tag{2.7}$$

$\overline{H}$ is also known as the Bellman backup operator. Another useful quantity is the Q-value, which represents the value of executing action a from b

$$Q^*(b, a) = R_B(b, a) + \gamma \int_{z \in \mathcal{Z}} p(z|b, a) V^*(\tau(b, a, z)) dz. \tag{2.8}$$

Note that the only difference between the Q^*-value and V^* is that the max operator is omitted. Q^* calculates the value of executing a from b by assuming that the agent follows the optimal policy every step after

executing a.

2.1.2 POMDPs

Originally proposed by the operations research community and subsequently embraced by the artificial intelligence community, the Partially Observable Markov Decision Process (POMDP) (Astrom, 1965; Dynkin, 1965) model is an extension of the Markov Decision Process (MDP) (Bellman, 1957) formalism, and is a popular model for planning under uncertainty in partially observable environments.

Our planning under uncertainty model follows the POMDP model closely. In contrast to the model presented in Section 2.1.1, however, POMDPs typically assume that the problem domains have discrete states, transition and observation models, though recently there have been a few exceptions that have sought to extend the POMDP techniques to parametric representations (Brooks et al., 2006; Porta et al., 2006; Brunskill et al., 2008). In addition, POMDPs assume that the reward model is a direct function of the state, rather than a function of the agent's belief. Such a restriction implies that POMDP models are unable to represent reward functions that, for example, directly values low entropy in the agent's probability distribution over the state space, since entropy is a function of the belief rather than of any particular state.

2.1.3 Stochastic/robust control

Within the controls community, the stochastic and robust control subfields have also explored planning in stochastic and uncertain problem domains. These subfields allow for uncertainty in the parameters of the plant model, which comprise both the state-transition and observation models. Stochastic and robust control algorithms therefore not only have to deal with non-deterministic state-transitions and noisy observations, but also have to account for the possibility that the transition and observation models may be incorrect.

The robust control problem typically consists of a system model that is described by a set of linear discrete time-difference equations,

$$s_{t+1} = A_t s_t + B_t a_t + F_t \epsilon_t, \tag{2.9}$$

$$z_t = C_t s_t + G_t \delta_t \tag{2.10}$$

Using notation similar to our planning under uncertainty model, s_t, a_t, and z_t denote the state, control inputs and outputs respectively. This planning model assumes a continuous representation of the state space, as opposed to the discrete state space that is common amongst POMDP problem domains. In addition, ϵ_t and δ_t represent the unmeasured noise that enters the system. Broadly, A_t and B_t represent the state-transition model parameters, without the non-deterministic component, which is represented by F_t. Similarly, C_t contains the parameters of the observation model, while G_t maps the noise into observation space. In contrast to our planning under uncertainty model that represents the transition and observation function with a single probability distribution function each, the robust control formulation therefore assumes that the

functions can be separated into deterministic and noisy components, and that the noise is independent of the state and action. Nevertheless, it is possible to represent both the transition and observation models from the controls community (Equations 2.9, 2.10) within our planning under uncertainty model, each as a probability distribution function. Indeed, in Chapters 5-7, we use the two system models interchangeably when formulating the problem.

In addition to the equations above, the controlled system can also be subjected to constraints on the state components, as well as stability constraints that are necessary to guarantee closed-loop stability of the controller. The planning model adopted by the controls community is therefore unique in that constraints can be specified separately from the reward function. In contrast, the POMDP planning model would have had to specify each constraint using large negative rewards.

Researchers have focused on developing controllers that are robust to errors in the model, achieving stable performance in the presence of bounded modeling errors. This emphasis contrasts with our planning under uncertainty model that seeks to maximize the agent's expected discounted reward, even if the resultant policy produces drastically reduced performance under slight variations in the transition or observation models. Problem domains analyzed by the controls community include designing controllers for industrial plants and for autonomous vehicles navigating through obstacle fields; these domains typically have hard constraints that must be satisfied, despite bounded uncertainty in their system models. In comparison, the range of POMDP problem domains is more varied.

2.1.4 Sensor resource management

Lastly, the sensor resource management community also analyzes the challenge of planning in partially observable environments, often in the context of a target-tracking problem. One or more agents are tasked with tracking multiple targets in an environment; these agents often have onboard sensors that have limited field-of-views, and/or generate noisy observations of the environment. Much of the research within this community has focused on analyzing and characterizing the agents' sensor and dynamics models, though increasingly, planning algorithms have been developed for the agents to operate in these partially observable environments. Not surprisingly, the transition and observation models of problem domains from this community are typically more complicated than those from the other communities.

In contrast to the POMDP community that seeks to generate policies that maximize the expected discounted reward, or the control community where the policies have to be robust to bounded errors in the plant model, the sensor resource management community focuses on finding policies that provide the greatest information gain, or lead to the greatest reduction in entropy associated with the agent's belief. Such policies are found by employing an information theoretic reward function, where the reward function is a direct function of the entropy associated with the agent's belief. A larger reward is obtained for a belief with a narrower distribution over the state space, relative to a belief that more closely resembles a uniform distribution over the state space. Such a reward function cannot be represented within the traditional POMDP model, but fits

within our planning under uncertainty model.

2.2 Algorithms for planning under uncertainty

The previous section reviews the different models that each research community has adopted to address the challenging task of planning under uncertainty in partially observable environments. As we will describe in subsequent chapters, the algorithms proposed in this thesis adopt the planning model presented in Section 2.1.1; these algorithms can be applied throughout the three research fields discussed above.

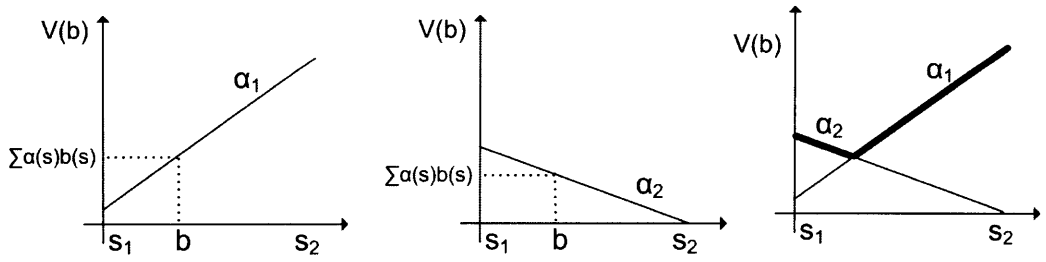
Nonetheless, existing algorithms adopt one of the traditional models for planning under uncertainty. In this section, we first provide a brief overview of algorithms from each of the research communities. As the literature is too vast to provide a comprehensive review of all the existing techniques, we instead focus on the most relevant research that will provide sufficient context to center our approach. Specifically, after presenting algorithms that provide exact solutions to particular planning under uncertainty formulations, we evaluate those approximate planning techniques that seek to exploit the problem domains in ways similar to the algorithms we propose in this thesis. In addition, we also present those techniques that we compare our algorithms against during our experimental evaluations.

A planning under uncertainty algorithm typically consists of two components — a policy generation component where the planner determines the action that the agent should execute, as well as an execution phase where the agent executes the action dictated by the policy and updates its belief. Although the execution phase is not technically part of the planner, existing planning approaches in the literature vary both in terms of how the policies are generated, as well as when the policies are generated relative to when they are executed.

2.2.1 Exact solution techniques

For a POMDP problem domain with a finite horizon H, the problem can be solved exactly using the value iteration algorithm (Sondik, 1971). Value iteration uses a dynamic programming approach to compute increasingly accurate values of the optimal value function V^* for each belief state. Computing the optimal value function also naturally computes the optimal policy π^*, which provides the action that maximizes the Bellman equation for every belief according to Equation 2.7.

The general principle of value iteration is to iteratively build on the value functions that have been computed for shorter horizons in order to generate policies for longer horizons. To illustrate the exact value iteration algorithm, we assume that there exists a two-state, two-action and two-observation POMDP problem. We will first show how to compute a one-step value function, which corresponds to the optimal value that the agent can obtain from every belief if it can only take a single action before the problem terminates. Using the one-step value function, we then show how the two-step value function can be computed. The approach is independent of the horizon of the original policy, and hence illustrates how successively longer plans can be constructed from shorter ones.



(a) $V_{1,p}(b)$ associated with each α-vector (b) $V_1(b)$

Figure 2-3: (a) Different α-vectors dominate different parts of the belief space. (b) The optimal value function can be represented by the supremum set of α-vectors.

Value function for single-step planning horizon

In the first iteration, the algorithm assumes that the agent can only make a single decision, i.e., the action it executes next is the first and last action it can execute. Under these conditions, there exist $|\mathcal{A}|$ possible actions that can be executed from any state.

We first compute the value of executing each action from every state in the problem. In the case of a single-step planning horizon, we consolidate, from each state s, the immediate rewards $R_S(s, a^i)$ of executing the action a^i. This process generates an $|\mathcal{S}|$-length vector for each action.

Following Sondik (1971), we define each $|\mathcal{S}|$-length vector as an α-vector. Each of these α-vectors also corresponds to a policy tree with depth equal to the planning horizon; a policy tree of depth h prescribes an action to take with h steps to go, as well as the subsequent actions to take for the remaining $h-1$ steps when different observations are obtained at every stage.

For the one-step planning horizon, the policy tree corresponding to the α-vector $\alpha_{1,p}$ is of depth one, and the action a^p is the action that should be taken at the root. From a particular belief $b_t(s)$, the expected value of executing the action a^p can then be found by performing a dot product between the α-vector $\alpha_{1,p}$ and the belief b_t, which is itself an $|\mathcal{S}|$-length vector that describes the agent's probability distribution over the state space. The expected value of executing the policy tree from b_t is therefore

$$V_{1,p}(b_t) = \langle \alpha_{1,p}, b_t \rangle \tag{2.11}$$

$$= \sum_{i=1}^{|\mathcal{S}|} \alpha_{1,p}(s_i) b_t(s_i) \tag{2.12}$$

where $\langle a, b \rangle$ is the inner dot-product of two equal-length vectors.

The optimal value function for the one-step planning horizon then consists of the supremum of all the dot-product values

$$V_1^*(b_t) = \max_{\alpha_{1,p} \in \Gamma_1} \langle \alpha_{1,p}, b_t \rangle \tag{2.13}$$

where Γ_1 is the set of $|\mathcal{A}|$ α-vectors. The value function over the entire belief space can therefore be represented by a set of α-vectors. Figure 2-3 shows the expected value of belief b when it executes the single-step policy tree associated with two different α-vectors, α_1 and α_2. Figure 2-3b also shows the value function over the belief space, demonstrating that different α-vectors can dominate different parts of the belief space, i.e. for different beliefs, different α-vectors may result in the highest expected value.

Computing value functions for longer horizons

Having computed the value function for a single-step horizon V_1^*, value iteration seeks to generate the two-step value function V_2^*. More generally, given a $h-1$ step value function, V_{h-1}, the algorithm uses the previous value function and the Bellman backup operator $\overline{H}$ (Equation 2.7) to compute the value function for the longer horizon V_h, according to

$$V_h^*(b) = \max_{a \in A} \left[R_B(b,a) + \gamma \sum_{z \in \mathcal{Z}} p(z|b,a) V_{h-1}^*(\tau(b,a,z)) \right] \tag{2.14}$$

while the optimal policy corresponds to the action that maximizes the value function from every belief

$$\pi_h^*(b) = \operatorname*{argmax}_{a \in A} \left[R_B(b,a) + \gamma \sum_{z \in \mathcal{Z}} p(z|b,a) V_{h-1}^*(\tau(b,a,z)) \right] \tag{2.15}$$

The Bellman backup operator in both equations includes the immediate rewards obtained from executing action a from belief b, followed by the expected value at the next timestep, given that the agent took action a. Because the agent is unable to anticipate which observations it may receive during the execution of the policy, and therefore does not know what posterior belief will result, it is necessary to incorporate the effect of all the possible observations, $z \in \mathcal{Z}$, that the agent could receive. The agent therefore takes an expectation over the observations that could be obtained, in order to compute an expected value of executing a particular action without certainty of the observation that would be received. Note that h represents both the number of times the value function has been iteratively updated, as well as the horizon length that the planner has planned out to.

The process of generating value function V_h from V_{h-1} is known as a value function backup. As an example, when using a one-step value function to compute a two-step value function, we generate all possible two-step policy trees. The set Γ_2 of possible two-step policy trees $\alpha_{2,p}$ consists of all the possible actions that can be taken at the first timestep, followed by all the possible observations that could be obtained after executing that action, as well as the different possible one-step policy trees that could be subsequently executed. Many different possible one-step policy trees have to be considered since the optimal one-step value function comprises different policy trees for different parts of the belief space. More generally, at every iteration h, new sets of α-vectors Γ_h^a, with each set associated with a different action $a \in \mathcal{A}$, have to be computed from the set of α-vectors from the previous iteration, Γ_{h-1}. Formally, this is done by first generating the

intermediate sets of α-vectors $\Gamma_h^{a,*}$ and $\Gamma_h^{a,z}$, for all $a \in \mathcal{A}$, $z \in \mathcal{Z}$,

$$\Gamma_h^{a,*} \leftarrow \alpha^{a,*}(s) = R_S(s,a) \tag{2.16}$$

$$\Gamma_h^{a,z} \leftarrow \alpha_i^{a,z}(s) = \gamma \sum_{s' \in \mathcal{S}} T(s,a,s')O(s',a,z)\alpha_i(s'), \quad \forall \alpha_i \in \Gamma_{h-1} \tag{2.17}$$

With these intermediate sets of α-vectors, we can then compute Γ_h^a by taking the cross-sum over observations, such that one $\alpha^{a,z}$ is included from each $\Gamma_h^{a,z}$

$$\Gamma_h^a = \Gamma_h^{a,*} + \Gamma_h^{a,z_1} \oplus \Gamma_h^{a,z_2} \ldots \tag{2.18}$$

where the cross-sum operator $\oplus$ over two sets $A = \{a_1, a_2, \ldots, a_m\}$ and $B = \{b_1, b_2, \ldots, b_n\}$ produces a third set $C = \{a_1 + b_1, a_1 + b_2, \ldots, a_1 + b_n, a_2 + b_1, \ldots, a_m + b_n\}$.

Finally, the set of α-vectors Γ_h can be found by taking the union of Γ_h^a sets

$$\Gamma_h = \cup_{a \in \mathcal{A}} \Gamma_h^a \tag{2.19}$$

For a policy of h timesteps, the number of possible policy trees that need to be evaluated is therefore the product of all the possible actions that could be taken, the observations that could be obtained, as well as the subsequent $h-1$ timestep policy trees that could be evaluated. For a two-step planning horizon for the two state, two action, two observation problem domain, $|\mathcal{A}||\mathcal{A}|^{|\mathcal{Z}|} = 8$ possible conditional policy trees are constructed.

Equation 2.14 therefore computes the discounted sum of expected rewards that the agent can expect to receive when executing the policy $\pi_h^*(b)$ from any belief state b for the next h timesteps. A planning under uncertainty problem with finite horizon H can therefore be solved exactly, because the value iteration algorithm always provides the optimal h-step value function after every iteration. Unfortunately, by performing the value function backup according to Equation 2.14, the number of α-vectors in the optimal h-step value function is an exponential function in the number of observations with each value backup iteration: $|\Gamma_h| = \mathcal{O}(|\mathcal{A}||\Gamma_{h-1}|^{|\mathcal{Z}|})$, and therefore grows in a double-exponential fashion in relation to the planning horizon.

To reduce the computational complexity of the value backup operation, exact value function algorithms, such as those by Sondik (1971), Littman et al. (1995b), Cassandra et al. (1997) and Zhang et al. (2001), have taken advantage of the piecewise-linear and convex properties of the value function. Specifically, these algorithms exploit the fact that some of these α-vectors will never provide the largest value for any belief in the belief space, and hence can be pruned without affecting the optimal value function. Unfortunately, not all the α-vectors can be pruned, and a large number of α-vectors may still be required to represent the exact value function.

From the controls community, an exact, optimal controller exists when the plant model is described

according to Equations 2.9 and 2.10, where the observation and linear models are linear, the noise models are Gaussian, and the reward model is a quadratic function of the states. For such plants, the well-known Linear Quadratic Gaussian (LQG) (Athans, 1971) controller provides an optimal solution. However, this controller is applicable to a very restrictive model, and if the problem model does not perfectly meet the requirements, the LQG controller no longer provides an exact solution.

Exact solution techniques are therefore restricted to either problems with a very small number of states or to a very restricted set of problem models. For example, a finite-state, infinite-horizon POMDP with boolean rewards has been shown to be EXPTIME-hard — the problem is undecidable for generic bounded reward functions (Madani et al., 1999), and even finite horizon POMDPs are known to be PSPACE-hard. Furthermore, the best known bound for an exact POMDP value iteration solution is doubly exponential in the planning horizon, and singly exponential when the starting state is known. Unsurprisingly then, beyond the small number of exact solution techniques that exist (Sondik, 1971; Cheng, 1988; Kaelbling et al., 1998), the majority of the research has focused on approximate solution techniques for addressing planning under uncertainty in partially observable domains.

2.2.2 Approximate offline techniques

A wide variety of approximate techniques have recently been developed for solving sequential decision making problems in stochastic and uncertain domains. These algorithms are approximate because they generate policies for the agent without computing the exact expected value of executing each action from different beliefs. In order to obtain computational efficiencies, approximate POMDP techniques typically exploit some property of the problem domain that they are solving. Broadly, the techniques can be categorized into the following classes:

- Point-based approaches leverage inherent structure in the value function to perform value function updates for only a subset of the beliefs.
- Belief compression techniques build upon the observation that the set of reachable beliefs typically resides on a lower-dimensional belief space.
- Some techniques exploit structure in the transition and observation models of the problem domains.
- Hierarchical approaches take advantage of problem structure to decompose large POMDP problems into smaller sub-problems that can be solved independently.

Point-based approaches

A class of approximate POMDP techniques that has been popular in the literature consists of the point-based methods (Kurniawati, Hsu, & W.S., 2008). These techniques sample a set of points from the belief space

and only perform backup operations to update the value function for these belief samples, rather than for the entire belief space.

Point-based methods leverage inherent structure in the POMDP value function to obtain estimates of the value function for the entire belief space without having to perform value backups for every α-vector. The basic intuition is that by finding dominating α-vectors at a set of sampled belief points B_s, it is likely that most of the α-vectors that matter have been found. Hence, a smaller set of α vectors could be sufficient to provide a lower-bound estimate of the value function that is close to the optimal value function.

Similar to the exact approaches, the approximate point-based value function V_h is computed by iteratively refining the value function from the previous iteration V_{h-1} using the Bellman equation. However, exact POMDP techniques maintain the entire set of α-vectors that are necessary to represent the value function exactly, which usually results in an exponential growth in the number of α-vectors as the planning horizon increases. In contrast, point-based approaches sample the belief space with a set of belief points, and by maintaining a single α-vector for each belief sample, fewer α-vectors are needed to maintain a lower-bound estimate of the value function, so long as the number of belief samples maintained does not grow exponentially with the planning horizon h. By changing the number of belief samples, these algorithms also have the flexibility of trading off computational time for better precision of the lower bound.

Formally, point-based approaches perform the α-vector backups by first generating the same intermediate sets of α-vectors $\Gamma_h^{a,*}$ and $\Gamma_h^{a,z}$ as in Equations 2.16-2.17

$$\Gamma_h^{a,*} \leftarrow \alpha^{a,*}(s) = R_S(s,a) \tag{2.20}$$

$$\Gamma_h^{a,z} \leftarrow \alpha_i^{a,z}(s) = \gamma \sum_{s' \in \mathcal{S}} T(s,a,s')O(s',a,z)\alpha_i(s'), \quad \forall \alpha_i \in \Gamma_{h-1} \tag{2.21}$$

However, by operating only over a finite set of belief points B_s, point-based approaches avoid having to perform the cross-sum operation (Equation 2.18), and instead perform a summation for each sampled belief point $b \in B_s$

$$\Gamma_h^a \leftarrow \alpha_b^a = \Gamma_h^{a,*} + \sum_{z \in \mathcal{Z}} \underset{\alpha \in \Gamma_h^{a,z}}{\text{argmax}} \langle \alpha, b \rangle, \quad \forall b \in B_s \tag{2.22}$$

Finally, the best α-vector α_b for each belief point $b \in B_s$, and the corresponding set of α-vectors Γ_h is found according to

$$\alpha_b = \underset{\Gamma_h^a, a \in \mathcal{A}}{\text{argmax}} \langle \Gamma_h^a, b \rangle \tag{2.23}$$

$$\Gamma_h = \cup_{b \in B_s} \alpha_b \tag{2.24}$$

A value function estimate can then be obtained for any belief in the belief space from the set Γ_h, even if

the belief does not belong to the set of belief points B_s, according to

$$V_h(b) = \max_{\alpha \in \Gamma_h} \langle \alpha, b \rangle \tag{2.25}$$

Point-based approaches have had a relatively long history within the POMDP literature. Both Cheng (1988) and Zhang & Zhang (2001) present methods that backup the value function at specific belief points in order to speedup the value iteration process, but interleave them with standard backups across the full belief space; the need for occasional full backups still constrained these algorithms to small POMDP problems.

Led by the PBVI algorithm (Pineau et al., 2003a), more recent approaches including HSVI2 (Smith & Simmons, 2005), PERSEUS (Spaan & Vlassis, 2005), FSVI (Shani et al., 2007) and SARSOP (Kurniawati et al., 2008) have done away with the need for full value function backups. In contrast to the earlier techniques, these techniques only operate over a finite set of belief points, and therefore avoid the cross-sum operations that were previously needed for an exact value update. These algorithms differ slightly in their choice of belief samples and their approach for computing the value update. PBVI uses a heuristic exploration strategy that chooses belief samples that seek to evenly cover the belief space. HSVI2 maintains upper and lower bounds of the value function, and uses heuristics to guide the sampler to sample beliefs from regions that will reduce the gap between the upper and lower bounds of the optimal value function. FSVI uses an MDP solution to guide the sampling of beliefs, making it an effective solution technique for certain problems, but this technique may under-explore important parts of the belief space, especially when there is large problem uncertainty. Recently, the SARSOP algorithm builds on both the HSVI2 and the FSVI approaches by seeking to find the subset of belief points that is reachable under the optimal sequence of actions. It uses a heuristic exploration strategy to sample the belief space, as well as a bounding technique to focus sampling near the region of the belief space that is most likely to be visited by the optimal POMDP policy. Finally, PERSEUS modifies the α-vector backup process by backing up only a subset of the points in the sampled belief set B_s, sufficient for improving the value of each belief point in the set. The key intuition for PERSEUS is that a single backup may improve the value of many points in the set. More recently, Porta (2006) extended the PERSEUS algorithm to continuous state spaces by representing the observation, transition, reward models and the agent's beliefs as Gaussian mixtures, so that the Bellman backup can be computed in closed form. The CPERSEUS algorithm utilizes the property that the value function for a continuous POMDP is in general convex, and PWLC when the states are continuous and the action and observation spaces are discrete.

While these algorithms have been able to scale to problem domains that are much larger than those that can be addressed by exact approaches, point-based approaches still suffer from a type of computational intractability known as the "Curse of Dimensionality" (Kaelbling et al., 1998). Because a POMDP with n states has a belief space that has $n-1$ dimensions, computing a policy over the entire belief space is still a very challenging task. Specifically, point-based approaches iteratively improve the accuracy of the value function by performing successive α-vector backup operations. Each of these operations has a complexity of at least

$\mathcal{O}(|\mathcal{S}|^2|\mathcal{Z}|)$, making it computationally demanding when the environment has many states. Furthermore, even if the state space is factored, point-based offline approaches are typically unable to take advantage of the independencies between the state dimensions, as in general the value function cannot be factored (Kearns & Koller, 1999).

More importantly, many of the most successful point-based approaches obtain good performance in large POMDP domains by using heuristics that channel the sampling of beliefs towards particular regions of the belief space. Unfortunately, these heuristics may not always guide the algorithm to sample beliefs in the right regions, and as our experimental results in Chapters 4 and 5 will illustrate, selective sampling of the belief space makes it particularly difficult to solve problems that require long information-gathering action sequences to be executed before good performance can be achieved.

Belief compression

Another class of approximate POMDP techniques seeks to directly address the "curse of dimensionality" by reducing the dimensionality of the belief space. Although the original dimensionality of the belief space is equivalent to the number of states in the POMDP problem domain, it has not only been observed that much of the belief space is typically unreachable, but more importantly, that the reachable beliefs also reside on a much smaller dimensional belief space, which can be represented by a much smaller set of parameters than the full belief space. Since an important reason for the computational complexity of many POMDP problem domains is the high dimensionality of the belief space, much more efficient algorithms can be developed if we are able to obtain the lower-dimensional belief space and plan within this reduced space without misrepresenting the reachable beliefs. Unfortunately, it is also often observed that the value function is no longer piecewise-linear and convex in the lower-dimensional belief space, and thus these algorithms often have to perform fitted value iteration over the reduced belief space.

This lower-dimensional belief representation takes different forms. The Augmented MDP (AMDP) (Roy & Thrun, 2000) algorithm approximates beliefs by using only the mean and entropy of the agent's belief to represent the belief space. Discretizing the mean and entropy values into a fixed number of bins, the algorithm then performs fitted value iteration on the much reduced belief space in order to compute a policy in an efficient manner. For problem domains where the set of reachable beliefs fit this restrictive belief representation, such as the problem of coastal navigation, the algorithm scales well to domains with a large number of states. However, by planning only with the mean and entropy parameters of the agent's beliefs, the AMDP algorithm restricts itself to representing uni-modal beliefs, and is unable to adequately represent beliefs that have multiple modes. Multi-modal beliefs are common in many planning under uncertainty problem domains, thereby limiting the applicability of this approach.

Brooks et al. (2006) use a similar belief compression approach for robot navigation tasks. The parametric POMDP (PPOMDP) algorithm represents the agent's belief as a two-dimensional diagonal Gaussian, which requires four statistics to be tracked — two mean values and the two diagonal covariances. The algorithm

uses a fitted value iteration approach that is similar to (Roy & Thrun, 2000), and is also unable to accurately represent multi-modal distributions, which could occur in robot navigation tasks when the agent navigates in an environment with two rooms that look identical. In addition, in order to restrict the belief representation to these four statistics, the observations are only used to provide information of the agent's position along the two principal axes.

Recently, Zhou et al. (2008) built on the previous two approaches and used density projection to reduce the dimensionality of the belief space. The algorithm projects the high dimensional belief space onto a lower-dimensional family of parametric distributions by minimizing the Kullback-Leibler (KL) divergence between the belief state and its projection. If the beliefs are being projected onto the exponential family of distributions, these projections can be computed analytically. As in the case of the AMDP or PPOMDP algorithms, the lower-dimensional parameter space can then be discretized and solved using fitted value iteration. By projecting onto the exponential family of distributions, this approach allows a much broader range of beliefs to be expressed, though the fact that each parameter of the low-dimensional belief representation still has to be discretized for fitted value iteration backups means that the number of states will still scale exponentially with the number of parameters.

More generally, while the reachable set of POMDP beliefs may lie on a much lower-dimensional manifold, these beliefs may not belong to any particular parametric distribution — an assumption made by the belief compression techniques presented above. Instead, the exponential family principal components analysis (Roy & Gordon, 2003) algorithm uses a variant of the principal components analysis dimensionality reduction technique to learn a small number of belief features that decomposes into a reduced dimensionality space. Because the POMDP belief manifolds are rarely linear, the algorithm uses the exponential principal components analysis (E-PCA) approach, whereby an exponential link function transforms the belief space into one that allows for linear dimensionality reduction, as well as to ensure that the resultant probability distributions are strictly positive. The algorithm then plans over the low-dimensional space by discretizing the features and performs value iteration on the beliefs in the reduced space.

In Chapter 5 of this thesis, we adopt a belief compression technique that is similar to the AMDP approach, restricting the agent's belief to be a uni-modal Gaussian distribution. In Chapter 7, we accommodate the need for a more general belief representation by using multi-modal Gaussian beliefs, significantly increasing the expressive power of our algorithms. In addition, as we do not use a value iteration approach, the parameter values do not need to be discretized, thereby avoiding the exponential growth in the number of states with an increase in the number of parameters.

Exploiting model structure

Recently, a number of POMDP techniques have resorted to exploiting model-specific structural properties in order to obtain approximate scalable algorithms for POMDPs with large state spaces. These techniques can alternatively be seen as seeking to restrict the problem domains within which these techniques are applicable,

in order to obtain greater computational efficiencies for greater applicability.

An important area of POMDP research has been to take advantage of problem domains that have factored representations, whereby for any timestep, the transition, observation and reward models of the problem domain depend only on a subset of the state variables. When the state space is well-factored, the α-vectors can be computed directly in factored form, thereby taking advantage of the structured representation of the value function. However, it is still difficult to represent the agent's belief, because in a partially observable setting, even if certain variables are initially independent, after a few timesteps, the variables could become correlated over time, making the compact representation invalid. Boyen and Koller (1998) approximate the belief state by projection in order to effectively utilize the factored representation of the state spaces, effectively breaking the joint distribution of the belief by marginalizing over subsets of variables, discounting dependencies among the variables, and providing a bound on the KL-divergence of the approximate belief states. However, translation of these bounds into decision quality error for POMDPs yield weak bounds.

Similarly, Hansen and Feng (2000) use algebraic decision diagrams (ADDs) to represent the beliefs, and they show that when applied to POMDPs, reasonably large problems can be solved. However, these approaches require that the problem domain exhibits a significant amount of structure such that an ADD representation will be useful. Recently, ADD representations have been combined with point-based approaches, including Symbolic PERSEUS (Poupart, 2005) which combines ADD operations with the PERSEUS solver. Similarly, Sim et al. (2008) combined ADDs with HSVI. As point-based approaches, these algorithms are guaranteed to be lower bounds on the value functions if the ADD operations are performed exactly.

In this thesis, we also address problems with particular problem structure in order to solve problem domains with much larger state, action and observation spaces. As with all techniques, the exploitation of model structure implies that our approach will not always be applicable to all problem domains.

A different form of structure exists in the observation model of some problem domains. When computing a POMDP policy, the planner has to incorporate all possible observations that the agent could receive, since the agent is unable to anticipate which observations it may receive during the execution of the policy. For problem domains with a large or continuous observation space, computing a policy becomes computationally intractable, and researchers have therefore sought to reduce the size of the observation space. For example, Hoey et al. (2005) present dynamic partitioning techniques for finding observation equivalence for different classes of observation space. Using the intuition that observations are only useful to the extent that they lead to different courses of actions, this algorithm seeks to find a lossless partition of the observation space while computing a policy, rather than making an *a priori* discretization of the observation space. In a one-dimensional observation space, the algorithm uses numerical solutions to find the boundaries of the observation set that results in the same action. With arbitrary multi-dimensional observations, the algorithm samples the observation space to build an approximate discrete observation function for performing point-based backups. Unfortunately, as the authors are quick to acknowledge, their dynamic partitioning technique is tightly integrated with point-based backup approaches such as PBVI and PERSEUS.

Atrash et al. (2006) adopt a different strategy for addressing large observation spaces, seeking to find low-dimensional observation spaces to reduce the planning complexity. They propose two algorithms that reduce the dimensionality of the observation space — an EM clustering algorithm to find a small set of summary observations that are used for planning, as well as a modified principal component analysis to reduce the dimensionality of the observation space. However, Atrash et al. are seeking to find observations that are equivalent at the "global" level, i.e., observations that result in the same policies regardless of the belief states they were observed from. Such observation equivalences are often uncommon in many problem domains.

Finally, Carlin et al. (2008) use a value-based observation compression technique to prune the least valuable observations while maintaining an error bound on the loss in value. They calculate the expected value lost by merging two observations, implicitly considering the probability of witnessing an observation. The focus here is therefore to actively generate a sub-optimal policy in order to reduce the computational complexity resulting from a large number of possible observations.

As will be discussed in Chapter 3, our semi-conditional planning approach seeks to exploit a similar property of the observation function. Rather than finding partitions of the observation space such that the agent should execute the same action for observations that belong to the same partition, our semi-conditioning approach assumes that there are regions of the belief space where all observations result in the same next action. Computational savings can then by obtained by not having to consider the possible observations for these regions of the belief space.

Hierarchical approaches

Another class of POMDP techniques builds upon the intuition that certain problem domains can be decomposed into smaller sub-problems that can be solved separately. If the overall task in the problem domain can be mapped into a hierarchy of subtasks, then rather than try and compute a policy over the entire state space, a hierarchical planner can take advantage of the structure and solve the individual subtasks separately.

The HPOMDP (Theocharous, 2002) formulation extends the Hierarchical Hidden Markov model by adding associated actions and rewards. Temporally-extended policies are designed or learnt for the exit state of each abstract state. The general principle of HPOMDP is similar to some of the techniques in the fully observable MDP approach, such as MAXQ (Dietterich, 2000) and ALisp (Andre & Russell, 2002), seeking to subdivide the state space into a tree of smaller planning problems. HPOMDP assumes that the large POMDP problem can be sub-divided into smaller POMDPs. However, in extending the fully-observable approaches to solve POMDPs, the HPOMDP approach continues to assume that for each subtask, the agent has full observability of both the start and end states, such that the transition between subtasks is well-defined. Such an assumption is brittle for general POMDP problems. Similar POMDP approaches have been proposed by Hernandez-Gardiol and Mahadevan (2001), Theocharous et al. (2001) and Wiering and Schmidhuber (1997), though these techniques all make the significant assumption that the planner is able to fully observe the completion of the sub-problems.

To address the difficulty in defining the start and termination conditions in state space for a POMDP problem, PolCA+ (Pineau & Thrun, 2002) is a hierarchical planning and execution algorithm that uses action abstraction to exploit hierarchy in a problem domain. PolCA+ uses a structural decomposition of the task/action space, taking advantage of the notion that in problems such as dialogue-management and robust robotics control, not every action is relevant to each sub-task. By limiting the set of actions that can be applied for each subtask, the set of states and observations that will be encountered in each subtask is also naturally limited. The name of the algorithm, Policy Contingent Abstraction, derives from the fact that the abstraction at the higher level is contingent on the policy for the subtasks at the lower level. The focus here is on reconstituting a complex policy by combining many sub-policies together, thereby enabling the algorithm to ignore having a subtask termination condition, which would have been difficult to adhere to in a POMDP setting. However, the algorithm requires a domain expert to provide the structural decomposition beforehand, and sub-reward functions may have to be specified in order to perform planning within the sub-problems. Furthermore, it assumes that there is a natural decomposition in the action space, while the value functions computed by PolCA+ are not guaranteed to be upper or lower bounds on the optimal policy.

Hansen and Zhou (2003) propose a different method for exploiting hierarchy using hierarchical controllers that exploit a user-specified hierarchy for planning. Kaelbling et al. (1998) point out that an optimal policy for a finite-horizon POMDP can be represented by an acyclic finite-state controller, where each machine state corresponds to a vector in a non-stationary value function. Hierarchical controllers extend the notion of finite state controllers by letting some nodes be sub-controllers, and defining certain special exit nodes in order to get in and out of sub-tasks. The policy is then optimized in a bottom up fashion, resulting in a more efficient process as there are fewer nodes to be optimized per sub-task.

Previous methods require that a hierarchy be provided to the planner in order to plan in a hierarchical manner. Charlin et al. (2008) extend the previous approach by providing a method for automatically discovering inherent hierarchy in the problem domain. However, the algorithm is no longer able to perform bottom up optimization, but must simultaneously optimize the policy at all levels of the hierarchy. Ironically, this may mean that the solver is no longer able to take full advantage of the hierarchical structure to reduce computational complexity.

The hierarchical techniques presented in this section may appear similar to the ideas presented in this thesis, in that both classes of techniques seek to abstract away parts of the problem so that a large planning under uncertainty domain becomes manageable. However, while these hierarchical controllers seek to exploit the structural decomposability in the problem, our approach does not seek to generate sub-tasks in the problem, but rather seeks to exploit particular structure in the reward function in order to perform planning to a long horizon. In addition, in Chapter 4, we avoid relying on domain experts to reveal problem structure, and instead provide a method for automatically discovering inherent structure in the problem.

Algorithm 1 Generic Forward search framework

Require: Initial belief b_0, Expansion depth $H_D \leq H$

1: $t \leftarrow 0$
2: **loop**
3: Initialize tree T with b_t at the root
4: **while not** PLANNINGTERMINATED() **do**
5: $b_E \leftarrow$NEXTNODETOEXPAND()
6: EXPAND(b_E, H_D)
7: UPDATEANCESTORS(b_E)
8: **end while**
9: Execute best action a_t for belief b_t
10: Obtain new observation z_t and reward r_t
11: $b_{t+1} = \tau(b_t, a_t, z_t)$
12: $t \leftarrow t + 1$
13: **end loop**

2.2.3 Online forward search

The POMDP techniques that have been presented above are traditionally referred to as offline algorithms, since they seek to compute a policy over the entire belief space before the agent commences the execution of any action. The policy computed offline provides the agent with an action to execute at every timestep, for all possible beliefs. Even the point-based methods that compute the value function at a finite set of beliefs assume that the finite set can adequately represent the policy across the whole belief space.

Unfortunately, in problem domains with a large number of possible states, actions and observations, it is difficult to exactly compute or approximate an optimal strategy for every possible belief. Computing a policy for the entire belief space may also be unnecessary, since the majority of the belief space may be unreachable. Instead, an alternative search technique focuses computational effort only towards belief states that are reachable from the current belief under different actions. These techniques compute a policy online and determine the next action to take by focusing on the agent's current belief. After executing the selected action and receiving an observation, the agent can update its belief, and the planner can then repeat the search for the next best action for the agent's latest belief. Such approaches are known as online planners, since the belief for which a plan is needed can only be identified online. As a result, they alternate between the policy computation and policy execution components. Typically, online planners replan after every timestep using the agent's most updated belief, though it is also possible for some planners to only replan periodically. Replanning only after every few timesteps allows a planner to forgo the ability to incorporate the latest observation obtained at every timestep, in exchange for searching out to a longer horizon when generating a policy.

Ross et al. (2008) provide a recent survey of online planners in the literature, as well as a generic forward search approach, which we reproduce here (Algorithm 1). During each planning phase, the algorithm first initializes a forward search tree with the agent's current belief b_t at the root (line 3). This forward search tree is then constructed by computing the set of beliefs that are reachable from the agent's current belief

within H steps, where H is the planning horizon (refer to Figure 1-3 in Chapter 1). Reachable beliefs are computed by first choosing a possible action that the agent can execute from the current belief node, followed by an observation that could be received after that action is taken. The forward search planning process then involves repeated execution of three sub-routines:

1. Selecting the fringe node under which the tree should be further expanded (line 5)
2. Constructing the set of reachable beliefs for a pre-determined expansion depth H_D and evaluating the value function at each of the new nodes (line 6)
3. Propagating the new values up the tree by taking the maximum expected value over the actions at each intermediate belief node, and an expectation of the expected values over each observation at each action node (line 7).

When evaluating the value function at each of the new nodes, a heuristic function is often first used at each of the leaf nodes to estimate the expected value at the leaves, thereby guiding the online search algorithm. These values are then propagated up the tree. The planner chooses the maximum value amongst the actions when propagating the value up the tree because it is able to choose which action to take next, whereas it has to take an expectation over the observations because it is unable to anticipate which observations it may receive during execution. The planning phase continues until some terminating condition is met, such as when a maximum depth is reached or when no more computational time is available for planning.

During the execution phase, the agent executes the best action that had been found during the planning phase for the root belief node b_t. The agent then obtains an observation z_t from the environment, and computes a new belief b_{t+1} according to Equation 2.3. The agent then creates a new forward search tree from the new belief b_{t+1}, and the cycle repeats.

Using this forward search approach, online planners have demonstrated the ability to compute policies for large problem domains, such as a sub-problem of the RoboCupRescue domain (Paquet et al., 2005). Nevertheless, although forward-search techniques can be substantially faster than computing a full policy, the computational cost of creating a full H-horizon forward search tree remains an exponential function of the action branching factor $|\mathcal{A}|$ and observation branching factor $|\mathcal{Z}|$. This restricts the horizon that one can plan out to, especially if there are real-time constraints that have to be met in order to perform online planning.

Existing techniques in the literature vary based on the methods used to limit the space of reachable beliefs that need to be explored to compute a good policy, with methods ranging from branch-and-bound pruning (Paquet, Chaib-draa, & Ross, 2006), Monte Carlo sampling (Bertsekas & Castanon, 1999; McAllester & Singh, 1999) and heuristic search (Washington, 1997; Ross & Chaib-draa, 2007). With reference to the general forward search framework presented in Algorithm 1, these techniques differ in the NEXTNODETOEXPAND, EXPAND and UPDATEANCESTORS subroutines. Unfortunately, existing forward search techniques typically scale poorly with the horizon length, and are unable to search out to longer horizons for a broad class of sensor, measurement and cost/reward models.

A variation of the forward search strategy that does not construct a forward search tree is proposed by Geffner and Bonet (1998), which discretizes the belief space and learns approximate values for the belief states visited by performing successive trials in the environment. While this technique may be useful for domains where a factorized representation is available, since the discretization can be applied separately, in general, the algorithm needs substantial amounts of memory to store all the learned belief state values. The number of discrete beliefs grows exponentially with the size of the state space, and hence an infeasibly large number of trials may be needed to learn good estimates for such a large number of beliefs.

The forward search framework is popular across multiple research communities (as we will see in the next few sections), further suggesting the applicability of this approach on real-world planning problems that often have large state spaces. In this thesis, we adopt the forward search approach for performing planning under uncertainty, though we seek to address the shortcomings of forward search using techniques described in the following chapters.

2.3 Model predictive control/receding horizon control

The controls community has developed a large variety of algorithms that deal with many different types of uncertainty, including model uncertainty, state uncertainty, control uncertainty, etc. In this section, we focus our attentions only on those techniques that directly relate to the algorithms presented in this thesis.

Following Section 2.2.3, another class of online, forward search approaches exists within the controls community, and is known interchangeably as the class of Model Predictive Control/Receding Horizon Control (MPC/RHC) techniques. The term model predictive control refers to the technique of assuming an explicit plant model and solving a finite horizon open-loop optimal control problem in an online fashion, using the current state of the plant as the initial state. The first few controls in this sequence are applied to the plant, and the planning process is repeated whenever new information is available for updating the parameters of the predictive model (Rawlings, 2000). Similarly, the receding horizon control approach assumes that it is computationally intractable to optimize the given reward function out to an infinite horizon, and hence focuses on solving the optimization problem out to a shorter, more limited horizon. The RHC model also assumes that a heuristic exists to provide a rough estimate of the reward beyond the limited horizon, and that the planner will replan before the limited horizon is reached.

Algorithms that adhere to the MPC/RHC model adopt the general approach of solving a finite-horizon optimal control problem for the current state/belief, executing the first few controls, and then repeating the optimization based on the latest information available. MPC/RHC algorithms therefore optimize over the forward search space by anchoring the initial condition of the optimization on the current state/belief. These techniques commonly optimize their conditional plans for a short, finite horizon, and use a heuristic to estimate the cost beyond the fixed horizon (Kuwata & How, 2004; Bellingham, Richards, & How, 2002). They are therefore similar to the forward search approaches that search out to a finite planning horizon, thereafter

using a heuristic function to estimate the expected value beyond the planning horizon.

The MPC/RHC model has been used in numerous industrial applications with great success, such as optimizing the process models in chemical industrial plants (Rawlings, 2000; Garcia, 1984). MPC has also been used to design controllers for motion planning tasks. Kuwata and How (2004) present a trajectory planning algorithm for a vehicle flying in 3D environments, using a linear program to optimize a detailed trajectory out to a limited horizon, and using a cost-map to estimate the cost-to-go beyond the limited horizon.

More directly related to the problem of planning in partially-observable environments, researchers within the MPC community have developed algorithms that are robust to various types of uncertainty and model errors (Rawlings, 2000). Dealing with robustness is challenging for MPC algorithms because of the open-loop nature of the optimal control problem, even though the receding horizon component provides an implicit feedback. Richards and How (2005) address the challenge of operating robustly in problem domains where imperfect information may result in an unknown but bounded state estimation error. Their controller is able to estimate bounds on the measurement and disturbances uncertainties, and can also adapt its robustness quality based on desired bounds of the prediction error. Badgwell (1997) ensures robustness to uncertainty in the plant parameters by appending a set of robustness constraints to the open-loop optimal control problem, while Kothare (1994) proposes the technique of optimizing over the state feedback gains, rather than the open-loop control sequence, in order to optimize the subsequent feedback response to handle uncertainties in the plant model.

On a similar note, Blackmore (2006) focuses on chance-constrained optimal planning, where the focus is on finding plans where the probability of failure is below a certain threshold, rather than incorporating the failure cost into the reward and seeking to maximize the expected reward. The algorithm uses constrained optimization to solve chance-constrained stochastic control problems, posing the stochastic predictive control problem as a Mixed Integer Linear Program. However, not all problems are easily representable as MILPs. In addition, the algorithm approximates the original stochastic problem by using a particle representation of the agent's belief; such an approach may be computationally expensive if a large number of particles is required to find optimal solution.

Therefore, in contrast to the online forward search techniques presented earlier, MPC techniques typically optimize the short-horizon control sequence without seeking to directly evaluate the set of states or beliefs that are reachable within the planning horizon. Instead, these techniques assume that the plant model at every re-planning step can be approximated with a model of a specific structure, thereby enabling the planning algorithm to compute a globally optimal control policy over a limited horizon. Conversely, this assumption also implies that each MPC algorithm is typically only applicable to a very restricted class of problem domains.

2.4 Strategies from the sensor resource management community

In the sensor resource management domain, planning under uncertainty techniques are used in the context of planning sensor placements to track single or multiple targets. As discussed in Section 2.1.4, much of this community's research focus has been on modeling the agent's sensor and dynamics models, rather than on planning in partially observable environments.

Nevertheless, amongst the planning algorithms that exist, researchers here have focused on generating approximate solution techniques by taking a forward search approach, due to the computational complexities involved in seeking to compute a policy over the entire state space even for problems of moderate size. Existing algorithms often adopt a myopic, or greedy strategy when it comes to planning, and recently, some have focused on analyzing the subclass of problems where choosing the action with the most immediate reward is close to optimal (Krause & Guestrin, 2007).

Nevertheless, other researchers, including Scott et al. (2009) and Kreucher et al. (2004), have proposed non-myopic strategies for target-tracking. Kreucher et al. describe a multi-target tracking problem and demonstrate the need for non-myopic sensor management in order to obtain good performance. The authors use a particle filter approach to represent the agent's belief of the target's location, and seek to find paths that will result in the greatest KL divergence in density before and after the measurement. To look ahead more than one action, the algorithm uses Monte Carlo sampling to generate possible observation outcomes. They also provide an information-directed path searching scheme to reduce the complexity of the Monte Carlo sampling, as well as problem-specific value heuristics that will help direct the search. While it is undeniable that non-myopic strategies are especially valuable for complicated problem domains, they can also result in computationally expensive algorithms that have restricted applicability. Specifically, Monte Carlo sampling approaches may be an unattractive option if a large number of samples are required for good performance. In Chapters 5-7, we propose algorithms that seek to reduce our dependency on Monte Carlo sampling.

Scott et al. (2009) directly formulate the target tracking problem as a POMDP, and propose the Nominal Belief Optimization (NBO) algorithm that computes the most likely belief after an action for deeper forward search. This approach is similar to the algorithm we present in Chapter 5, but instead of only computing the most likely belief, our algorithm explicitly computes the entire set of possible posterior beliefs after each macro-action.

Choi and How (2008) addresses the problem of planning paths for mobile sensors in order to improve long-term forecast performance. It uses mutual information between the measurement and a subset of the future state as the performance metric, formalizing a method for quantifying information obtained along a continuous measurement path. Finally, Bertuccelli and How (2006a, 2006b) address target-searching and target-tracking problems by using uncertain probability maps to represent the uncertainty in the probability of a target being at different cells in a map. By using Beta distributions to represent the probability uncertainty of each cell, the state transition and measurement updates can be computed in closed-form. In Chapters 5-7, we also propose algorithms that seek to analytically compute the posterior beliefs after the belief updates.

2.5 Summary

In this chapter, we have presented a unified framework for planning under uncertainty in partially observable domains, bringing together a variety of problem formulations from different research communities. We have also presented different approaches that currently exist for addressing this problem, and show that despite the progress that has been made, existing approaches still have difficulties solving large problems that require long horizon planning. Specifically, problems that not only have a large number of states, actions and observations, but also require planning to a long horizon have thus far not been solvable. In subsequent chapters, we address this issue by introducing the notion of semi-conditional planning, and present new algorithms to address this challenge.

Chapter 3

Semi-conditional planning with macro-actions

The previous chapter reviewed past research on planning under uncertainty in partially observable domains. Though some of the techniques presented have demonstrated promising performance when tackling sequential decision making under uncertainty, existing planners still face difficulty solving many real-world sized partially observable problems, especially those that require planning out to a long horizon.

In this chapter, we present a new approach for sequential decision making, which we term *semi-conditional planning*. This approach seeks to plan out to a long horizon in partially observable domains, while balancing the need to search deeply with the contrasting need of conditioning the agent's actions on the observation obtained at previous timesteps. We use forward-search planning as presented in Section 2, but rather than determine the agent's next action by generating conditional plans that branch on the observations received after *every* action, we hypothesize that actions can be chosen by only conditioning on the received observations at certain key points along the horizon. Between these key points, we assume that a temporally-extended, fixed length action sequence is executed in an open-loop fashion. By reducing the number of conditional branches, the planner is able to search to a longer horizon while meeting the real-time constraints of forward search approaches.

We first review techniques that have explored the use of temporally-extended action sequences for planning from both the fully-observable and partially observable planning literature. This review will allow the reader to better understand why we have adopted a particular approach for using macro-actions in this thesis. We then present a generic framework for semi-conditional planning in partially observable domains; this framework provides the baseline architecture for the semi-conditional planning variations that will be presented in subsequent chapters of this thesis. Cognizant of the fact that semi-conditional planners may not be useful in every problem domain, we also explore the conditions under which semi-conditional planning is effective for planning under uncertainty in partially observable domains. Finally, we describe some of the

new technical challenges that need to be overcome in order to perform semi-conditional planning effectively; these challenges motivate the semi-conditional planners that we present in subsequent chapters.

3.1 Fully-conditional, unconditional and semi-conditional planning

Most existing planners for partially observable domains construct a policy where each action in the policy is conditioned on the observations that are obtained at each timestep. Since the planner conditions the agent's actions on the possible observations received at every timestep out to a finite horizon when constructing the policy, we describe these planners as *fully-conditional* planners. Unfortunately, fully-conditional planning is computationally expensive. The space of conditional plans grows in a doubly exponential fashion with the number of timesteps the agent must look ahead in order to select the best action, and is therefore particularly challenging when good control performance requires considering situations far into the future.

At the other extreme, a planner could completely ignore the observations received during execution, and plan assuming that the agent does not incorporate any of the observations when performing a belief update. We describe these planners as *unconditional* (or "open-loop") planners, since the agent does not consider the different observations that could be received when it estimates the expected value of taking a particular action at the next timestep, even if it actually incorporates the observation received when it subsequently performs a belief update. Open-loop planning can be extremely fast and perform surprisingly well in certain domains. For example, Yu et al. (2005) use open-loop planning for multi-robot tag. In addition, the QMDP (Littman, Cassandra, & Pack Kaelbling, 1995a) approach could also be said to be a form of open-loop planning, since it assumes that the problem will be fully observable once the first action is executed. Nevertheless, acting well in most real-world domains requires plans where at least some action choices are conditional on the future observations.

This thesis focuses on the significant subset of POMDP domains, including scientific exploration, target surveillance, and chronic care management, where it is possible to act well by planning using conditional sequences of open-loop, fixed-length action chains, or "macro-actions." We call this approach *semi-conditional* planning, in that actions are chosen based on the received observations only at the end of each macro-action, rather than after every primitive action. In between these branching points at the end of each macro-action, the planner assumes that the primitive actions are executed without considering the different possible observation sequences that could have been obtained.

A semi-conditional planner therefore spreads out the points where the planner conditions the agent's subsequent actions, such that the branching points now span over a larger number of timesteps. Although the computational complexity of a semi-conditional planner is still doubly exponential, the complexity grows in a doubly exponential fashion with the number of successive branching points, rather than with the number of timesteps.

Therefore, if the macro-actions have length greater than one, and if only a subset of the admissible action

seqeunces and possible observation sequences are evaluated, then relative to a fully-conditional planner, the semi-conditional planner will be able to plan to a much longer horizon for the same amount of planning time.

3.2 Planning with macro-actions

In this thesis we approach the challenge of planning far into the future by adopting macro-actions (Sutton et al., 1999), which we define as fixed-length, open-loop policies. In other words, a macro-action is a finite sequence of primitive actions that is executed without regard to the observations received during the execution of the action sequence.

3.2.1 Options

For applicability to partially observable worlds, our macro-action definition is more restrictive than the broader class of macro-actions that have been used within the reinforcement learning community known as "options", first proposed by Sutton et al. (1999). Options generalize the set of actions that can be considered to include temporally extended courses of actions, allowing the planner to choose closed-loop sub-policies for taking actions over multiple timesteps. Options are often discussed as part of the fully-observable semi-Markov Decision Process (SMDP) (Puterman, 1994; Mahadevan & Khaleeli, 1999) planning framework. SMDPs extend the MDP problem formulation to allow for the modeling of continuous-time, discrete-event systems, and can therefore model actions that take variable amounts of time, including temporally-extended action sequences. Existing SMDP algorithms demonstrate how the transition probabilities and rewards associated with these temporally-extended action sequences can be calculated (Sutton et al., 1999), as well as how dynamic programming techniques can be extended to SMDPs (Howard, 1971).

Within the SMDP framework, options are temporally-extended sub-policies that can be regarded as black-boxes when generating the overall policy. Generally, an option consists of three components: a policy $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$, a termination condition $\beta : \mathcal{S} \rightarrow [0, 1]$, and an initiation set $\mathcal{I} \in \mathcal{S}$. The initiation set contains the set of possible states within which the agent can execute that particular option, i.e., an option $< \mathcal{I}, \pi, \beta >$ is available in state s if and only if $s \in \mathcal{I}$. Actions are then selected according to the option's policy, π, with a pre-determined probability β of terminating at every timestep. When the option terminates, the agent then has the opportunity to choose another option to execute. By having an initiation set and a termination condition, an additional advantage is that the option's policy now needs to be defined only over a subset of the state space. For planning in a fully-observable world, where there may be uncertainty in the outcome of the agent's actions but the state is always fully observable, options enable temporally-extended actions to be used for planning in a natural way.

The type of option defined above is known as a *Markov option*. Both the policy and termination condition are Markov, in that they are dependent only on the current state of the MDP, as opposed to other variables such as the time elapsed since the option was initiated. A more general type of option, known as a *semi-*

Markov option, allows the planner to define options that terminate after a pre-defined number of timesteps, enabling policies and termination conditions to be dependent on the history of actions and states, rather than just on the immediate prior state.

Over the past decade, a large body of literature has emerged for using options within the SMDP framework. It is beyond the scope here to review that body of work exhaustively, but different algorithms have focused on generating policies over options (Sutton et al., 1999; Precup, 2000), learning the set of options without relying on prior knowledge of the task (Perkins & Barto, 2003), as well as learning the subgoals in the problem domains that will help facilitate the overall goal of the task (McGovern & Barto, 2001; Precup, 2000).

Nevertheless, hierarchical SMDP algorithms typically assume that each option persists until a termination state is achieved. Unfortunately, such policies are not tenable in partially observable environments, since in general the agent will only have a probability distribution over whether the agent has reached the terminal state, resulting in a terminal condition that is not well-defined.

It may be possible to extend the concept of options to the partially observable domain by defining the initiation and termination conditions over the belief space, as opposed to over the state space. With this modification, different options will be admissible for different portions of the belief space, and these options will terminate with certain probabilities when the agent's belief belongs to a subset of the belief space. Unfortunately, it is non-trivial to compute the posterior belief or expected reward for a "partially observable" option. Furthermore, it is not immediately obvious how a domain expert should define such options, nor whether these partially observable options can be automatically generated.

We therefore leave further exploration of partially observable options to future work, and in this thesis, we assume the use of fixed-length open-loop policies, which can be easily applied to partially observable domains.

3.3 Semi-conditional planning under uncertainty

We plan in a forward-search manner when performing our semi-conditional planning under uncertainty, for reasons described in Section 2.2.3. Specifically, because at every timestep we are primarily interested in choosing the next immediate action for the agent, it is sufficient for the planner to focus computational effort only on those beliefs that are reachable from the agent's current belief; this sufficiency assumes that the planner can replan fast enough every timestep after incorporating the latest information available. In addition, forward search algorithms are able to plan directly with factored dynamics and observation models without having to reconstruct the non-factored models. These two advantages enable forward search approaches to address planning under uncertainty problems with large state, action and observation spaces that better mirror real-world problems.

To obtain a set of reachable beliefs, naïve forward search algorithms consider all possible action-observation

sequences of a specific length, or depth in the search horizon. The computational cost of this procedure scales as $\mathcal{O}((|\mathcal{A}||\mathcal{Z}|)^H)$, where H is the primitive-action forward search planning horizon, or depth of the search. Unfortunately, reasonably large discrete or continuous action and observation sets (large $|\mathcal{A}|$ or $|\mathcal{Z}|$, respectively) severely limit the maximum search depth achievable in real-time. Instead, by restricting our action space to a set of length L macro-actions, we can reduce the computational complexity due to action branching from $|\mathcal{A}|^H$ to $|\tilde{\mathcal{A}}|^{\tilde{H}}$, where $\tilde{\mathcal{A}}$ is the set of length L macro-actions, and $\tilde{H} = \frac{H}{L}$ is the macro-action horizon, or the number of branching points between the root and leaf belief nodes. In general, $|\tilde{\mathcal{A}}|$ is much smaller than $|\mathcal{A}|^L$, and hence this restriction of the action space results in significant computational savings, due to the smaller exponent in the computational complexity.

Algorithm 2 Forward Search with Macro-Actions

Require: Initial belief b_0, Discount factor γ, Macro-action search depth $\tilde{H}$, Sampling parameter N_s

$t \leftarrow 0$
loop
  Compute set of macro-actions $\tilde{\mathcal{A}}$ for b_t
  for each macro-action $\tilde{a}^i \in \tilde{\mathcal{A}}$ **do**
    $Q(b_t, \tilde{a}^i) = \text{MACROEXPAND}(\tilde{a}^i, b_t, \gamma, \tilde{H}, N_s)$
  end for
  Execute first action $\hat{a}$ of $\tilde{a} = \text{argmax}_{\tilde{a}}\, Q(b_t, \tilde{a})$
  $a_t \leftarrow \hat{a}$
  Obtain new observation z_t and reward r_t
  $b_{t+1} = \tau(b_t, a_t, z_t)$
  $t \leftarrow t + 1$
end loop

Algorithm 2 presents a general algorithm for using macro-actions for planning under uncertainty with forward search. This algorithm bears strong resemblance to the traditional forward search algorithms presented in Chapter 2 (Algorithm 1). Given an initial belief state, the planner calculates the expected future reward of taking each macro-action through the MACROEXPAND subroutine. The set of macro-actions can be fixed for all beliefs, or different macro-actions can be used at different belief states. In addition, there are many problems where we can take advantage of some knowledge of the kinds of macro-actions that can be used, or have a domain expert generate the set of macro-actions [1].

Starting from the agent's current belief b_t, and for each macro-action $\tilde{a}$, a macro-action forward-search tree is constructed by considering the potential future action and observation trajectories from b_t. The best macro-action $\tilde{a}_{opt}$ is the macro-action that has the highest expected future reward. To evaluate the expected reward of a macro-action, note that a macro-action $\tilde{a}$ of length L will generate a sequence of L beliefs $[b_0, \ldots, b_t, \ldots, b_L]$ when executed. Each belief will have an expected immediate reward $R_{B,t}$, where $R_{B,t} = \sum_s R_S(s, a_t) b_t(s)$, and a cumulative reward $\tilde{r}_t$, where $\tilde{r}_t = \sum_{d=0}^{t} \gamma^d \sum_s R_S(s, a_t) b_t(s)$. If the belief sequence for a macro-action were deterministic, then the posterior belief b_L at the end of the macro-action would be known exactly, and the value of the macro-action would be the (discounted) sum of expected im-

[1] In problems where a domain expert is unavailable, in Chapter 4 we will also provide an algorithm that automatically generates macro-actions.

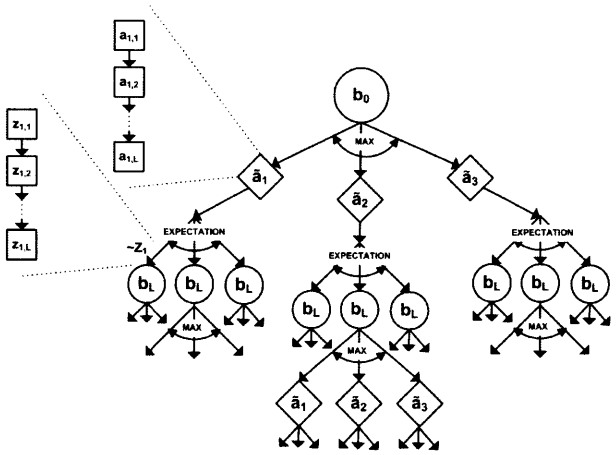


Figure 3-1: Macro-action forward search tree. Macro-actions are conditioned on received observation sequences only at the end of each macro-action.

mediate rewards $\tilde{r}_L$ added to the value of the best subsequent macro-action. The value of the best subsequent macro-action would be found by repeating the process recursively from the posterior belief b_L.

However, the sequence of beliefs for a macro-action is *not* deterministic but rather is a function of the observations received during the execution of $\tilde{a}$. As a result, the value of a macro-action is the expected value taken over all possible belief sequences, according to the probability of each belief sequence given by the corresponding observation sequence. At the end of each macro-action, for each posterior belief b_L that results from each observation sequence, we recursively compute the value of the best macro-action from *that* posterior belief.

Therefore, to compute the expected reward for a macro-action $\tilde{a}$ from the root, we simulate its execution and generate all possible observation sequences from the root belief, resulting in a set of possible expected rewards and posterior beliefs after every primitive action along the macro-action. Posterior beliefs are sequentially updated for the given action and observation sequence according to Equation 2.3. For each posterior belief obtained after performing sequential updates for the entire length of each macro-action, we obtain a new set of macro-actions either by using the fixed set of macro-actions, or by dynamically generating a new set of macro-actions. We then expand out the forward search, as shown in Figure 3-1, and repeat this process until the tree depth reaches H/L, and a planning horizon H.

When the search reaches a depth of H primitive actions (H/L macro-actions), the value of the posterior belief nodes is set to zero. A common practice amongst existing forward search approaches is to use value heuristics to estimate the future discounted rewards from the leaf nodes, or to compute an offline approximation of the value function to be used at the leaves (Ross et al., 2008). By being able to search deeper, our macro-action planners instead enable us to achieve good performance on several longer horizon problems without using value heuristics to estimate the cost-to-go from the leaf belief nodes. To fully explore the po-

tential for macro-action planning, we therefore avoid the use of value heuristics at the leaf nodes for most of the algorithms presented in this thesis,[2] though it is trivial to incorporate additional value heuristics if desired. Instead, our planners only rely on the expected rewards computed along the search to estimate the value of a macro-action. Doing so enables us to avoid relying on domain knowledge to generate good value heuristics, nor be restricted to problem domains that are small enough for an offline solver to compute an approximation of the value function.

The expected values of the beliefs are then carried back up the tree, taking the maximum expected value over the macro-actions at each belief, and an expectation over observation sequences. Given the expected future reward of all the macro-actions, the macro-action with the highest expected future reward is chosen, and the first action $\hat{a}$ of this action sequence is executed. The agent then updates its current belief (Equation 2.3) based on the action taken and observation obtained, and the cycle repeats. Re-planning after each timestep is particularly advantageous in the context of macro-actions, as it allows the agent to condition its subsequent actions on the observations received in a manner that it was unable to during the previous macro-action planning step. The agent is therefore able to consider a wider range of policies, since it can now execute different subsequent actions based on the observations it has already received. This action-selection approach is also known as open-loop feedback control (Bertsekas, 2000), which forms the basis of many control methods, including Model Predictive Control. Planning and execution interleave until a termination condition is reached. Possible termination conditions include reaching a maximum number of execution steps in the environment, or receiving an observation that indicates that the agent should stop acting.

3.3.1 Computing the set of posterior beliefs

The MACROEXPAND routine involves taking the input macro-action $\tilde{a}$, computing the expected reward that will be received during the primitive action sequence which constitutes the macro-action $\tilde{a}$, and then computing the expected future reward after the macro-action completes. Computing the expected reward of a single macro-action $\tilde{a}$, which consists of a sequence of L primitive actions, requires taking an expectation of the posterior beliefs for possible observations that could be received during the macro-action execution. One trivial approach to compute the expected reward of a macro-action is to first enumerate all possible observation sequences of length L. Each observation sequence, plus the primitive action sequence of the macro-action, generates a posterior belief. Since both the expected reward of each posterior belief and the probability of each observation sequence can be computed, the expected reward for the macro-action starting at the given belief can then be computed from the expected rewards of all beliefs, weighted by the likelihood of the observation sequence that generated the belief, since there are still $|\mathcal{Z}|$ potential observations for each primitive action taken within a macro-action.The total number of potential observations for a sequence of $\tilde{H}$, L-length macro-actions is the same as a naïve forward search that branches on $|\mathcal{Z}|$ possible observations after every

[2]The only exception is the MMPBD target-tracking algorithm described in Chapter 7, which uses a naive heuristic to ensure that each macro-action is evaluated out to the same planning horizon.

Algorithm 3 MACROEXPAND() (Sampling observation sequences)

Input: Macro-action $\tilde{a}$, Belief state b_t, Discount factor γ, Macro-action search depth $\tilde{H}$, No. sampled observation trajectories N_z
if $\tilde{H} = 0$ **then**
  return 0
else {Expand Macro-Action $\tilde{a}$=$\{a^1, \ldots, a^L\}$}
  $V = 0$
  for $i = 1$ to N_z **do**
    $V_i = 0$
    for $j = 1$ to L **do**
      $V_i = V_i + \gamma^j R_B(b_t, a^j)$
      Perform transition update $b^{a,j} = \int_{s \in S} p(s'|s,a) b_{j-1}(s) ds$
      Sample observation z_j based on belief $b^{a,j}$
      Perform observation update $b_j(s') = \eta p(z_j|s',a) b^{a,j}(s')$
    end for
    Generate next set of macro-actions $\tilde{\mathcal{A}}^{next}$
    for $\tilde{a}_i^{next} \in \tilde{\mathcal{A}}^{next}$ **do**
      $Q(b_i, \tilde{a}^{i,next})$ = MACROEXPAND($\tilde{a}^{i,next}$,b_i,γ,$\tilde{H} - 1$,N_z)
    end for
    $V = V + \frac{1}{N_z}(V_i + \gamma^L \text{argmax}_{\tilde{a}^{i,next}} Q(b_i, \tilde{a}^{i,next}))$
  end for
  return V
end if

action: $|\mathcal{Z}^{\tilde{H}L}| = |\mathcal{Z}^H|$. Therefore, using macro-actions alone does not in itself break the exponential growth in computational cost as the horizon length increases, though it does reduce the computational complexity from $\mathcal{O}((|\mathcal{A}||\mathcal{Z}|)^H)$ to $\mathcal{O}(|\tilde{\mathcal{A}}|^{\tilde{H}}|\mathcal{Z}|^H)$.

Instead of enumerating all $\mathcal{O}(|\mathcal{Z}|^L)$ possible observations that could be received during a single macro-action, one could sample observation sequences for each L-length macro-action. Sampling observation sequences depends on the assumption that for most real-world problems, some observation sequences are much more likely than others, and hence a small set of observation sequences can have a disproportionate impact on the evaluation of the policy. If this assumption is valid, sampling a small number of observation sequences would be sufficient to accurately approximate the distribution of posterior beliefs that could result after executing a macro-action, as well as the value associated with the macro-action.

For each observation sequence, the planner alternates between a transition update based on the macro-action $\tilde{a}$, sampling an observation sequence based on the current belief, and updating the belief based on the sampled observation sequence and macro-action sequence. N_z observation trajectories are sampled for the same macro-action. At each posterior belief resulting from a particular observation sequence, the planner then evaluates the expected future reward of that posterior belief by recursively using the MACROEXPAND() subroutine to search out to a depth of $\tilde{H}$. Algorithm 3 depicts the MACROEXPAND() routine when posterior beliefs are obtained by sampling observation sequences.

This semi-conditional planning algorithm that samples observation sequences is valid both when the state space is discrete or continuous. When the problem domain has a discrete state space, we refer to the algorithm

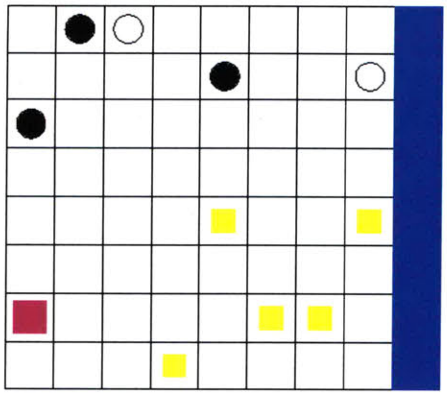


Figure 3-2: ISRS problem: Agent (purple square) has to retrieve rocks of high value (white circles). It receives noisy observations of each rock's value; observation accuracy increases with proximity to information beacons (yellow square).

as MACRO-ACTION DISCRETE(MAD) algorithm, whereas when the problem domain has a continuous state space, the algorithm is referred to as the MACRO-ACTION CONTINUOUS(MAC) algorithm.

3.4 Experimental results

3.4.1 RockSample

In this section, we present experimental results demonstrating the performance of the semi-conditional planner. We begin with a variant of the scientific exploration RockSample (Smith & Simmons, 2004) problem called the Information Search RockSample (ISRS) problem, shown in Figure 3-2. The scientific exploration RockSample problem is a benchmark POMDP problem first proposed by Smith and Simmons (2004), and subsequently extended to the FieldVisionRockSample (FVRS) problem by Ross and Chaib-draa (2007). Initial experiments in these domains revealed that a short look-ahead search was sufficient to obtain good policies. As our interest is in domains which require long-horizon look-ahead, we created the ISRS variant of the RockSample problem. In ISRS(n,k) an agent explores and samples k rocks in a $n \times n$ grid world. The positions of the agent (pink square) and the rocks (circles) are fully observable, but the value of each rock (good or bad) is unknown to the agent. At every timestep, the agent receives a binary observation of the value of each rock. The accuracy of this observation depends on the agent's proximity to rock information beacons (yellow squares) that each correspond to a particular rock. Unlike the original RockSample problem, the ISRS stresses the need to search far into the future, in that an agent must plan first to visit beacons to identify the value of rocks before sampling rocks. Information is physically separated from reward.

The agent gets a fixed positive reward for collecting a good rock (white circle), a negative reward for collecting a bad rock (black circle), and a smaller positive reward for exiting the problem. A discount factor

γ encourages the agent to collect rewards sooner. All other actions have zero rewards.

The observation model is a Bernoulli distribution:

$$p(z_{i,t}|s_i, r_t, RB_i) = \begin{cases} 0.5 + (s_i - 0.5)2^{\frac{-\|r_t - RB_i\|_2}{D_0}} & z_{i,t} = 1 \\ 0.5 - (s_i - 0.5)2^{\frac{-\|r_t - RB_i\|_2}{D_0}} & z_{i,t} = 0 \end{cases} \tag{3.1}$$

where $z_{i,t}$ is a binary $\{0 \text{ or } 1\}$ observation for the value of the rock i at time t,

s_i is the true value $\{0 \text{ or } 1\}$ of the rock,

r_t is the agent's position at time t,

RB_i is the location of the information beacon associated with rock i,

D_0 is a tuning parameter that controls how quickly the accuracy of the observations decrease with greater distance between the agent and the beacon.

Therefore, the FVRS and ISRS variants of the RockSample problem only differ in the definition of RB_i, the location where the agent gets perfect information of the hidden value of rock i. The transition and reward functions remain unchanged.

As the RockSample family of problems originates from the POMDP literature, we compared our macro-action algorithm to existing state-of-the-art POMDP solvers: a fast upper bound QMDP (Littman et al., 1995b), the point-based offline value-iteration techniques HSVI2 (Smith & Simmons, 2005) and SARSOP (Kurniawati et al., 2008), as well as RTBSS (Paquet et al., 2006), an online, factored, forward search algorithm. Since all approaches, including our own, are approximations, we also include as an upper bound the value of the fully observable problem, which would have been obtained had the hidden value of each rock been fully observable.

Given that the ISRS problem domain has a discrete state space, we compared the MAD semi-conditional planner against the benchmark POMDP algorithms. Similar to the RTBSS algorithm, we take advantage of the factored nature of the ISRS problem when performing forward search. Furthermore, we assume that domain knowledge is available, and we manually choose macro-actions that consist of the sequence of actions that are expected to get the agent to a rock, an information beacon, or to the nearest exit. Since the world is both stochastic and partially observable, no open loop action sequence is actually guaranteed to get the agent to a rock or beacon. The exit as an absorbing state may have been a special case. If the agent is currently on a rock, the action sequences have the additional option of sampling the rock as the first action in the action sequence, resulting in twice as many macro-actions. Figure 3-3 shows two of the macro-actions that are generated given the agent's current position.

Table 3.1 compares the performance of the different algorithms in the ISRS problem. Each algorithm was tested on 10 sets of hidden rock values, and each scenario was tested 20 times. The HSVI2 and SARSOP algorithms were executed offline for a range of durations,[3] while the forward search algorithms were allowed

[3]The offline execution durations for both HSVI2 and SARSOP were chosen empirically. HSVI2 was able to execute the ISRS[8,5] problem for 1,000s offline before running out of memory. It was found that the regret rates computed by SARSOP remained constant

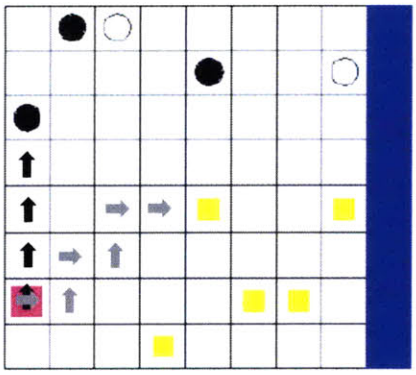


Figure 3-3: Macro-actions for the ISRS problem consist of the sequence of primitive actions that are expected to get the agent to a rock (black arrows), an information beacon (grey arrows), or to the nearest exit.

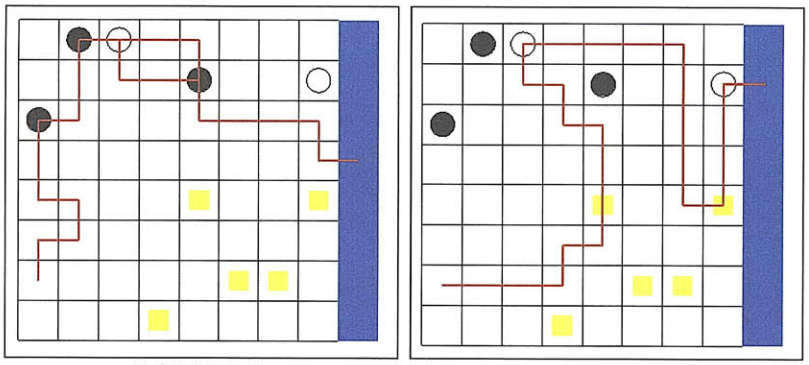
	ISRS[8,5]		
	Avg rewards	Online time (s)	Offline time (s)
QMDP	1.11 ± 0.43	0.0001	3.03
HSVI2	6.76 ± 2.55	0.017	150
HSVI2	6.78 ± 2.46	0.051	1000
SARSOP	4.47 ± 2.74	0.007	150
SARSOP	8.25 ± 2.35	0.045	10000
SARSOP	8.46 ± 2.46	0.070	25000
RTBSS (d5, s10)	9.78 ± 1.69	17.64	0
MAD (d3,s50)	**15.88** ± 1.58	4.81	0
Fully observable	21.37	N.A.	N.A.

Table 3.1: ISRS results. HSVI2 and SARSOP were executed offline for a range of durations. For the forward search algorithms, the numbers in brackets represent the search depth (d) and number of posterior beliefs obtained (s) at the end of each action/macro-action. Online time indicates the average time taken by the planner to return a decision at every timestep. Standard error values are shown.

to search out to a pre-defined depth. Here, depth refers to the primitive action depth in the RTBSS algorithm, and the macro-action depth for the MAD algorithm. In addition, a pre-defined number of samples were used to obtain posterior beliefs after every action/macro-action. We abuse the language here slightly by using samples to refer to observations in the RTBSS algorithm and observation sequences in the MAD algorithm.

Our macro-action algorithm does significantly better than the other benchmark solvers, demonstrating that approaches that can search deeper can yield better policies on this domain. Figure 3-4a and 3-4b compare the agents resultant paths based on the policies generated by the SARSOP algorithm and the MAD algorithm respectively in the ISRS problem. Macro-actions allow our forward search planner to uncover the potential value of moving to an information beacon, weighing the relative benefits of traveling to either the beacons or the rocks. Specifically, the MAD planner reasons that in expectation, it is more valuable to first travel to

after 25,000s.



(a) SARSOP policy (b) MAD policy

Figure 3-4: ISRS problem. White circles correspond to rocks with positive value, black otherwise. Yellow squares indicate locations of the rock information beacons. Red lines indicate paths taken by an agent executing the (a) SARSOP and (b) MAD policies.

the beacons if it has large uncertainty of the rock values, before conditioning its subsequent actions on the observations that it obtains. This allows our macro-action forward search approaches to perform much better than prior primitive-action approaches. Figure 3-4b shows that the MAD agent's policy involves visiting some of the information beacons to gather information about which of the rocks are good (white circles), before traveling to those rocks to sample them. Although the detour to the beacons requires a longer amount of time, the ability to obtain observations that are less noisy makes the detour a valuable tradeoff when the agent is highly uncertain.

In contrast, both SARSOP and HSVI2 explore parts of the belief space guided by an upper bound on belief-action values. As it takes a long look-ahead to realize that visiting beacons and then rocks has a higher value that visiting rocks, it will take many iterations and therefore substantial computation time for SARSOP and HSVI2 to sample the beliefs that will lead to them computing a higher-value policy. SARSOP seldom explores parts of the belief space that have lower value in the short planning horizon, but larger rewards over a much longer planning horizon. Similarly, HSVI2 uses a heuristic search to choose the belief points for point-based value iteration, and focuses the search on beliefs that have a high upper value bound. However, some beliefs can only be found to have a high value by performing a very long-horizon search, and so HSVI2 will unlikely explore these beliefs. In the considerable offline computation time provided, both SARSOP and HSVI2 did not discover that it is valuable for the agent to make a detour to the information beacons before approaching the rocks. Instead, as shown in Figure 3-4a, they directly approach the rocks and make decisions based on the noisy observations that are obtained due to the large distance from the information beacons.

3.5 Applicability of semi-conditional planning

The semi-conditional planning approach is a new technique for planning under uncertainty in partially observable domains. Specifically, for problems that require a far-lookahead, our MAD planner obtains better performance relative to state-of-the-art POMDP solvers, trading the computational savings from avoiding fully-conditional planning with the ability to search out to a longer horizon. Nevertheless, semi-conditional planning does not provide a magic bullet for solving all partially observable planning under uncertainty problems. In fact, it may even appear counter-intuitive that the idea of performing semi-conditional planning — using open-loop macro-actions and only conditioning at certain key points, would even be useful for any partially observable domain. After all, an agent is typically required to update its belief at every timestep based on the latest observation obtained, so that it has the latest available information to make the most informed action choice possible.

Nevertheless, as the experimental results in Section 3.4 demonstrate, there are partially observable problem domains where semi-conditional planning produces good performance. We would therefore like to understand the conditions under which a semi-conditional planner can generate good policies without having to condition on the observations after every action.

Intuitively, one condition where semi-conditional planning would be useful is when there is little process noise in the problem domain, or little stochasticity in the state-transition model. The agent's actions is then approximately deterministic, and assuming that the agent is relatively certain of its initial state, the agent will be able to operate in an open-loop fashion most of the time, and only needs to incorporate the observations obtained once every few timesteps. Therefore, although a semi-conditional planner conditions on the observation sequences only at the end of each macro-action, when there is little stochasticity in the state-transition model, this amount of conditioning should be sufficient for the agent to choose good actions to execute. An example of a partially observable problem with little stochasticity in the state transition model is one where the state space can be fully factorized into fully observable and partially observable components, and where the stochasticity is fully contained within the fully observable state variables. The class of Rocksample problems from the POMDP community possesses this property, since the partially observable state variables in these problem domains (rock values) are stationary. Similarly, a robot with low noise in its motors and actuators that attempts to localize itself globally would only need to incorporate the observations received when it believes that it has reached key locations in the environment.

Second, in certain problem domains, the optimal policy may entail that for certain parts of the belief space, the agent should execute a fixed action at the next timestep, regardless of the observations that are obtained at the current timestep. Such a policy implies that for those regions of the belief space, conditioning the agent's subsequent actions on the observations received is redundant, and the planner can instead compute an equally good policy by planning in a semi-conditional fashion and evaluating open-loop macro-actions. For example, back to the Rocksample problem, if the agent is highly certain that a particular rock has high value, the optimal policy would involve taking the action sequence that will get the agent to the rock, regardless of

the observations that are obtained by the agent enroute.

We would like to be able to determine which problem domains would be admissible to semi-conditional planning just by analyzing the transition, observation and reward models. The first condition of having low process noise can be trivially determined by analyzing the problem domain's transition model. Unfortunately, the second condition appears difficult to verify without computing the optimal policy for the problem domain. In addition, the performance of our semi-conditional planners depends heavily on the quality of the macro-actions that are being evaluated.

Nevertheless, we can develop heuristics to help predict if a semi-conditional planner, using a good set of macro-actions, can ignore conditioning along a macro-action without affecting the quality of the computed policy. Specifically, we analyze the problem domain's reward model, and hypothesize that if for every action, there is low variance in immediate rewards for taking that action over the state space, then there are likely to be regions where the agent should execute the same action regardless of the observations that are obtained. When there is low-variance in the rewards for taking a particular action, and the same condition holds for most actions in the action space, then the same action will maximize the agent's expected immediate rewards from any posterior belief, regardless of the observations obtained. The agent will only need to condition its subsequent actions on the observations obtained when the agent's belief has support over those states that will return drastically different rewards relative to an action's average reward.

Problem domains with low-variance reward models typically have a small set of sub-goals that need to be accomplished, and examples of such problem domains include goal-oriented motion-planning tasks. In between trying to reach these sub-goals, the agent typically receives little or no rewards during the intermediate timesteps, and therefore the planner can accurately compute the expected value of taking different actions from the agent's current belief without conditioning at every step.

In contrast, problem domains that require careful consideration of the risk of taking different actions at every step will find that planning with macro-actions is unlikely to result in a good policy, since the expected rewards computed if the agent executes the same action regardless of the observations obtained (semi-conditional planning) would typically be drastically different from the expected value of a policy that considers executing different subsequent actions when different observations are obtained (fully-conditional planning). Many partially observable stochastic games (Hansen et al., 2004) would therefore not find macro-actions useful, since these games require each agent to immediately react to the observations it receives about the state of the other agents in the environment, and typically the reaction to each observation is different.

More formally, we define a performance metric $R_{\bar{\sigma}}$, which describes the average reward variance across the different possible actions in the action space $\mathcal{A}$. For each action a^i, we first define the mean reward

$R_\mu(a^i)$ and reward variance $R_\sigma(a^i)$ as follows

$$R_\mu(a^i) = \frac{1}{|\mathcal{S}|}\sum_{j=1}^{|\mathcal{S}|} R_S(s_j, a^i) \tag{3.2}$$

$$R_\sigma(a^i) = \frac{1}{|\mathcal{S}|}\sum_{j=1}^{|\mathcal{S}|}\Big(R_S(s_j, a^i) - R_\mu(a^i)\Big)^2 \tag{3.3}$$

The average reward variance across all the actions is therefore

$$R_{\overline{\sigma}} = \frac{1}{|\mathcal{A}|}\sum_{i=1}^{|\mathcal{A}|} R_\sigma(a^i) \tag{3.4}$$

Therefore, the larger the reward variance, the less likely that semi-conditional planning will obtain good performance when applied to those problem domains. Nevertheless, the reward variance alone ignores the observation model and process noise, both of which affect the planner's ability to plan accurately for many steps in an open-loop fashion. In addition, for many planning under uncertainty problems, it may be more valuable to determine when is it necessary for the planner to condition the agent's actions on the observations obtained, rather than to find problem domains where conditional planning is generally unnecessary. The applicability criteria in this case may then focus more on the *distribution* of reward variance over the state space, rather than the mean reward variance. We seek to explore other criteria in future research.

3.5.1 Experimental results: ISRS with potholes

Having provided a criterion for determining which problem domains are amenable to semi-conditional planning, we now provide some experimental results demonstrating how the reward model can influence the applicability of semi-conditional planning.

We first modify the Information RockSample problem by making the agent's navigational actions slightly noisy, and the agent's knowledge of its own position partially observable. Whenever the agent executes a move action, it moves to its intended location with 0.9 probability, and with the remaining probability moves to an adjacent location. Beyond the action uncertainty, the agent also suffers from state uncertainty of its own location. The agent only obtains perfect information of its location when it is over a rock, but gets no information of its position when it is in any other grid cell. Hence, the longer the agent moves around the environment without passing a rock, the more uncertain it will be of its own location.

In addition, we introduce the notion of "potholes" into the problem. As the agent moves around the environment, there are certain squares for which the agent will incur a negative reward just by virtue of landing in that square; in contrast with the rocks, where the agent obtains positive or negative rewards only when it executes the sample action when it is over the rock. Because the agent's location is partially observable, it is possible for the agent to land in one of these potholes unknowingly.

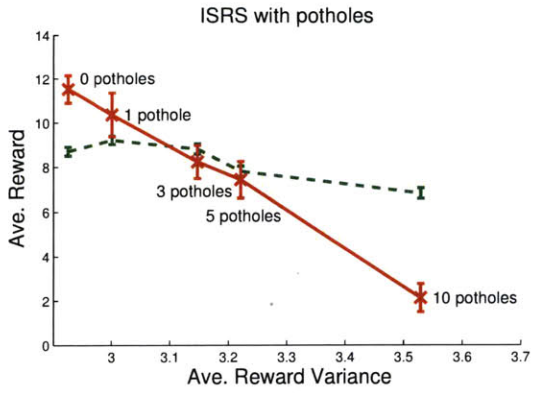


Figure 3-5: Performance of semi-conditional planner (red line) relative to SARSOP (green dotted line) on the ISRS problem with potholes, when the reward variance increases.

To test the influence of the reward model, we varied the number of potholes in the problem, changing the variance of the reward model. The size of the problem domain, as well as the rock and beacon locations were kept constant, while the locations of the potholes were randomly sampled for each problem. For this test, macro-actions were generated using the method discussed in Chapter 4.

We compared the performance of our semi-conditional planner to the SARSOP algorithm for a variety of problem domains; SARSOP is widely considered as a state-of-the-art point-based POMDP algorithm. Figure 3-5 shows the performance of both algorithms as the number of potholes varies. When there are no potholes in the problem, our semi-conditional planner performs significantly better than SARSOP. This result tallies with the results presented earlier, even with the addition of partial observability of the agent's location. However, as the number of potholes in the problem increases, not only does the performance of the semi-conditional planner worsen, but more importantly, the semi-conditional planner experiences a much steeper performance decline relative to SARSOP. When there are 10 potholes in the problem domain, the semi-conditional planner performs significantly worse than SARSOP. This result matches the hypothesis that when there exist a large variance in rewards, it is necessary for a planner to condition subsequent actions on the observations received more frequently.

3.5.2 Performance analysis

Planning out to a long horizon enables a planner to incorporate rewards that may only be obtained many timesteps in the future. A forward search planner can then compute the expected value of taking different actions more accurately, resulting in policies that produce better performance.

For completeness, we first re-derive the known result (Puterman, 1994) that the error between the optimal infinite-horizon expected belief-action value and the optimal H-horizon expected belief-action value is bounded as a function of the horizon H:

Lemma 1 *Let the possible rewards be bounded by 0 and R_{max}. Then the error between the infinite-horizon optimal expected belief-action value $Q^*(b,a)$ and the optimal H-length horizon expected belief-action value $Q_H(b,a)$ is a bounded function of H:*

$$Q^*(b,a) - Q_H(b,a) \leq \frac{R_{max}\gamma^{H+1}}{1-\gamma}. \tag{3.5}$$

Proof Let R_i be the expected reward on the i-th timestep. Then

$$\begin{aligned} Q^*(b,a) - Q_H(b,a) &= \sum_{i=0}^{\infty}\gamma^i R_i - \sum_{i=0}^{H}\gamma^i R_i && (3.6)\\ &\leq \sum_{i=H+1}^{\infty}\gamma^i R_{max} && (3.7)\\ &= \sum_{i=0}^{\infty}\gamma^i R_{max} - \sum_{i=0}^{H}\gamma^i R_{max} && (3.8)\\ &= \frac{R_{max}}{1-\gamma} - \frac{R_{max}(1-\gamma^{H+1})}{1-\gamma} && (3.9)\\ &= \frac{R_{max}\gamma^{H+1}}{1-\gamma}. && (3.10) \end{aligned}$$

where the first inequality follows since the additional rewards are at most R_{max} for the subsequent timesteps, the second equality just re-expresses the sum as a difference of two sums, the third equality takes the geometric sum, and the final equality is our desired result. □

Assuming that the reward function is strictly positive, searching to a longer horizon therefore allows the planner to compute an expected value that is closer to the infinite horizon optimal expected value. However, semi-conditional planners search out to a longer horizon at the expense of conditioning the agent's actions on the observations obtained at every timestep, and hence it is impossible to guarantee that the computed expected value is arbitrarily close to the infinite horizon optimal expected value. Nevertheless, if the problem domain is amenable to semi-conditional planning based on the criteria discussed in Section 3.5, and if a sufficiently large number of macro-actions are evaluated, it is likely that a semi-conditional planner will obtain similar performance as a fully-conditional planner planning up to the same horizon.

3.5.3 Computational complexity

We also consider the computational complexity of performing semi-conditional planning with macro-actions in a forward search manner. By using a set of $\tilde{\mathcal{A}}$ macro-actions, each of length L, we can reduce the branching factor from $|\mathcal{A}|$ at each depth to only $|\tilde{\mathcal{A}}|$ every L steps, after a macro-action completes. This process reduces the total computational cost due to action branching from $|\mathcal{A}|^H$ to $|\tilde{\mathcal{A}}|^{H/L}$. If we consider that N_z unique observation trajectories are sampled per macro-action, then the number of forward search tree nodes expanded is $\mathcal{O}(|\tilde{\mathcal{A}}|^{H/L}N_z^{H/L})$. Belief updating and calculating the expected reward are order $\mathcal{O}(|\mathcal{S}|^2)$ and $\mathcal{O}(|\mathcal{S}|)$

operations respectively, and must be performed at each node in the tree. Therefore the total computational cost to compute this tree is

$$\mathcal{O}(|\tilde{\mathcal{A}}|^{H/L} N_z^{H/L} (|\mathcal{S}|^2 + |\mathcal{S}|)), \tag{3.11}$$

which is a substantial reduction in computational cost relative to a fully-conditional forward search tree.

Now consider the complexity when using a factored-discrete state representation. Assume that each of the D state dimensions is independent of the other dimensions, and that each state dimension is divided into g uniformly-spaced grid cells. The computational cost includes performing belief updates and evaluating the reward along each of the N_z sampled length-L observation sequences. The cost of performing a belief update is a quadratic function of the state space, but each dimension can be updated independently, leading to a cost of $\mathcal{O}(g^2 D)$. Assuming the reward model is also factored, computing the expected reward consists of D dot products between g-length vectors. The computational complexity of planning in a factored-discrete state representation is therefore

$$\mathcal{O}(|\tilde{\mathcal{A}}|^{H/L} N_z^{H/L} L(g^2 D + gD)). \tag{3.12}$$

Note that in high-dimensional problems, this cost is significantly less than the cost of using a fully dependent model, which would result in $|\mathcal{S}| = g^D$ states, and an associated

$$\mathcal{O}(|\tilde{\mathcal{A}}|^{H/L} N_z^{H/L} L(g^{2D} + g^D)) \tag{3.13}$$

computational complexity for planning. In continuous domains, the quality of a discrete approximation will depend on the grid cell granularity: by making the tiling sufficiently fine, it is possible to make the discretized dynamics, reward and observation models accurate to any specified precision. However, finer grids have a direct impact on the computational cost of planning, particularly in fully dependent (non-factored) representations.

3.6 Macro-actions planners for partially observable environments

The planning under uncertainty literature contains a small number of techniques that use macro-actions for sequential decision-making in uncertain and stochastic domains. While similar to the hierarchical POMDP approaches discussed in Chapter 2, these techniques do not seek to enforce a hierarchical decomposition of the problem domain.

Theocharous and Kaelbling's (2003) discrete-state reinforcement learning approach samples observation trajectories and solves for the expected reward of a discrete set of belief points using function approximation. It takes as input a POMDP model and a set of macro-actions, and by performing successive trials of the

environment using the macro-actions, outputs a set of grid points in belief space and associated action-values. By interpolating the computed values at these belief grid points, the algorithm specifies a value function over the entire belief space that can be used to compute a policy for the agent. Such an approach therefore continues to perform fully-conditional conditioning when computing the expected value from any belief point. In addition, while their experimental results demonstrate the potential advantages of macro-actions, no bounds are provided on the resulting solution quality.

Yu et al. (2005) provide an optimal algorithm for planning if no observations were available, but do not provide performance guarantees when noisy sensor information is received. Foka and Trahanias's (2007) solution involves building a hierarchy of nested representations and solutions. Their focus is on discrete-state problems, particularly navigation applications, and no performance guarantees are provided. Hsiao et al. (2008) used a form of macro-actions for robot manipulation tasks, but the focus of their work is on robust manipulation under uncertainty, and their work only considers a very short horizon of action trajectories.

3.6.1 Using macro-actions in point-based POMDP solvers

To our knowledge, macro-actions have not been used in an online, forward search manner for decision-making under uncertainty. Nevertheless, a natural question is whether macro-actions can be used within an offline, point-based solver, such as those presented in Section 2.2.2, rather than plan in an online, forward search manner.

For example, Kurniawati et al. (2009)'s recent work uses macro-actions to guide the sampling of belief points for use in a point-based POMDP solver. This approach is applied to partially observable motion-planning tasks. The authors prove that if the sampled beliefs cover the reachable belief space sufficiently densely, then the resulting optimal value at the sampled beliefs will be arbitrarily close to the value of nearby beliefs that are not sampled. Macro-actions are generated by sampling states from the state space and computing a set of sequential actions that will get the agent from one state sample to another. Observation sequences are similarly sampled, resulting in a set of reachable beliefs. A policy is computed by performing value function backups over this belief set.

Though experimental results demonstrate significant improvements over prior POMDP solvers, the computational complexity of performing the value backup is still significant, since MiGS is an offline approach that explicitly computes a representation of the value function from α-vectors for the sampled beliefs, and value backups only use the primitive actions and observations. The MiGS algorithm is therefore still planning in a fully-conditional manner when performing value function backups, and the computational complexity of the value function operator scales as the product of the primitive action space, the observation space, the size of the state space squared, and $|\Gamma|$, the number of prior α-vectors, i.e. $\mathcal{O}(|\mathcal{A}||\mathcal{Z}||\mathcal{S}|^2|\Gamma|)$. In contrast, by performing semi-conditional planning in a forward search manner, our approach will scale better in large spaces. Furthermore, while our forward search algorithms can exploit factored models to quickly compute belief updates and expected reward calculations for large state problems, MiGS seems likely to struggle with

large problems with a high number of state dimensions. Kearns and Koller (1999) showed that the value function in general cannot be factored, implying that MiGS cannot directly leverage factored representations. Finally, the MiGS algorithm is also mostly focused on motion planning tasks rather than generic planning under uncertainty problem domains.

Taking the question of applying macro-actions in point-based solvers further, it is worth exploring whether macro-actions could be directly used within some of the current state-of-the art point-based solvers, such as HSVI2 (Smith & Simmons, 2005) or SARSOP (Kurniawati et al., 2008). HSVI2 and SARSOP both iteratively expand trajectories of belief states until a desired bound on the value function at the initial belief is achieved. HSVI2 and SARSOP also both maintain upper and lower bounds on the value function, and use these bounds to choose action-observation pairs that construct a trajectory of beliefs.

Following the approaches taken by HSVI2 and SARSOP would require us to be able to compute the upper and lower bounds on the value of a macro-action, in order for the planner to decide which macro-action to apply from any of the existing belief state samples. The lower bound can be computed using the standard Bellman backup out to the length of the macro-action, though as mentioned above, that can be expensive. On the other hand, an approximate upper bound on the macro-action value could be computed by using a variation of the QMDP (Littman et al., 1995a) heuristic, which itself has been popular for computing an upper bound in existing point-based POMDP solvers. Specifically, we could obtain possible posterior beliefs after the macro-action is executed by sampling observation sequences of corresponding length. Applying the QMDP approximation at each of the possible posterior beliefs and propagating the values up the macro-action, we can then choose the highest value amongst the sampled posterior beliefs to obtain an approximation of the upper bound of the macro-action. However, since we do not exhaustively evaluate all possible posterior beliefs that could result from the macro-action, our computed value remains only an approximation of the macro-action upper bound. Using macro-actions would then remove a key strength of these approaches: the ability to selectively sample actions that are based on the exact upper and lower bounds. Developing value function bounds that operate directly on distributions of beliefs, or representing value functions over distributions of beliefs, might allow the upper bound to be computed exactly, though that is beyond the scope of this thesis.

Point-based POMDP algorithms also rely on an explicit representation of the value function over beliefs by using a set of α-vectors that are created by backing up individual beliefs along the trajectory. As discussed in Section 2.2.2, maintaining and updating this value function representation in large problems is expensive. Therefore, while macro-actions could be used to sample belief points in a point-based POMDP solver like SARSOP or HSVI2, doing so would remove the ability for point-based POMDP solvers to selectively sample actions in a theoretically precise manner, but would still require the full expense of value function backups. On the other hand, it may be possible to use the macro-actions within the value function backup operator, by performing value function backups not just using the primitive actions and observations, but including macro-actions and macro-observations of corresponding length. Such an approach would introduce semi-conditional planning into offline POMDP techniques, avoiding the need to condition on every primitive action during the

value function backup. However, evaluating a macro-action extension to the value-function backup operator is again beyond the scope of this thesis, and we leave this potential research for future work.

3.7 Challenges of semi-conditional planning

The macro-action planner presented in this section provides a general framework for being able to plan under uncertainty in a forward search manner out to a long horizon. However, a number of issues have not yet been addressed, and in some problem domains there may exist additional problem structure that can be exploited for greater computational efficiencies.

First, semi-conditional planning will only result in good performance if good macro-actions are constructed. While macro-actions restrict the plan space of the search process such that the planner can search much farther in the future, poor performance may result if the plan space is overly restrictive. It is therefore crucial to generate macro-actions that will lead to good performance, i.e., the macro-actions that are anticipated to be part of a good plan. In addition, the planner will only choose good actions for the agent if good choice points are chosen along the macro-action forward search tree to incorporate the observations, so that the expected value of the different root actions are more accurately computed. Generating good macro-actions requires not just the ability to know which actions to chain together into an action-sequence, but also the ability to define the optimal length of each macro-action. Thus far, we have assumed that a domain expert is available to hand-code the set of macro-actions that restricts the policy space while retaining the good macro-actions. For many problems, however, it may not be easy to find a domain expert that can encode a good set of macro-actions, or it may not even be clear *a priori* when to condition for good performance.

We have also discussed in this chapter that not all problem domains are amenable to semi-conditional planning, since some problem domains require conditioning on the observations that the agent receives at every timestep. Nevertheless, we would like to be able to ensure that even in problems that require fully-conditional planning, our algorithms will still be able to perform well.

The second technical challenge concerns the set of possible posterior beliefs that can be obtained after a macro-action is executed. The expected immediate reward received during a macro-action, and the expected future reward received after a macro-action, are both functions of the set of posterior beliefs that could result after a macro-action is executed. A question that naturally follows is how to obtain the set of posterior beliefs that could result after a single macro-action. Exhaustive enumeration of the possible posterior beliefs, which would require exhaustive enumeration of the possible observation sequences that resulted in those beliefs, will be intractable when there are a large number of observations, or when the macro-action consists of a long sequence of primitive actions. Sampling the observation trajectories yields an approximation of the true posterior distribution of beliefs, but can still be computationally intensive due to the number of calculations that must occur at each step of each sampled observation sequence.

In the subsequent chapters, we present new algorithms that address these technical challenges for semi-

conditional planning.

Chapter 4

PUMA: Automatic macro-action generation and iterative refinement

(Joint work with Emma Brunskill)

This chapter builds on the previous chapter and addresses the challenge of constructing good macro-actions in order to obtain good semi-conditional planning performance. As we discussed in Chapter 3, using macro-actions implicitly restricts the plan space of the search process, possibly resulting in reduced performance. It is therefore crucial to have macro-actions that are anticipated to be part of a good plan. While a domain expert could hand-code a good set of macro-actions for some problems, this may be challenging or impractical for many domains. In this chapter, we present a technique for automatically constructing finite-length open-loop macro-actions. Our approach uses sub-goal states based on immediate reward and potential information gain to generate candidate macro-actions that are evaluated in a forward search manner. In addition, because it may be beneficial for the planner to condition on the associated observation sequences even before the sub-goal is reached, we provide an anytime algorithm that incrementally refines the initial macro-action plan as more computational time is available for planning. The refinement process progressively increases the amount of conditional planning in the tree, resulting in good performance even in domains that may require close to fully-conditional plans. As more planning time is made available, this anytime algorithm guarantees eventual convergence to an ϵ-optimal policy.

We present the semi-conditional planner PUMA, which stands for *Planning under Uncertainty with Macro-Actions*. We adopt the POMDP framework from the artificial intelligence community, rather than the more general planning under uncertainty framework that we presented in Chapter 2. PUMA outperforms a state-of-the-art POMDP planner both in terms of plan quality and computational cost on two large simulated POMDP problems. While PUMA takes longer to plan than using hand-constructed macro-actions, it provides comparable performance without reliance on domain knowledge either for macro-action construction or for heuristic value functions at the leaves of the forward search. We additionally show that anytime improvement

Algorithm 4 GENERATEMACROS()

Require: Belief b_N, Num macros M, Max length L

for $k = 1$ to M **do**
  Sample state $s_{s,k}$ from b_N
  $V_g \leftarrow 0$
  while $V_g(s_{s,k}) == 0$ **do**
    Sample sub-goal state $s_{g,k}$
    Compute SSP policy π_g for sub-goal $s_{g,k}$
  end while
  $j \leftarrow 1$, Set state $s_c \leftarrow s_{s,k}$
  while $j < L$ and state $s_c \neq$ goal state s_g **do**
    $\tilde{a}_k(j) = \pi_g(s_c)$ (select action using SSP policy π_g)
    $s_c = \text{argmax}_{s'} P(s'|s_c, \tilde{a}_k(j))$
    $j \leftarrow j + 1$
  end while
  Add macro-action $\tilde{a}_k$ to $\tilde{\mathcal{A}}$
end for
Return macro-action set $\tilde{\mathcal{A}}$

allows PUMA to achieve good performance on a domain that requires fully conditional planning, suggesting PUMA has significant potential as a generic planner for large POMDP problems.

4.1 Generating macro-actions

Performing semi-conditional planning effectively requires a planner to generate a set of macro-actions $\tilde{\mathcal{A}}$ of maximum length L and evaluate them using forward search. However, the best macro-action that can be executed from any two beliefs often varies greatly, especially since the beliefs could have support over very different regions of the state space. Rather than have the planner evaluate a large, fixed set of macro-actions for every belief, our planner instead re-generates a new set of macro-actions at every step.

To generate macro-actions automatically, a naïve approach would be to chain together randomly sampled primitive actions. However, the space of possible macro-actions of maximum length L grows exponentially larger with L, making it unlikely that a randomly-generated macro-action would be the best possible open-loop action sequence that the agent could have executed. In contrast, many macro-action planners in fully-observable domains create mini-policies designed to achieve sub-goals (for example, see McGovern (1998), and Stolle and Precup (2002)). We hypothesize that sub-goals are similarly applicable within a POMDP context.

4.1.1 Reward exploitation and information gathering weights

To select suitable sub-goals, we take inspiration from Cassandra et al.'s (1996) and Hsiao et al.'s (2008) work showing that good POMDP performance can be achieved by choosing either actions with high rewards or actions with a large gain in information under the current belief. Our macro-actions are therefore finite-length open-loop plans designed to take the agent from a possible state under the current belief towards a

state that will provide either a large immediate reward or a large gain in information. For example, recall that for the ISRS problem presented in Chapter 3, our macro-actions were manually chosen to consist of the sequence of actions that will get the agent to a rock, an information beacon, or to the nearest exit. Similarly, when generating macro-actions automatically, sub-goals for the ISRS problem could include having the agent arrive at one of these landmarks, regardless of the underlying rock values.[1]

More formally, we compute two set of weights, each corresponding to either the reward exploitation or information gain metric. Each set of weights determines the probability that the planner will choose a particular state as a sub-goal for generating the macro-action when that particular sub-goal metric is chosen. For each state s_i, we compute the reward exploitation heuristic (RE) according to:

$$RE(s_i) = \max_{a \in |\mathcal{A}|} \frac{R_S(s_i, a) - R_{min}}{R_{max} - R_{min}} \tag{4.1}$$

The reward exploitation heuristic corresponds to the rewards that the agent can expect to receive, assuming it executes the optimal action after reaching the sub-goal state. These rewards are first normalized by the minimum and maximum instantaneous rewards, so as to ensure that each of the values are strictly positive. Finally, the set of reward exploitation weights are computed by normalizing the reward exploitation values such that the set of weights sum to one.

We next determine which are the "informative" states in the problem domain — states that will enable the agent to ascertain the state of the world with high certainty through the observations that the agent receives. Seen from another angle, our information gain heuristic should be biased towards states that have a high probability of emitting observations that would lead to a sharp reduction in the entropy of the agent's belief — entropy in information theory is a measure of the uncertainty associated with a random variable. We therefore seek to determine which observations would result in the largest reduction in entropy of the agent's belief after the observation is incorporated. before determining which states have a high probability of emitting these observations.

Although the relative importance of each observation for entropy reduction is typically belief-dependent, it is computationally expensive to compute a new set of information gain weights for every belief. Instead, we compute a heuristic value $IG_Z(z)$ for each observation by assuming that the prior belief is a uniform distribution over the state space, according to

$$IG_Z(z) = \mathcal{H}(b_0) - \mathcal{H}(b_z) \tag{4.2}$$

where $\mathcal{H}(b) = -\sum_s b(s) \log b(s)$ is the entropy of the agent's belief, and $b_0(s) = 1/|\mathcal{S}|$ represents the prior uniform belief and $b_z(s) = p(s|z)$ represents the posterior belief after incorporating the observation.

Having computed the relative importance of each observation in the observation space $\mathcal{Z}$, the expected

[1] Note that in the ISRS problem, a state instance includes both the location of the agent in the grid world, as well as a value for each of the rocks.

information gain of each state, $IG(s)$, can be calculated based on the probability of the agent obtaining each observation from that state.[2]

$$IG(s) = \sum_{z \in \mathcal{Z}} p(z|s) IG_{\mathcal{Z}}(z) \tag{4.3}$$

The values are then normalized to obtain a set of information gain weights that sum to one. It is worth noting that since both the reward exploitation and information gain heuristics were computed in a belief-independent fashion, both sets of weights only need to be computed once for each problem domain.

4.1.2 Stochastic shortest path

Having computed both sets of weights, we can now sample a sub-goal state $s_{g,k}$ from a distribution over the state space, weighted by the normalized reward exploitation/information gain values. Separate sample distributions are maintained for the reward and information heuristics, and at each instance we use one of the two distributions with equal probability to sample a sub-goal.

For each sub-goal state, we formulate a stochastic shortest path (SSP) problem (Bertsekas & Tsitsiklis, 1991), which is a specific type of MDP that has positive costs and an absorbing goal state. A high reward is associated to the sub-goal state, zero rewards to all other states, and the transition model is made identical to the POMDP transition model. A discount factor $0 < \gamma < 1$ biases the policy towards action sequences that will arrive at the goal state in the smallest number of timesteps. This policy is computed using a standard value iteration procedure, though more efficient techniques may exist within the stochastic shortest path literature (Bonet & Geffner, 2002). The resultant MDP policy therefore provides a mapping from states to actions, indicating the action that should be taken from every state that, given the problem domain's reward and transition model, will maximize a value function. It is possible that even with non-deterministic actions in the problem domain, the goal state is not reachable from some of the states. These states can be identified as those states with an expected value of zero under the converged SSP policy.

Finally, bringing all the pieces together, whenever we seek to generate a macro-action from the current belief state b_t, we first sample a start state $s_{s,k}$ from b_t and a sub-goal state $s_{g,k}$. Given these two states, we then search for a sequence of actions that will move the agent from $s_{s,k}$ towards $s_{g,k}$ under the fully observable model. Assuming that the sub-goal may eventually be reached from the current belief, we generate the macro-action by simulating execution of the MDP, assuming that the maximum likelihood state is reached at every step. Actions are added until either the sub-goal $s_{g,k}$ is reached or the macro-action is of maximum length L, and the actions taken under the sub-goal policies constitute the macro-action. We repeat the

[2] It can be shown that when all observations have a non-zero probability of occurrence at every state, and when the observations have identical noise models over the state space, then our information gain heuristic for each state is equivalent to the entropy of the observation emission probabilities for that state. The observation emission probability $p(z_k|s_i)$ indicates the probability that observation z_k is obtained by the agent if the state of the world is s_i, and the entropy of the observation emission probabilities can then be computed according to $IG(s) = -\sum_{z \in \mathcal{Z}} p(z|s) \log p(z|s)$. For computational efficiency reasons, we adopt this alternative representation when computing our information gain weights for the experimental problem domains in this chapter.

macro-action generation process M times for a particular belief to generate M macro-actions. Algorithm 4 summarizes our macro-action generation process.

4.2 Forward search using automatically generated macro-actions

Having proposed a technique for generating macro-actions automatically and without domain knowledge beyond the model itself, we can directly apply this technique to the general macro-action framework (Algorithm 2) that was proposed in Chapter 3. The algorithm alternates between a planning and execution phase at every iteration. During the planning phase, the algorithm automatically generates a set of M macro-actions with maximum length L from the current belief b_c, using the GENMACROS process (Algorithm 4) described in Section 4.1. We set M to always be at least as large as the number of primitive actions $|\mathcal{A}|$, so that the planner will possess convergence properties that will be discussed in Section 4.4.

To compute the expected reward for a macro-action $\tilde{a}$ from the root belief node b_c, the planner simulates the execution of each macro-action and generates possible observation sequences that could be obtained, resulting in a set of possible expected rewards and posterior beliefs. The posterior beliefs are sequentially updated according to Equation 2.3. For each posterior belief, a new set of macro-actions is automatically generated, and these are used to expand the forward search tree further (refer to Figure 3-1 in Chapter 3). We repeat this process until the tree depth reaches a pre-specified horizon H. The expected values of the beliefs are then carried back up the tree by taking the maximum expected value over the macro-actions at each belief and the sum over observation sequences.

4.3 Anytime iterative refinement

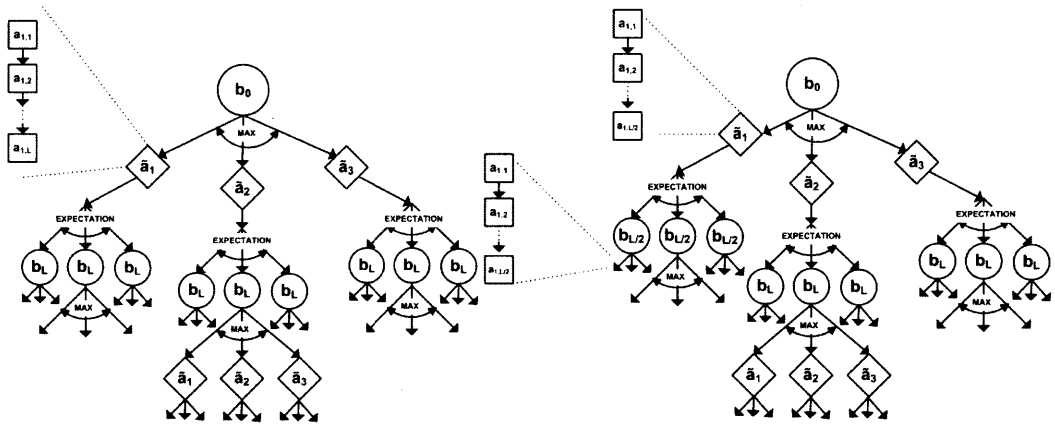
One limitation of our notion of generating macro-actions is that plans based only on open-loop action sequences between sub-goals may not correspond to the optimal, or even a good policy. In particular, so far we have assumed that the user can provide a reasonable estimate of the maximum macro-action length L. However, it may not be obvious how to set L for a given problem; in some cases where fully-conditional planning is required for good performance, every primitive action should depend on the previous observation and L should be 1. To handle this issue, we extend the planner to allow the algorithm to converge to ϵ-optimal performance for any finite-state POMDP in an any-time manner.

If additional computation time is available after the initial tree is built and evaluated, we iteratively refine macro-action branches in the forward-search tree. To select the next macro-action and belief node to refine, we first find the prior belief node that is nearest to the root belief node and has a macro-action that is of length $L > 1$. Of the macro-actions that were previously simulated from that prior belief node, we choose the macro-action has the longest length. Ties are broken arbitrarily. Once a macro-action $\tilde{a}_{refine}$ and belief node b_{refine} is chosen, the sequence of actions in the macro-action is pruned from its original length ($Length(\tilde{a}_{refine})$) to

Algorithm 5 PUMA

Require: Current belief b_c, Planning horizon H, Planning time t_{plan}, Num macros M, Num obs. traj. samples N_z

1: $t_{elapsed} = 0$; $L = H$; $b_e = b_c$
2: **while** $t_{elapsed} < t_{plan}$ **do**
3: $\tilde{\mathcal{A}}$ = GENMACROS(b_e, M, L)
4: **for** each macro-action $\tilde{a}^k \in \tilde{\mathcal{A}}$ **do**
5: **for** $i = 1 : N_z$ **do**
6: Sample observation traj $\tilde{z}^i$ from b_e and $\tilde{a}^k$
7: $b_i = \tau(b_e, \tilde{a}^k, \tilde{z}^i)$ (Equation 2.3)
8: Compute $V(b_i)$ by recursive forward search
9: **end for**
10: $Q(b_e, \tilde{a}^k) = \sum_{i=1}^{N_z} V(b_i)/N_z$
11: **end for**
12: Choose $\tilde{a}_{refine}$ and b_{refine} to refine
13: $L = \lfloor Length(\tilde{a}_{refine})/2 \rfloor$; $b_e = b_{refine}$
14: Go to Line 2
15: **end while**
16: Return $\tilde{a}_{opt} = \text{argmax}_{\tilde{a}}\, Q(b_c, \tilde{a})$



(a) Before refinement (b) After refinement

Figure 4-1: Refining the macro-action forward search tree. The length of the macro-action is pruned to half its original length, and new posterior beliefs are created at the end of the pruned macro-action.

a maximum length of $\lfloor Length(\tilde{a}_{refine})/2 \rfloor$. If the refined, shorter macro-action has already been evaluated for that particular belief, this new macro-action is discarded. Otherwise, a new belief node b' is created by propagating b_{refine} forward according to the pruned macro-action, and $V(b')$ is computed using forward search as before. Note that none of the tree under the previous macro-action can be reused — the forward search under the new (shorter) macro-action must be regenerated from scratch. Figure 4-1 shows an example where the root macro-action $\tilde{a}_1$ is selected to be refined. The new value is propagated up the tree, and if the root belief node b_c now has a macro-action with a higher expected value, the best macro-action $\tilde{a}_{opt}$ is updated. This cycle repeats until all computation time available for planning t_{plan} has expired; this process

gradually increases the amount of conditional branching in the tree.

The process of generating macro-actions, performing an initial search and then iterative improvement is our complete *planning under uncertainty with macro-actions* algorithm, PUMA. Algorithm 5 shows the full PUMA algorithm. As computational resources for planning are available, PUMA will in the limit compute a fully-conditional policy. Attaining a fully-conditional policy provides a guarantee on the global optimality of the computed policy, and removes a critical limitation of many macro-action planners which do not guarantee good performance on problems that require a fully-conditional policy.

4.4 Analysis

Theorem 4.4.1 *Let H be the primitive-action horizon length used in the PUMA algorithm, and assume that the POMDP problem has a finite set of states, actions and observations. Assume that at least $|\mathcal{Z}|$ unique observation sequences are sampled per macro-action. Then, as the amount of computational time available for selecting an action increases, the PUMA policy followed will converge to an ϵ-optimal policy where*

$$\epsilon = \frac{\gamma^{H+1} \max_{a,s} R_S(s,a)}{1-\gamma}. \tag{4.4}$$

Proof First note that the PUMA macro-action creation process ensures that at a given macro-action branching point, there are always at least $|\mathcal{A}|$ *unique* macro-actions. New alternate macro-actions are generated if some of the existing macro-actions are the same. In addition, at least $|\mathcal{Z}|$ unique observation sequences are sampled for each given macro-action. As the amount of time available to select an action for the root belief increases, the PUMA splitting criterion ensures that eventually all macro-actions in the forward search tree will be refined to length one. Combined with the prior condition on the number of unique macro-actions and the stated assumption on the number of unique sampled observation sequences, the forward search tree will become a fully-conditional H-depth tree, with all actions are expanded at each depth, followed by all observations. Therefore, the action-values at the root belief will be exactly the H-step value of taking those actions from the root belief.

Recall from Theorem 1 in Chapter 3 that the difference between the belief-action H-horizon optimal value $Q_H(b,a)$ and the belief-action infinite-horizon optimal value $Q^*(b,a)$ is bounded:

$$Q^*(b,a) - Q_H(b,a) \leq \frac{\gamma^{H+1} \max_{a,s} R_S(s,a)}{1-\gamma} \tag{4.5}$$

Therefore, since in the limit of computational time PUMA converges to the optimal H-horizon policy for the current belief, the value of the PUMA policy is guaranteed to be at worst ϵ-close to the optimal infinite-horizon policy.

In addition, note that Theorem 1 provides a principled way to select the planning horizon H based on the desired ϵ value, such that $H \geq \frac{\epsilon(1-\gamma)}{(\ln \gamma \max_{a,s} r(s,a))} - 1$

4.4.1 Performance analysis

In general, the values computed at the root are only estimates of the optimal belief-action values, with no finite-sample guarantees of how performance changes as more planning time is provided. However, if all observation trajectories are exhaustively enumerated and weighted by their probability, then the computed root belief-action values are the exact belief-action values of following the macro-action tree policy.

We would like to ensure that after a macro-action refinement, the new (exact) computed values at the root node will at least be the same value as they were before the split occurred, if not higher. This result will enable us to commit additional computational time towards refining the tree further without worrying that the algorithm's performance will deprove. To achieve this result, when a length-L macro action is refined, the new $\lfloor L/2 \rfloor$-length macro-actions always include the first and second half of the original length-L macro-action. This assumption implies that the new policy space after a split has occurred always includes the original policy space, plus new potential policies due to the additional conditional branching. In this case, the computed root-belief node value is guaranteed to monotonically increase as the tree is iteratively refined given more planning time.

Due to the prohibitive computational cost of enumerating all observation sequences in large horizons, for our experiments we sample a small number of observation trajectories per macro-action. This yielded good empirical performance.

4.4.2 Computational complexity

The computational complexity of PUMA is a function of the number of nodes in the forward search tree, plus an additional cost for macro-action generation. MDPs with different sampled sub-goals can be solved once and the solutions stored, amortizing the cost over multiple steps of planning. Sampling a macro-action of length greater than 1 for a particular belief requires an $\mathcal{O}(|\mathcal{S}| + L)$ operation — $\mathcal{O}(|\mathcal{S}|)$ to sample a state from the original belief, and a constant cost if hash tables are used to get to the next state, for L iterations. Belief update and expected reward calculations, order $\mathcal{O}(|\mathcal{S}|^2)$ and $\mathcal{O}(|\mathcal{S}|)$ operations respectively, must be performed at each node in the tree.

The number of nodes in the tree depends on the macro-action length. As we have discussed in Chapter 3, if the macro-actions are initially of length L, the desired horizon depth is H, and N_z unique observation trajectories are sampled per macro-action, then the number of forward search tree nodes expanded is $\mathcal{O}(|\tilde{\mathcal{A}}|^{H/L} N_z^{H/L})$.

Therefore the total computational cost to compute this tree is the cost for solving the original sub-goal MDPs, plus

$$\mathcal{O}((|\tilde{\mathcal{A}}|(|\mathcal{S}| + L))^{H/L} + |\tilde{\mathcal{A}}|^{H/L} N_z^{H/L}(|\mathcal{S}|^2 + |\mathcal{S}|)), \tag{4.6}$$

where the first term is due to computing macro-actions, and the second term is due to the cost of performing

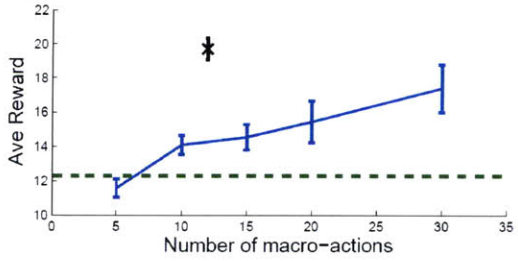


Figure 4-2: Performance of PUMA (blue line) without refinement for ISRS improves when more macro-actions are used, outperforming SARSOP (green dotted line) and approaching the performance of a forward search algorithm which uses hand-coded macro-actions (black cross).

belief updating and expected reward computations per node.

As we iteratively refine each macro-action, the number of nodes grows from $\mathcal{O}(|\tilde{\mathcal{A}}|^{H/L} N_z^{H/L})$ to a full forward search $\mathcal{O}(|\mathcal{A}|^H |\mathcal{Z}|^H)$. Therefore the PUMA cost converges to the cost of a full forward search, plus the overhead of iterative macro-action refinement.

4.5 Experiments

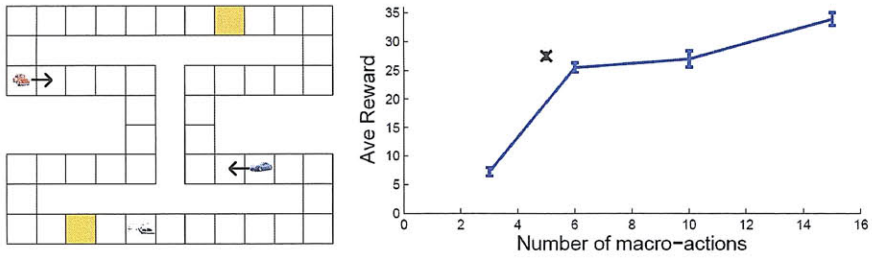
The PUMA algorithm consists of two phases: an initial search process using automatically generated macro-actions, and then a subsequent phase of any-time iterative improvement. In order to analyze these two phases, we first evaluated PUMA with no iterative improvement, and then examined the effect of the refinement process.

4.5.1 Experimental results: Macro-action generation

Because of our desire to develop a domain-independent macro-action algorithm, we adopt the recently proposed POMDPX representation from the Approximate POMDP Planning (APPL) Toolkit[3] to represent our POMDP problems. POMDPX allows POMDP problems to be described using a factored representation, which allows a forward search algorithm such as PUMA to directly take advantage of independencies in the problem model for efficient belief updating. We first implemented the PUMA algorithm without iterative improvement on the ISRS problem that was presented in Chapter 3. The PUMA algorithm was compared to SARSOP (Kurniawati et al., 2008), a state-of-the-art offline POMDP planner, as well as a macro-action planner that used the hand-coded macro-actions that we had specified in Section 3.4.1.

We compared PUMA's performance as we varied M, the number of macro-actions that were sampled. As we increase M, PUMA quickly achieves performance equivalent to a macro-action algorithm that uses the hand-coded macros (Figure 4-2) at a slightly larger computational cost. This suggests that our macro-action generation algorithm is a good heuristic for finding effective macro-actions. H was limited to 15 for

[3] Approximate POMDP Planning Toolkit. `http://bigbird.comp.nus.edu.sg/pmwiki/farm/appl/`



(a) Target-tracking (50)

(b) PUMA performance (Tracking)

Figure 4-3: (a) Tracking: Yellow squares are regions of high value. When the agent (helicopter) is over a target (cars) as the target enters a yellow square, a large positive reward is obtained. If a target enters a yellow square without the agent present, a large negative reward is gained. (b) Performance of PUMA (blue line) without refinement improves when more macro-actions are used, outperforming a forward search algorithm which uses hand-coded macro-actions (black cross). SARSOP was not compared due to the large state space.

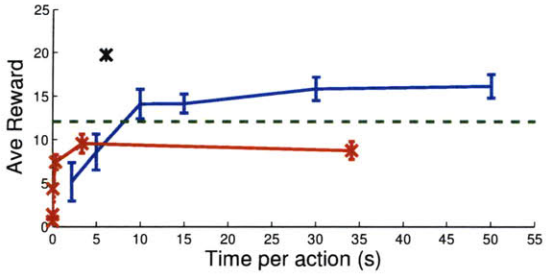
computational reasons. L was set to 5, based on experimental evidence that forward-search action branching was typically able to condition at three points along the planning horizon ($H/3 = 5$).

Target-tracking

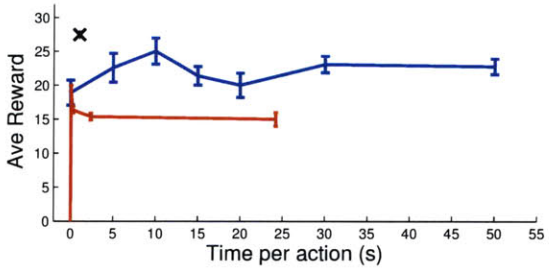
As a second example, we introduce a larger, multi-target extension of the single target-tracking problem introduced by Hsu et al. (2008): see Figure 4-3a. The original problem consisted of an agent and a target moving along a closed track, and the objective of the agent is to localize the target while minimizing the distance traveled. We generalize this problem to two targets. At every timestep, each of the two targets can move zero, one or two steps forward, each with certain non-zero probability, on a track with 50 states. The track is closed, and therefore forms a loop. The actions taken by the targets are hidden from the agent, though the agent's planner is given knowledge of the transition model of each of the targets. The agent can similarly move along the track one or two steps forward, one or two steps backwards, or stay at the same spot. The agent receives a large positive reward if it is in the vicinity of the targets whenever either of the targets land in the yellow square regions, and a negative reward if it is far from the target when the target enters a yellow region. A small cost is incurred for every movement action. Finally, although the target's actions are hidden from the agent, the agent is equipped with a noisy, limited range sensor, and obtains a noisy observation of the target's position when it is in the vicinity of the target.

Although this problem is inherently factored, in terms of the states and transition dynamics of the agent and two targets, offline approaches are typically unable to take advantage of the factored properties of the problem domains, as discussed in Section 2.2.2. Therefore, even though the problem only has 50 environmental states, the non-factored model has $50^3 = 125,000$ states. Without using sparse representations of the system models, this problem domain is not representable, let alone solvable, by offline non-factored approaches. [4] In contrast, the forward search approach allows us to exploit the factored nature of the problem

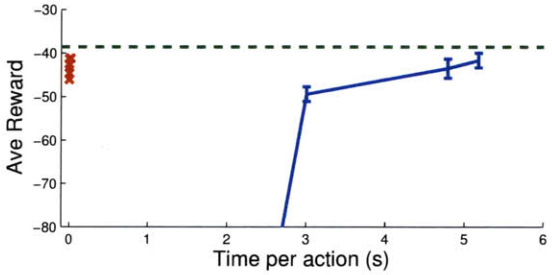
[4] For example, the non-sparsified transition matrix alone would have required $|\mathcal{S}|^2|\mathcal{A}| = 625GB$ of memory.



(a) ISRS (8,5), PUMA 15 macro-actions



(b) Target-tracking (50), PUMA 5 macro-actions



(c) Noisy Tiger, PUMA 3 macro-actions

Figure 4-4: Performance of PUMA (blue line) as more planning time is allocated for iterative refinement, compared to traditional forward search (red line) searching out to a progressively longer horizon, SARSOP (green dotted line), and a hand-coded macro-action forward search algorithm (black cross).

domain easily. We therefore compared PUMA to a traditional forward search, as well as a macro-action forward search algorithm where the macro-actions were specified using domain knowledge. These hand-coded macro-actions consisted of the action sequence that would enable the agent to reach the two regions of interest and each target's most likely position in the shortest number of timesteps, as well as to stay in the same position for a pre-defined number of timesteps.

Figure 4-3b shows that the PUMA algorithm not only performed better as more macro-actions were automatically generated, but also outperformed a policy computed using hand-coded macro-actions when a sufficiently large number of macro-actions were automatically generated.

4.5.2 Experimental results: Iterative refinement

We examined the performance of iterative refinement on ISRS and tracking. As the amount of available online computation time increases, the iterative refinement of the macro-action forward search tree improved performance in the ISRS problem, as shown in Figure 4-4a. Using 15 macro-actions at each branch point, the algorithm evolved from being an unconditional planner ($L = H$), where performance was worse than both a traditional forward search and SARSOP, into a semi-conditional planner where conditioning at different points in the planning horizon resulted in better performance than alternative POMDP approaches. It is

worth noting that the performance improvement tapered off beyond 10 seconds of planning time per action, suggesting that increasing the amount of conditioning became progressively less valuable, and that better performance may be obtained by increasing the number of macro-actions being evaluated and/or increasing the planning horizon.

For the target-tracking problem, iterative refinement also improved performance (Figure 4-4b), though empirically the performance did not improve monotonically with more planning time. More analysis will be required to fully understand why performance degraded as the forward search tree was refined, though we currently hypothesize that it is either due to sampling error or that bad conditioning points were chosen. These additional conditioning points were not only unnecessary, but also resulted in an inaccurate evaluation of the relative value of the new macro-actions, and by extension, an inaccurate evaluation of the corresponding primitive actions.

We also applied PUMA to a POMDP benchmark problem, Tiger, to demonstrate that the design of PUMA still allows good performance on problems that are not well suited to macro-actions, albeit at higher computational cost. Tiger is a small POMDP problem that offers two challenges to semi-conditional planners. First, it requires fully conditional planning to achieve good performance. Second, good action sequences depend on multiple "listen" actions that do not actually change the state itself: approaches such as the Milestone Guided Sampling (MiGS) algorithm of Kurniawati et al. (2009) that generate macro-actions based only on sampling state sub-goals will struggle in this domain. To require planning to a longer horizon, the observation noise was increased from the standard model of 0.15 noise to 0.35 noise, such that the agent has a lower probability of obtaining an accurate observation when it executes the "listen" action: we call this "Noisy Tiger." PUMA iteratively refines the macro-actions until it reaches a primitive forward search of horizon 5, where all primitive actions are considered at each depth: Figure 4-4c shows that although PUMA initially performs poorly due to the length and type of macro-actions generated, the iterative refinement process enables the PUMA algorithm to converge towards the optimal policy as more computation time is made available for planning.

Table 4.1 summarizes the results of the different algorithms for each POMDP problem, reporting the best performance achieved by each algorithm. As anticipated, PUMA performs very well on large simulation problems that are well-suited to macro-actions, such as those where hand-coded macro-action policies also do well. However, even for problem domains that require fully-conditional policies, PUMA can still converge towards optimal performance given sufficient planning time. These two properties make PUMA well-suited to generic POMDP problems that may require a long look-ahead. PUMA will likely quickly achieve good performance if the problem turns out to be amenable to semi-conditional planning, but if not, PUMA is still guaranteed to eventually converge to an ϵ-optimal policy.

	Ave. reward	Offline time	Online time
Noisy Tiger			
SARSOP	-38.61 ± 0.53	0.50	0.000
Naive fs	**-41.19** ± 1.70	0.000	0.018
PUMA	-41.67 ± 1.66	0.000	5.18
ISRS (8,5)			
SARSOP	12.10 ± 0.26	10000	0.000
Naive fs	9.56 ± 1.08	0.000	3.36
Hand-coded	**19.71** ± 0.63	0.000	0.74
PUMA	17.38 ± 1.41	0.000	162.48
Tracking (50)			
Naive fs	19.58 ± 0.42	0.000	0.023
Hand-coded	27.48 ± 0.49	0.000	1.010
PUMA	**35.55** ± 1.28	0.000	28.52

Table 4.1: Summary of experimental results using the best explored parameters for each algorithm.

4.6 Related Work

The fully-observable planning community has explored the challenge of automatically learning macro-actions in order to plan in a domain-independent fashion. Iba (1989) learns macro-actions by using the "peak-to-peak" heuristic to detect peaks along evaluation functions, where a peak is defined as a node whole value is greater than each of the two adjacent nodes along the given path. Botea et al. (2005) propose a method for learning macro-actions as part of their Macro-FF algorithm, analyzing problem domain and structural information to propose candidate macro-actions, before using a filter and ranking system to select the most useful macro-actions. Newton et al. (2007) go further by using a genetic algorithm to learn from plans of small problems for evaluation on larger problems. The intuition here is that by learning on problem domains of different sizes, the characteristics of good macro-actions will emerge. McGovern (1998) provides a method for growing macro-actions in a fully-observable reinforcement-learning framework by finding finding peak points in the temporal history of rewards, as well as examining state visitation frequencies across the trajectories to identify useful sub-goals for the macro-actions. McGovern and Barto (2001) subsequently introduce another method that generates sub-goals by searching for bottlenecks in the agent's observation space, coupled with a learning method that finds regions that the agent frequently visits on successful trajectories but not on unsuccessful ones. Finally, Hauskrecht et al. (1998) define macro-actions as local closed-loop policies over a region of the state space, performing a region-based decomposition of the state space to generate candidate macro-actions. Using an abstract MDP consisting only of states that lie on the borders of adjacent regions, a hierarchical macro-policy is then obtained mapping peripheral states to macro-actions.

Moving back to the partially-observable domain, we discussed in Chapter 3 that POMDP macro-action planners by Theocharous and Kaelbling (2003), and Hsiao, Lozano-Pérez and Kaelbling (2008) provide good empirical performance, but do not focus on learning the macro-actions or on providing performance guarantees. The closest related approach to PUMA is the MiGS algorithm by Kurniawati et al. (2009). As

discussed in Section 3.6.1, the authors use macro-actions to reduce the number of the beliefs sampled over a long horizon, and iteratively refine these beliefs. They also sample milestone, or sub-goals, by using a heuristic that estimates the expected reward and localizability at each state, though the performance metric used is significantly more complicated and requires more knobs to be tuned. In addition, the differences between using macro-actions in an online and offline manner, presented in Section 3.6.1, such as the ability to scale better in large spaces and exploit factored models for quicker belief updates, are also applicable when comparing between PUMA and MiGS. Finally, unlike MiGS, PUMA, with sufficient planning time, will produce good solutions for generic POMDPs, including domains like Tiger which require sensing-only action sequences. The current implementation of MiGS will struggle because it only constructs macros by sampling states, though it should also be possible for MiGS to be extended to include an iterative refinement component.

4.7 Conclusion and future work

In summary, we have presented an anytime semi-conditional forward-search planner for partially observable domains. PUMA automatically constructs macro-actions and iteratively refines the macro-actions as planning time allows. PUMA achieves promising experimental results relative to state-of-the-art offline planners, as well as a similar planner that uses hand-selected macro-actions, both on several large domains that require far look-ahead, and also on domains that require fully-conditional planning.

There are several interesting avenues for future work. We currently use standard MDP solution techniques to generate macro-actions, computing a policy over the entire (potentially very large) state space. It would be interesting to investigate a more scalable alternative, using efficient open-loop forward search algorithms like the rapidly-exploring random trees (RRT) to find a feasible path (and corresponding macro-action) between two points in high-dimensional search spaces.

Another interesting issue is deciding the length of each macro-action during refinement. We currently fix the length of the macro-action after refinement to be half the original length, but it may be possible to obtain better performance by explicitly considering the best point along the macro-action to perform additional conditional planning. We are also exploring other splitting criteria for deciding the best macro-action to refine next.

Finally, when additional computational time is available, it may be more valuable to increase the planning horizon H, rather than increasing the amount of conditioning in the forward search tree with a fixed planning horizon. It is therefore also worth exploring how the planner can trade-off increasing the amount of conditioning (breadth of tree) with increasing the planning horizon (depth of tree).

Chapter 5

PBD: Efficiently predicting the distribution of posterior beliefs

(Joint work with Emma Brunskill)

In the previous chapter, we addressed the technical challenge of automatically generating macro-actions for semi-conditional planning, as well as a method for iterative refinement so as to address planning under uncertainty problems where it may be necessary for more conditioning to obtain good performance. In the next three chapters, we address the second technical challenge of efficiently computing the set of posterior beliefs that can be obtained after a macro-action is executed.

We have shown that planning with macro-actions in a forward search manner leads to a substantial reduction in computational cost for problem domains that require planning to a long horizon. However, during the execution of each macro-action, the agent will still receive one of a large set of possible observation sequences. Assuming that every possible observation sequence is enumerated, the complexity of the search remains an exponential function of the horizon length and the number of primitive actions. While this computational cost can be reduced by sampling observation sequences, as we have done with the general macro-action framework in Chapter 3, the number of samples necessary for a good estimate of the expected reward can be substantial, and the cost of belief updating per sample can be expensive. Therefore, there is considerable scope to further improve the efficiency of forward search by efficiently handling the observation branching.

In this chapter, we exploit the structure inherent in specific types of environments in order to perform efficient macro-action forward search. One of our core contributions is an analytic method for computing the distribution over the set of posterior beliefs that could result from a single macro-action, so long as the agent's belief of the world can be represented as a Gaussian distribution. This posterior belief distribution represents the potential resulting posterior beliefs given all possible observation sequences that the agent may receive during the macro-action, and therefore removes the need to explicitly enumerate the observation sequences.

In addition, for a large class of reward models, we can directly compute the rewards associated with the distribution of posterior beliefs, which then allows us to analytically compute the expected reward associated with a macro-action without having to enumerate all possible observation sequences and individually compute the rewards associated with each corresponding posterior belief. The ability to compute a distribution over posterior beliefs forms the core of our POSTERIOR BELIEF DISTRIBUTION (PBD) algorithm. We propose two variants of our PBD algorithm: an open-loop macro-action planner with performance guarantees, and a conditional macro-action planner that in practice outperforms the open-loop planner but does not have the same associated performance guarantees. In addition, we provide techniques that allow an approximate computation of the posterior belief distribution and expected rewards.

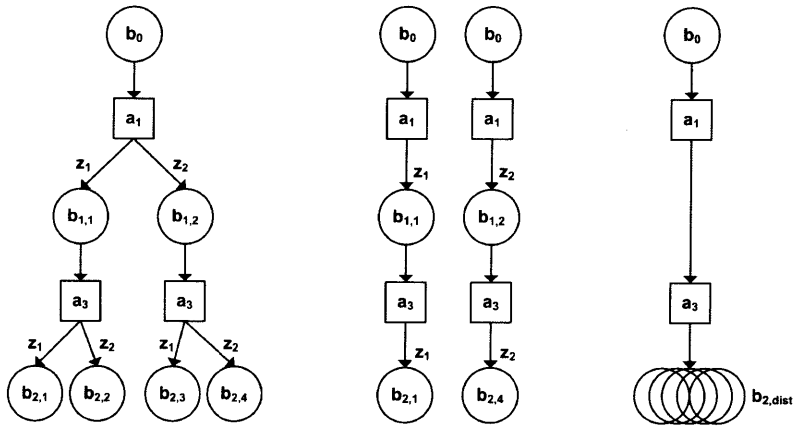
In problem domains that are not well-captured by the continuous, parametric representations of the PBD algorithm, such as some discrete environments, we cannot analytically represent the resulting distribution of beliefs after a macro-action. Instead, we use the generic approach that had been presented in Chapter 3, sampling observation sequences to produce a particle approximation of the resulting distribution over beliefs. Following the naming conventions in Chapter 3, we refer to the approach that uses a discrete state representation as a MACRO-ACTION DISCRETE (MAD) forward search planner, while the MACRO-ACTION CONTINUOUS (MAC) algorithm adopts the same parametric representation as the PBD algorithm but samples observation sequences to obtain the set of posterior beliefs. As we have demonstrated in Chapter 3, when the underlying environmental models exhibit a factored structure, MAD significantly outperforms prior approaches.

In this chapter, we present a formal analysis of the PBD algorithm and analyze its performance and computational complexity (Section 5.2). This analysis shows that relative to alternative approaches, PBD possesses a significant reduction in computational complexity due to the use of a (possibly approximate) analytic expression for the resulting distribution over beliefs (Section 5.2.2). Finally, in the next chapter (Chapter 6), we highlight the promise of our macro-action planners by presenting experimental results, both in simulation and on actual robotics hardware.

5.1 Computing the posterior belief distribution

For a single L-length macro-action, there are still $|\mathcal{Z}|^L$ potential observation sequences that could be received at the end of the macro-action, where $|\mathcal{Z}|$ is the number of possible observations. If we exhaustively enumerated all potential observation sequences (Figure 5-1a), and performed belief updating along each sequence, that would yield a set of $|\mathcal{Z}|^L$ possible posterior beliefs after a single macro-action.

The expected immediate reward received during a macro-action, and the expected future reward received after a macro-action, are both functions of the set of potential posterior beliefs resulting from a macro-action, rather than being a function of a single belief. A natural question then is how to obtain the set of posterior beliefs that could result after a single macro-action. Exhaustive enumeration of observation sequences will



(a) Exhaustive (b) Sampled (c) Analytic

Figure 5-1: Three methods to represent the resulting set of beliefs after a single macro-action. (a) All possible observations are expanded. (b) A subset of possible observation trajectories are sampled. (c) Compute an analytic distribution over the posterior beliefs, which could have been generated via an exhaustive enumeration of all possible observation sequences. b_0 is the initial belief, while $b_{i,j}$ refers to the j^{th} belief leaf node at depth i.

be intractable when there are a large number of observations, or when the macro-action consists of a long list of primitive actions. In the prior subsection we presented one alternative, which is to sample observation sequences for a given macro-action; see Figure 5-1b for an illustration. Sampling observation trajectories yields a particle approximation of the true posterior distribution of beliefs. However, as we will discuss further in Section 5.2.2, sampling can still be computationally intensive due to the number of belief updates and expected reward calculations that must occur at each step of each sampled observation sequence.

Instead, we show that for specific kinds of world models, we can analytically compute the distribution over beliefs that arises from a macro-action without sampling observations: a graphical depiction of this process is shown in Figure 5-1c. By analytically computing a distribution over beliefs, we completely avoid both the exponential computational dependence on the original horizon length, and also the costly step of performing belief updating along each observation trajectory associated with the macro-action.

Specifically, when the agent's belief is a Gaussian, the distribution over posterior beliefs is itself a Gaussian over Gaussian beliefs, as illustrated in Figure 5-2. We will show that when the Gaussian belief update is exact, all posterior beliefs in this distribution have the same covariance, and the posterior means are normally distributed over the continuous state space. Given an action sequence, the posterior distribution over beliefs is therefore a joint distribution over the posterior means and the corresponding distribution over states.

5.1.1 Exact computation of posterior belief distribution

Let us assume for now that the agent's belief can be exactly represented as a Gaussian distribution over a continuous state space, and that the observation and transition models are both linearGaussian. Formally, the

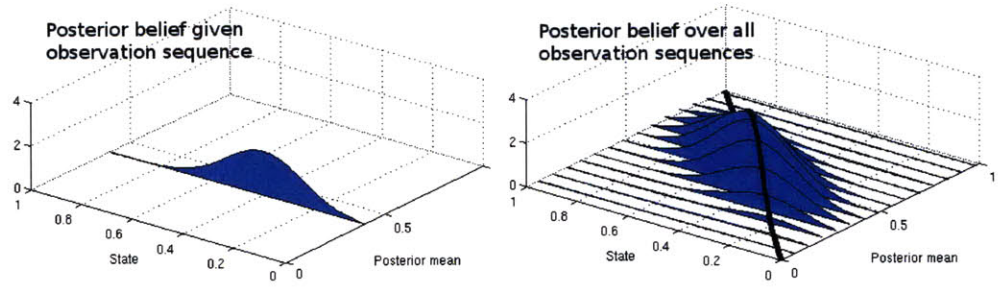


Figure 5-2: Distribution of posterior beliefs. a) A Gaussian posterior belief results after incorporating an observation sequence. b) Over all possible observation sequences, the distribution of posterior means is a Gaussian (black line), and for each posterior mean, a Gaussian (blue curve) describes the agent's posterior belief.

state transition and observation models can be represented as follows:

$$s_t = A_t s_{t-1} + B_t a_t + \varepsilon_t, \qquad \varepsilon_t \sim \mathcal{N}(0, P_t) \tag{5.1}$$

$$z_t = C_t s_t + \delta_t, \qquad \delta_t \sim \mathcal{N}(0, Q_t) \tag{5.2}$$

where A_t and B_t are dynamics matrices, C_t is the observation matrix, P_t is the covariance of the Gaussian dynamics process and Q_t is the covariance of the measurement noise.

When the state-transition and observation models are normally distributed and linear functions of the state, the Kalman filter (Kalman, 1960) provides a closed-form solution for the posterior belief $\mathcal{N}(\mu_t, \Sigma_t)$ given a prior belief $\mathcal{N}(\mu_{t-1}, \Sigma_{t-1})$,

$$\overline{\mu}_t = A_t \mu_{t-1} + B_t a_t \qquad \mu_t = \overline{\mu}_t + K_t(z_t - C_t \overline{\mu}_t) \tag{5.3}$$

$$\overline{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + P_t \qquad \Sigma_t = (C_t^T Q_t^{-1} C_t + \overline{\Sigma}_t^{-1})^{-1}, \tag{5.4}$$

where $\mathcal{N}(f, F)$ is a D-dimensional Gaussian with mean f and covariance matrix F,
$K_t = \overline{\Sigma}_t C_t^T (C_t \overline{\Sigma}_t C_t^T + Q_t)^{-1}$ is the Kalman gain and $\overline{\mu}_t$ and $\overline{\Sigma}_t$ are the mean and covariance after an action is taken but before incorporating the measurement.

Mean update

We are interested in computing an expression for the distribution over the mean of the posterior beliefs under any possible observation. To do this, we first re-express the observation model as

$$p(z_t | s_t) = \mathcal{N}(C_t s_t, Q_t) \tag{5.5}$$

which we can use to compute an expression for the probability of an observation given the belief mean, $p(z_t|\overline{\mu}_t)$, by marginalizing over $s_t \sim \mathcal{N}(\overline{\mu}_t, \overline{\Sigma}_t)$:

$$p(z_t|\overline{\mu}_t) = \int p(z_t|s_t, \overline{\mu}_t)p(s_t|\overline{\mu}_t)ds_t \tag{5.6}$$

$$= \mathcal{N}(C_t\overline{\mu}_t, C_t\overline{\Sigma}_tC_t^T + Q_t). \tag{5.7}$$

Using this expression, and performing linear transformations, we can obtain an expression for the distribution of posterior means, under any potential observation:

$$p(z_t|\overline{\mu_t}) = \mathcal{N}(C_t\overline{\mu}_t, C_t\overline{\Sigma}_tC_t^T + Q_t) \tag{5.8}$$

$$p(z_t - C_t\overline{\mu}_t|\overline{\mu_t}) = \mathcal{N}(0, C_t\overline{\Sigma}_tC_t^T + Q_t) \tag{5.9}$$

$$p(K_t(z_t - C_t\overline{\mu}_t)|\overline{\mu_t}) = \mathcal{N}(0, K_t(C_t\overline{\Sigma}_tC_t^T + Q_t)K_t^T) \tag{5.10}$$

$$p(\overline{\mu}_t + K_t(z_t - C_t\overline{\mu}_t)|\overline{\mu}_t) = \mathcal{N}(\overline{\mu}_t, K_t(C_t\overline{\Sigma}_tC_t^T + Q_t)K_t^T) \tag{5.11}$$

$$p(\mu_t|\overline{\mu}_t) = \mathcal{N}(\overline{\mu}_t, K_t(C_t\overline{\Sigma}_tC_t^T + Q_t)K_t^T) \tag{5.12}$$

$$p(\mu_t|\overline{\mu}_t) = \mathcal{N}(\overline{\mu}_t, \overline{\Sigma}_tC_t^TK_t^T) \tag{5.13}$$

where Equation 5.13 is computed by substituting the definition of the Kalman gain.

At this point, a somewhat unusual change has occurred, in that μ_t, the mean of the distribution itself, is now a random variable. Without knowing the value of the particular observation that occurs after a primitive action, we cannot deterministically predict the posterior mean and covariance of the belief. However, we can model the probability of any specific belief state, which effectively means that we will compute a distribution over the belief means μ and variances Σ. Equation 5.13 shows that the distribution over the belief means is normally distributed about $\overline{\mu}_t$, with a covariance that depends on the prior covariance $\overline{\Sigma}_t$ and the observation model parameters. Sampling a mean from this distribution is equivalent to selecting a particular observation.

We can use Equation 5.13 to compute the posterior distribution over beliefs for a particular action sequence and any possible observation sequence: this will allow us to compute an analytic expression for the posterior distribution over beliefs that could result from a single macro-action. To compute the posterior distribution over belief means, we first combine the process and measurement updates for a single primitive action belief update in order to get an expression for the posterior belief means in terms of the prior belief mean. We do this by marginalizing out $\overline{\mu}_t$, the posterior belief after the transition update but before the observation update, using $p(\mu_t|\mu_{t-1}) = \int p(\mu_t|\overline{\mu}_t)p(\overline{\mu}_t|\mu_{t-1})d\overline{\mu}_t$. As $\overline{\mu}_t$ is a deterministic function of μ_{t-1} (see Equation 5.3a), then $p(\overline{\mu}_t|\mu_{t-1})$ is simply a delta function, which means that $p(\mu_t|\mu_{t-1})$ is identical to Equation 5.13 after substituting $\overline{\mu}_t$ using Equation 5.3a:

$$p(\mu_t|\mu_{t-1}) = \mathcal{N}(A_t\mu_{t-1} + B_ta_t, \overline{\Sigma}_tC_t^TK_t^T). \tag{5.14}$$

In a one-step belief update, the belief mean at the prior timestep, μ_{t-1}, was assumed to be a fixed value. However, for a macro-action, once the first primitive action has been taken, the posterior belief mean will depend on the received observation. In absence of the knowledge of that received observation, we will instead have a distribution over the belief means, which may be represented as a Gaussian $\mu_{t-1} \sim \mathcal{N}(m_{t-1}, \Sigma^{\mu}_{t-1})$ where m_{t-1} and Σ^{μ}_{t-1} are random variables. In order to compute the probability distribution over μ_t, we must integrate over this distribution of prior belief means μ_{t-1}:

$$p(\mu_t | m_{t-1}, \Sigma^{\mu}_{t-1}) = \int_{\mu_{t-1}} p(\mu_t | \mu_{t-1}) p(\mu_{t-1} | m_{t-1}, \Sigma^{\mu}_{t-1}) d\mu_{t-1}. \tag{5.15}$$

This shows that the mean of the posterior belief means is a Gaussian distribution over a function of the prior mean of the belief means and covariance:

$$\mu_t \sim \mathcal{N}(A_t m_{t-1} + B_t a_t, \Sigma^{\mu}_{t-1} + \overline{\Sigma}_t C_t^T K_t^T) \tag{5.16}$$

or

$$\mu_t \sim \mathcal{N}(m_t, \Sigma^{\mu}_t) \tag{5.17}$$

where $m_t = A_t m_{t-1} + B_t a_t$ and $\Sigma^{\mu}_t = \Sigma^{\mu}_{t-1} + \overline{\Sigma}_t C_t^T K_t^T$. Equation 5.16 can now be used to perform a prediction of the posterior mean distribution after a multi-step action sequence. Assuming that the agent is currently at time t and has a particular prior mean μ_{t-1} (which we can also express as a Gaussian with zero covariance, $\mathcal{N}(\mu_{t-1}, 0)$), the posterior mean after an action sequence of T timesteps is distributed as follows:

$$\mu_{t+T} \sim \mathcal{N}(f(\mu_{t-1}, A_{t:t+T}, B_{t:t+T}, a_{t:t+T}), \Sigma^{\mu}_{t:T}) \tag{5.18}$$

where $f(\mu_{t-1}, A_{t+1:t+T}, B_{t+1:t+T}, a_{t+1:t+T})$ is the deterministic transformation of the means according to $\mu_{t+k} = A_{t+k}\ \mu_{t+k-1} + B_{t+k}\ a_{t+k}$ and $\Sigma^{\mu}_{t:T} = \sum_{i=t}^{t+T} \overline{\Sigma}_i C_i^T K_i^T$. Since an observation on its own does not shift the mean value m_{t+k} of the distribution of posterior means, m_{t+k} is dependent only on the state-transition model parameters and can be calculated via a recursive update along the action sequence.

Covariance update

The posterior covariance of the agent's belief after an action is taken and an observation is received can be calculated using Equation 5.4. These covariance update equations are only dependent on the model parameters of the problem, and are independent of the observations that may be obtained. The posterior covariance can therefore be computed in closed form, and is independent of the posterior mean.

For greater efficiency, it has been shown (Prentice & Roy, 2007) that for a Kalman filter model, the process and measurement updates can be compiled into a single linear transfer function by using a factored

Algorithm 6 MACROEXPAND() (PBDE)

Input: Macro-action $\tilde{a}$, Belief state b_t or posterior distribution over belief b_{dist}, Discount factor γ, Macro-action search depth $\tilde{H}$,
if $\tilde{H} = 0$ **then**
  return 0
else {Expand Macro-action $\tilde{a}$=$\{a^1, \ldots, a^L\}$}
  $V = 0$
  $b_{dist} = b_t$
  for $j = 1$ to L **do**
    $V = V + R_B(b_{dist}, a^j)$
    Update the posterior distribution of beliefs b_{dist} using Equation 5.20
  end for
  Generate next set of macro-actions $\tilde{\mathcal{A}}^{next}$
  for $\tilde{a}_i^{next} \in \tilde{\mathcal{A}}^{next}$ **do**
    $Q(b_{dist}, \tilde{a}_i^{next})$ = MACROEXPAND($\tilde{a}_i^{next}$,b_{dist},γ,$\tilde{H} - 1$)
  end for
  $V = V + \gamma^L \operatorname{argmax}_{\tilde{a}_i^{next}} Q(b_{dist}, \tilde{a}_i^{next}))$
  return V
end if

representation of the covariance, $\Sigma_{t-1} = B_{\Sigma,t-1} C_{\Sigma,t-1}^{-1}$, as well as a matrix inversion lemma. Adapting this result to our model, the complete covariance update step can be written as:

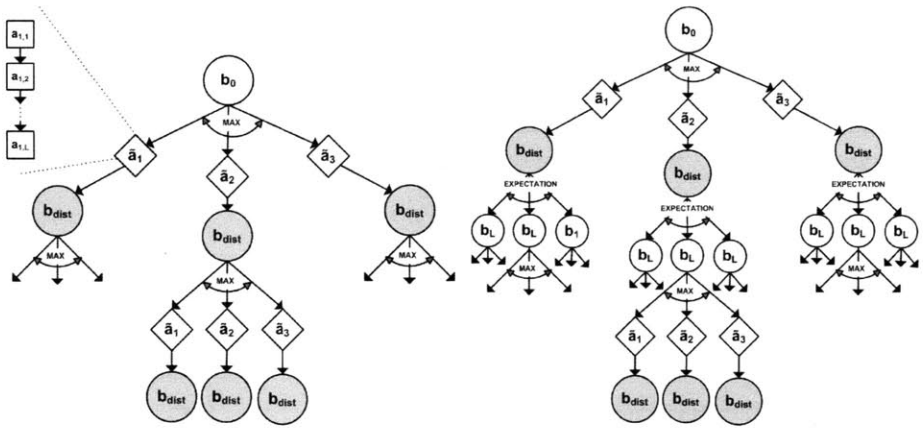
$$\Psi_t = \begin{bmatrix} B_\Sigma \\ C_\Sigma \end{bmatrix}_t = \begin{bmatrix} 0 & I \\ I & CQC^T \end{bmatrix}_t \begin{bmatrix} 0 & A^{-T} \\ A & PA^{-T} \end{bmatrix}_t \begin{bmatrix} B_\Sigma \\ C_\Sigma \end{bmatrix}_{t-1}, \tag{5.19}$$

where the first two matrices on the right are linear functions of the problem model parameters. Equation 5.19 enables us to collapse multi-step covariance updates into a single step $\Psi_{t+1:T} = \prod_{i=1}^{T} \Psi_i$, recovering the posterior covariance Σ_{t+T} from $\Sigma_{t+T} = B_{\Sigma,t+T} C_{\Sigma,t+T}^{-1}$. This single-step update only applies for the covariance; we still need to perform an $\mathcal{O}(n)$ computation to evaluate the posterior mean associated with the macro-action.

We define b_{dist} as the posterior distribution over beliefs after a macro-action. b_{dist} consists of an expression for the distribution over belief means after a macro-action (Equation 5.18) and an expression for the distribution over the covariance of a belief, which is a Dirac delta distribution for the reasons explained in the prior paragraph. More formally,

$$b_{dist}(\mu_{t+T}, \Sigma) = \mathcal{N}(f(\mu_{t-1}, A_{t:t+T}, B_{t:t+T}, a_{t:t+T}), \Sigma_{t:T}^{\mu}) \cdot \delta(\Sigma, \Sigma') \tag{5.20}$$

where $b_{dist}(\mu_{t+T}, \Sigma)$ is the probability of arriving in posterior belief $b = \mathcal{N}(\mu_{t+T}, \Sigma)$ after taking a particular macro-action, and Σ' is computed by iteratively applying Equation 5.4. This expression shows that for problems with linear Gaussian state-transition and observation models, we can exactly calculate the distribution of posterior beliefs associated with a macro-action.



(a) PBDE (b) PBD

Figure 5-3: Planning with posterior belief distributions. a) In PBDE, macro-actions $\tilde{a}^i$ are chained directly. b) In PBD individual beliefs b are sampled from the posterior distribution over beliefs b_{dist}, implicitly sampling a particular observation trajectory. Then the best macro-action is selected for each sampled posterior belief. A sum is taken over all the sampled beliefs, again corresponding to a sum over the implicitly sampled observation sequences. Here, b_i refers to beliefs at macro-action depth i.

5.1.2 Open-loop macro-action planning: PBDE

We can use this approach for directly computing the posterior belief distribution in order to perform efficient macro-action forward search. Given a starting belief state or distribution over beliefs, we first compute an analytic expression for the posterior distribution over beliefs for each macro-action, and then branch on all possible macro-actions after each macro-action. This is repeated out to search depth $\tilde{H}$ in order to obtain the best sequence of macro-actions that the agent could execute: see Figure 5-3a for an illustration. We call this algorithm the POSTERIOR BELIEF DISTRIBUTION EXACT (PBDE) algorithm, and it uses the MACROEXPAND() routine presented in Algorithm 6. In Section 5.2 we provide a detailed analysis of the performance and computational complexity of the PBDE algorithm.[1]

5.1.3 Conditional macro-action planning: PBD

Although PBDE avoids a direct dependence on the timestep horizon H and is only a function of the macro-action horizon $\tilde{H}$, it is not a very useful algorithm since it suffers from an important limitation. Specifically, although it accurately captures the distribution of possible posterior beliefs after multiple macro-actions are executed, it never conditions on a particular sequence of observations that might be obtained. In other words, the macro-actions will be selected to maximize the expected reward assuming that any possible observation sequence might occur. Chaining multiple macro-actions together without ever conditioning the macro-actions selected on the potential observations that could be received is equivalent to planning to take a single, very

[1] Due to the infinite-state problem formulation in this chapter, the ϵ-optimal result proven in Chapter 4 is not applicable.

Algorithm 7 MACROEXPAND() (PBD)

Input: Macro-action $\tilde{a}$, Belief state b_t, Discount factor γ, Macro-action search depth $\tilde{H}$, No. posterior belief samples per macro-action N_s
if $\tilde{H} = 0$ **then**
  return 0
else {Expand Macro-action $\tilde{a}$=$\{a^1, \ldots, a^L\}$}
  $V = 0$
  $b_{dist} = b_t$
  for $j = 1$ to L **do**
    $V = V + R_B(b_{dist}, a^j)$
    Update the posterior distribution of beliefs b_{dist}
  end for
  for $i = 1$ to N_s **do**
    Sample posterior mean n_i according to $\mathcal{N}(m_{t+T}, \Sigma^{\mu}_{t+T})$
    $b_i \leftarrow \mathcal{N}(n_i, \Sigma_{t+T})$
    Generate next set of action sequences $\tilde{\mathcal{A}}^{next}$
    for $\tilde{a}_i^{next} \in \tilde{\mathcal{A}}^{next}$ **do**
      $Q(b_i, \tilde{a}_i^{next})$ = MACROEXPAND($\tilde{a}_i^{next}$,b_i,γ,$\tilde{H} - 1$,N_s)
    end for
    $V = V + \frac{1}{N_s}\gamma^L \operatorname{argmax}_{\tilde{a}_i^{next}} Q(b_i, \tilde{a}_i^{next}))$
  end for
  return V
end if

long macro-action. However, when the agent acts in the world, it will in fact only receive one observation sequence while executing a macro-action. While the intuition of macro-actions is that it is reasonable to act in an open-loop fashion for a little while, the received observation sequence does provide information about the underlying belief that is likely to be useful for selecting future actions. In general it will be helpful to periodically condition future macro-actions on the particular sequence of observations that was actually received. We expect that algorithms that do condition on the potential set of observations received are likely to outperform PBDE and other unconditional policies.

Ideally we would partition the distribution of beliefs after each macro action into subsets which have the same best next conditional policy. However, as we are computing the remaining conditional policy using forward search, we do not yet have access to those future policies. Therefore it is not possible for us to partition the distribution of beliefs into subsets that share the same next best policy. Instead of performing this partitioning, in order to implicitly condition on different observation sequences, we simply sample beliefs from the posterior distribution over beliefs that arises after a macro-action. This process is shown in Figure 5-3b. We continue the forward search from each sampled belief separately by finding the optimal next macro action, and the next macro-action followed can be different for different beliefs arising after a single macro-action. Each sampled belief is implicitly a sample of a particular L-length observation sequence possible given the macro-action, but the observation sequence is sampled without having to actually perform belief updates or expected reward calculations along each (implicitly) sampled observation trajectory. The sampled beliefs essentially form a non-parametric, particle estimate of the posterior distribution of beliefs that is

present after taking the macro-action. As the number of samples N_s goes to infinity, the sampled distribution will become an arbitrarily good approximation of the full posterior distribution of beliefs.

Even though this algorithm has lost some of the computational benefits of using only a closed form evaluation of the posterior belief distribution, sampling directly from the posterior distribution still provides a significant computational advantage over performing belief updates for each observation trajectory. As the covariance is a Dirac delta distribution, sampling is needed only for the posterior mean distribution; posterior belief samples are generated by associating each posterior mean sample with the same posterior covariance Σ_{t+T}. Forward search then proceeds from each of these sampled beliefs. We refer to the process of planning by computing the posterior distribution over beliefs, sampling from this distribution, and then repeating the planning process for each posterior sample as the POSTERIOR BELIEF DISTRIBUTION (PBD) algorithm (Algorithm 7). While sampling from the posterior distribution of beliefs after each macro-action adds an additional computational cost relative to the PBDE algorithm, the resulting cost is still independent of the primitive action horizon. More importantly from a performance perspective, conditioning the future actions based on the (implicit) observation sequences will yield significant performance gains in most domains of interest. This is also true for the macro-action algorithms that explicitly sample observation sequences, though these, as we will shortly see, incur much larger computational costs than PBD.

For the macro-action planners that sample observation trajectories, we use both a discrete state representation (MAD algorithm presented in Chapter 3), and a parametric belief representation (referred to as the MAC algorithm). In Chapter 6, we provide empirical validation of these observations by comparing the PBD algorithm to the MAC and MAD approaches.

5.1.4 Exponential family Kalman filter

Unfortunately, most planning under uncertainty problems have observation functions that are non-Gaussian. When the linear Gaussian assumption does not hold, the traditional Kalman filter model no longer provides an exact belief update, and for the PBD/PBDE algorithms, the distribution of posterior beliefs cannot be calculated exactly. Nevertheless, a more general approximate version of the Kalman filter update exists for problems with observation models that belong to a larger class of parametric distributions, known as the exponential family of distributions (Barndorff-Nielsen, 1979). Examples of exponential family distributions include the Gaussian, Bernoulli, gamma and Poisson distributions, among others. The probability distributions in this class share a certain algebraic representation, and are chosen for mathematical convenience as well as generality.

Building on statistical economics research for time-series analysis of non-Gaussian observations (Durbin & Koopman, 2000), West, Harrison and Migon (1985) showed that a dynamic generalized linear model provides the exponential family equivalent of the Kalman filter (efKF). This allows a closed-form approximation of the posterior belief for a larger class of observation models. The key idea is to construct linear Gaussian models that approximate the non-Gaussian exponential family model in the neighborhood of the conditional

mode, $s_t|z_t$. The approximate linear Gaussian observation model can then be used in a traditional Kalman filter.

A system with linear Gaussian state-transition models and observation models that belong to the exponential family of distributions can be represented as follows:

$$s_t = A_t s_{t-1} + B_t a_t + \varepsilon_t, \qquad s_{t-1} \sim \mathcal{N}(\mu_{t-1}, \Sigma_{t-1}), \qquad \varepsilon_t \sim \mathcal{N}(0, P_t) \tag{5.21}$$

$$p(z_t|\theta_t) = \exp(z_t^T \theta_t - \beta_t(\theta_t) + \kappa_t(z_t)), \qquad \theta_t = W(s_t). \tag{5.22}$$

For the state-transition model, s_t is the system's hidden state, a_t is the control actions, A_t and B_t are the linear transition matrices, and ϵ_t is the state-transition Gaussian noise with covariance P_t.

The observation model belongs to the exponential family of distributions. θ_t and $\beta_t(\theta_t)$ are the canonical parameter and normalization factor of the distribution, and $W(\cdot)$ maps the states to canonical parameter values. $W(\cdot)$ is also known as the canonical link function, and depends on the particular member of the exponential family. For ease of notation, we let

$$\upsilon_t(z_t|\theta_t) = -\log p(z_t|\theta_t) = -z_t^T \theta_t + \beta_t(\theta_t) + \kappa_t(z_t). \tag{5.23}$$

Following the traditional Kalman filter, the process update can be written as

$$\overline{\mu}_t = A_t \mu_{t-1} + B_t a_t, \qquad \overline{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + P_t, \tag{5.24}$$

where $\overline{\mu}_t$ and $\overline{\Sigma}_t$ are the mean and covariances of the posterior belief after the process update but before the measurement udpate.

For the exponential family observation model, constructing the approximate linear Gaussian model requires computation of the first two moments of the distribution and the linearization around the mean estimate at every timestep. The first two moments of the distribution (West et al., 1985) are,

$$E(z_t|\theta_t) = \dot{\beta}_t = \frac{\partial \beta_t(\theta_t)}{\partial \theta_t}\Big|_{\theta_t = W(\overline{\mu}_t)} \quad \text{Var}(z_t|\theta_t) = \ddot{\beta}_t = \frac{\partial^2 \beta_t(\theta_t)}{\partial \theta_t \partial \theta_t^T}\Big|_{\theta_t = W(\overline{\mu}_t)} \quad \theta_t = W(s_t), \tag{5.25}$$

where $\dot{\beta}_t$ and $\ddot{\beta}_t$ are the derivatives of the exponential family distribution's normalization factor, both linearized about $\overline{\theta}_t = W(\overline{\mu}_t)$.

To compute the first two moments of the observation model, we therefore seek to find the conditional

mode

$$\mu_t = \arg\max_{s_t} p(s_t|z_t) \tag{5.26}$$

$$= \arg\max_{s_t} p(z_t|s_t)\overline{b}(s_t) \qquad \text{(Bayes rule)} \tag{5.27}$$

$$= \arg\max_{s_t} p(z_t|\theta_t)\overline{b}(s_t) \tag{5.28}$$

$$= \arg\max_{s_t} \exp(-J_t), \text{ where } J_t = -\log p(z_t|\theta_t) + \frac{1}{2}(s_t - \overline{\mu}_t)^T \overline{\Sigma}_t^{-1}(s_t - \overline{\mu}_t) \tag{5.29}$$

$$\Rightarrow \qquad 0 = \frac{\partial J_t}{\partial s_t}\Big|_{s_t=\mu_t} = \frac{\partial v_t(z_t,\theta_t)}{\partial \theta_t}\frac{\partial \theta_t}{\partial s_t} + \overline{\Sigma}_t^{-1}(\mu_t - \overline{\mu}_t). \tag{5.30}$$

Taking the derivative of $\theta_t = W(s_t)$ about the prior mean $\overline{\mu}_t$, we let

$$Y_t = \frac{\partial W(s_t)}{\partial s_t}\Big|_{s_t=\overline{\mu}_t}. \tag{5.31}$$

Similarly, performing a Taylor expansion on $\frac{\partial v_t(z_t|\theta_t)}{\partial \theta_t}$ about $\overline{\theta}_t = W(\overline{\mu}_t)$,

$$\frac{\partial v_t(z_t|\theta_t)}{\partial \theta_t} = \frac{\partial v_t(z_t|\theta_t)}{\partial \theta_t}\Big|_{\theta_t=\overline{\theta}_t} + \frac{\partial^2 v_t(z_t|\theta_t)}{\partial \theta_t \partial \theta_t^T}\Big|_{\theta_t=\overline{\theta}_t}(\theta_t - \overline{\theta}_t) \tag{5.32}$$

$$\frac{\partial v_t(z_t|\theta_t)}{\partial \theta_t} = \dot{v}_t + \ddot{v}_t(\theta_t - \overline{\theta}_t) \tag{5.33}$$

where

$$\dot{v}_t = \frac{\partial}{\partial \theta_t}(-z_t^T\theta_t + \beta_t(\theta_t) - \kappa_t(z_t))\Big|_{\theta_t=\overline{\theta}_t}, \qquad \text{(Eqn. 5.23)} \tag{5.34}$$

$$= \frac{\partial \beta_t(\theta_t)}{\partial \theta_t}\Big|_{\theta_t=\overline{\theta}_t} - z_t \tag{5.35}$$

$$\dot{v}_t = \dot{\beta}_t - z_t \tag{5.36}$$

and

$$\ddot{v}_t = \frac{\partial^2 \beta_t(z_t|\theta_t)}{\partial \theta_t \partial \theta_t^T}\Big|_{\theta_t=\overline{\theta}_t}(\theta_t - \overline{\theta}_t). \tag{5.37}$$

$$\ddot{v}_t = \ddot{\beta}_t \tag{5.38}$$

Plugging Equations 5.36 and 5.38 into Equation 5.33, and then into Equation 5.30,

$$Y_t^T(\dot{\beta}_t - z_t + \ddot{\beta}_t(\theta_t - \overline{\theta}_t)) = -\overline{\Sigma}_t^{-1}(\mu_t - \overline{\mu}_t) \tag{5.39}$$

$$Y_t^T \ddot{\beta}_t(\ddot{\beta}_t^{-1}(\dot{\beta}_t - z_t) - \overline{\theta}_t + \theta_t) = -\overline{\Sigma}_t^{-1}(\mu_t - \overline{\mu}_t) \tag{5.40}$$

$$Y_t^T \ddot{\beta}_t((\overline{\theta}_t - \ddot{\beta}_t^{-1}(\dot{\beta}_t - z_t)) - \theta_t) = \overline{\Sigma}_t^{-1}(\mu_t - \overline{\mu}_t) \tag{5.41}$$

$$Y_t^T \ddot{\beta}_t(\tilde{z}_t - W(s_t)) = \overline{\Sigma}_t^{-1}(\mu_t - \overline{\mu}_t), \tag{5.42}$$

where $\tilde{z}_t = (\overline{\theta}_t - \ddot{\beta}_t^{-1}(\dot{\beta}_t - z_t))$ is the projection of the observation onto the parameter space of the exponential family distribution, and is independent of s_t. In Equation 5.42 we substituted θ_t using Equation 5.22.

Mean update

Using Equation 5.42 and substituting μ_t for s_t,

$$\overline{\Sigma}_t^{-1}(\mu_t - \overline{\mu}_t) = Y_t^T \ddot{\beta}_t(\tilde{z}_t - W(\mu_t)) \tag{5.43}$$

$$= Y_t^T \ddot{\beta}_t(\tilde{z}_t - W(\mu_t)) + W(\overline{\mu}_t) - W(\overline{\mu}_t) \tag{5.44}$$

$$= Y_t^T \ddot{\beta}_t(\tilde{z}_t - W(\overline{\mu}_t)) - Y_t^T \ddot{\beta}_t(W(\mu_t) - W(\overline{\mu}_t)). \tag{5.45}$$

Linearizing $W(s_t)$ about $\overline{\mu}_t$,

$$W(s_t) = W(\overline{\mu}_t) + W'(s_t)_{s_t=\overline{\mu}_t}(s_t - \overline{\mu}_t) \tag{5.46}$$

$$= W(\overline{\mu}_t) + Y_t\,(\mu_t - \overline{\mu}_t) \tag{5.47}$$

$$\Rightarrow \quad \overline{\Sigma}_t^{-1}(\mu_t - \overline{\mu}_t) = Y_t^T \ddot{\beta}_t(\tilde{z}_t - W(\overline{\mu}_t)) - Y_t^T \ddot{\beta}_t Y_t\,(\mu_t - \overline{\mu}_t) \tag{5.48}$$

$$Y_t^T \ddot{\beta}_t(\tilde{z}_t - W(\overline{\mu}_t)) = (\overline{\Sigma}_t^{-1} + Y_t^T \ddot{\beta}_t Y_t\,)(\mu_t - \overline{\mu}_t) \tag{5.49}$$

$$= \Sigma_t^{-1}(\mu_t - \overline{\mu}_t) \tag{5.50}$$

$$\Rightarrow \quad \mu_t - \overline{\mu}_t = \Sigma_t Y_t^T \ddot{\beta}_t(\tilde{z}_t - W(\overline{\mu}_t)), \tag{5.51}$$

where $\Sigma_t Y_t^T \ddot{\beta}_t = \tilde{K}_t$ is the Kalman gain for non-Gaussian exponential family distributions. Via a standard transformation, the Kalman gain can be written in terms of covariances other than Σ_t,

$$\tilde{K}_t = \overline{\Sigma}_t Y_t^T (Y_t \overline{\Sigma}_t Y_t^T + \ddot{\beta}_t^{-1})^{-1} \tag{5.52}$$

$$\text{and} \qquad \mu_t = \overline{\mu}_t + \tilde{K}_t(\tilde{z}_t - W(\overline{\mu}_t)). \tag{5.53}$$

Covariance update

Given a Gaussian posterior belief, $\frac{\partial^2 J}{\partial s_t^2}$ is the inverse of the covariance of the agent's belief

$$\Sigma_t^{-1} = \frac{\partial^2 J}{\partial s_t^2} \tag{5.54}$$

$$= \frac{\partial}{\partial x}(\overline{\Sigma}_t^{-1}(s_t - \overline{\mu}_t) - Y_t^T \ddot{\beta}_t(\tilde{z}_t - W(s_t))) \tag{5.55}$$

$$= \overline{\Sigma}_t^{-1} + Y_t^T \ddot{\beta}_t Y_t \tag{5.56}$$

$$\Rightarrow \Sigma_t = (\overline{\Sigma}_t^{-1} + Y_t^T \ddot{\beta}_t Y_t)^{-1}. \tag{5.57}$$

5.1.5 Approximate computation of posterior belief distribution

Given an action-observation sequence, the posterior mean of the agent's belief in the efKF can then be updated according to

$$\overline{\mu}_t = A_t\mu_{t-1} + B_t a_t \qquad \mu_t = \overline{\mu}_t + \tilde{K}_t(\tilde{z}_t - W(\overline{\mu}_t)), \tag{5.58}$$

$$\overline{\Sigma}_t = A_t\Sigma_{t-1}A_t^T + P_t \qquad \Sigma_t = (\overline{\Sigma}_t^{-1} + Y_t^T\ddot{\beta}_t Y_t)^{-1}, \tag{5.59}$$

where $\tilde{K}_t = \overline{\Sigma}_t Y_t(Y_t\overline{\Sigma}_t Y_t^T + \ddot{\beta}_t^{-1})^{-1}$ is the efKF Kalman gain, and $\tilde{z}_t = \overline{\theta}_t - \ddot{\beta}_t^{-1}\cdot(\dot{\beta}_t - z_t)$ is the projection of the observation onto the parameter space of the exponential family observation model. $Y_t = \frac{\partial\theta_t}{\partial s_t}\big|_{s_t=\overline{\mu}_t}$ is the gradient of the exponential family distribution's canonical parameter, linearized about $\overline{\mu}_t$.

The belief covariance comes from Equation 5.59, and Equation 5.18 can be modified based on the efKF equations:

$$\mu_{t+T} \sim \mathcal{N}(f(\mu_{t-1}, A_{t:t+T}, B_{t:t+T}, a_{t:t+T}), \sum_{i=t}^{t+T}\overline{\Sigma}_i Y_i^T\tilde{K}_i^T) \tag{5.60}$$

However, it is worth noting that in contrast to Equations 5.4 and 5.18 , which are exact and completely independent of the observations, Equations 5.59 and 5.60 are no longer independent of the observations obtained, because the observation model parameters are linearized about the prior mean $\overline{\mu}_t$. Hence while the parameters are independent of the observation that will be obtained for a macro-action sequence of length 1, for a longer macro-action, the observation model parameters depend on the prior observations obtained. Nevertheless, we approximate this update by linearizing about the mean of the prior mean distribution m_t at each step along the action sequence, rather than the true prior belief mean μ_t. We still obtain good experimental results using this approximation.

This approximate method for computing the posterior distribution over beliefs can be used as a substitute for exactly calculating the posterior distribution over beliefs in either the PBD or PBDE algorithm.

5.2 Analysis

Here we provide a formal analysis of the accuracy and computational complexity of our macro-action forward search algorithms. We start by analyzing the value computed by the PBDE algorithm, and then consider the value computed by the PBD algorithm. We then analyze the computational complexity of these algorithms. For comparison, we briefly consider some alternate representations and approaches and analyze their respective accuracy and computational cost.

5.2.1 Performance

Ideally we would like to bound the performance of our macro-action forward-search algorithms relative to optimal performance. However, our algorithms can only explore a subset of the full policy space, due to their use of macro-actions. Though the tradeoff of more depth for lesser breadth leads to significant computational advantages, as we shall analyze shortly, the optimal policy may lie outside of the policy subspace covered by the macro-actions. Therefore we cannot guarantee that our algorithm will compute the optimal policy. However, we can guarantee that the finite horizon belief-action values computed at the root belief by the PBDE algorithm are exact: that is, these values represent the true expected value that will be received when executing the PBDE algorithm. Note that this is not always the case. Many planning under uncertainty algorithms that use approximations, where the values calculated by the algorithm may or may not match the true expected reward from executing that algorithm's policy, and do not provide guarantees on whether the calculated values are upper or lower bounds on the value of the optimal policy.[2] The fact that the PBDE algorithm computes a value which matches the true value of the PBDE policy allows us to guarantee that the computed PBDE value is a lower bound to the optimal policy (Theorem 1) as we prove in the next subsection. This guarantee may also have practical significance: for example, it means that the values returned by PBDE could potentially be used at the leaves of a forward search algorithm to provide a provably lower bound on the future value of a leaf node.

Performance analysis of open-loop planning using macro-actions (PBDE)

First note that the value of the action at the start of a L-length macro action, where L is greater than one, is necessarily a lower bound of the optimal L-step value of that action. This is simply because the macro-action pre-determines a set of subsequent actions, whereas the optimal value involves taking the maximum over the reward of subsequent action choices.

Recall that in Chapter 3, we have derived the known result (Puterman, 1994) that the error between the optimal infinite-horizon expected belief-action value and the optimal H-horizon expected belief-action value is bounded as a function of the horizon H. If we know that the optimal H-horizon macro-action policy is a lower bound, then it remains to be shown that PBDE exactly computes the value of the PBDE policy, proving that the value computed by PBDE is a lower bound on the optimal, infinite-horizon policy.

Theorem 5.2.1 *Assume that the dynamics and observation models are linear Gaussian, and the reward model is either a weighted sum of Gaussians or a polynomial function of the state space. Then the H-horizon belief-action value of the best root action a^{PBDE} computed by the PBDE algorithm is a lower bound on the infinite-horizon optimal value of a^{PBDE}.*

[2] In practice, any policy which is non-optimal will have an expected reward during execution which is lower than the expected reward of the optimal policy. However, the policy value *calculated* by an algorithm may be larger than the optimal value, even if the *executed* policy's expected rewards are lower than the optimal value.

Proof We will first show that we can compute the PBDE H-horizon belief-action values exactly, and then combine this result with Lemma 1 to prove the theorem.

First, let the primitive action a^{PBDE} be the first action in the L-length macro-action $\tilde{a}^{PBDE}$. The value of a^{PBDE} is the expected L-horizon value of the macro-action $\tilde{a}^{PBD}$ plus the expected future value after the macro-action. Let $\tilde{Q}_H$ be used to represent values when macro-actions are used, and the subscript will be used to represent the value horizon either in terms of primitive actions (H) or in terms of the number of macro-actions if a tilde is used ($\tilde{H}$). More specifically, $\tilde{Q}_{\tilde{H}}(b_{dist}, a)$ is the value of taking $\tilde{H}$ macro-actions starting from a distribution of beliefs b_{dist} and primitive action a, which is the first action in macro-action $\tilde{a}$. Note that b_{dist} could be a delta function over a single belief b. $\tilde{Q}_L(b_{dist}, \tilde{a})$ is the value of the single macro-action $\tilde{a}$ taken from the distribution of beliefs b_{dist}. Then

$$\tilde{Q}_{\tilde{H}}(b_{dist}, a^{PBDE}) = \tilde{Q}_{\tilde{H}}(b_{dist}, \tilde{a}^{PBDE}) \tag{5.61}$$

$$= \tilde{Q}_L(b_{dist}, \tilde{a}^{PBDE}) + \max_{\tilde{a} \in \tilde{A}} \tilde{Q}_{\tilde{H}-1}(b_{dist}^{\tilde{a}^{PBDE}}, \tilde{a}), \tag{5.62}$$

where $b_{dist}^{\tilde{a}^{PBDE}}$ is the distribution over beliefs after macro-action $\tilde{a}^{PBDE}$. The first equality holds trivially as the value of the action at the start of a macro-action under PBDE must be the same as the value of the macro-action itself. The second equality holds because the value of $\tilde{a}$ under the PBDE algorithm is the expected reward of following $\tilde{a}$ plus the value of following the best next macro-action, given the distribution over beliefs after taking $\tilde{a}$. This definition is recursive, and so the PBDE value of a macro-action $\tilde{a}$ is simply the value of the best $\tilde{H}$ sequence of macro-actions that lead to the highest expected reward given the initial belief distribution b_{dist}. Note that there is no sampling performed (of beliefs or observations) and so this is essentially an $L\tilde{H}$-length open loop policy. The expected value computation in Equation 5.62 is therefore identical to finding the value of a single best macro-action $\tilde{a}^H$ that starts with a^{PBDE} and consists of $\tilde{H}$, L-length macro-actions in a row (for a total of $H = L\tilde{H}$ primitive actions):

$$\tilde{Q}_{\tilde{H}}(b_{dist}, a^{PBDE}) = \tilde{Q}_L(b_{dist}, \tilde{a}^{PBDE}) + \tilde{Q}_{\tilde{H}-1}(b_{dist}^{\tilde{a}^{PBDE}}, \tilde{a}^2) \tag{5.63}$$

where $\tilde{a}^2$ is the second macro-action in the best $\tilde{H}$ sequence of macro-actions starting from b_{dist}. Alternately, we could view it as a single macro-action $\tilde{a}^H$ consisting of $H = l\tilde{H}$ primitive actions and calculate the value of the first primitive action in this sequence, followed by the expected reward of the remaining $H-1$ primitive actions:

$$\tilde{Q}_{\tilde{H}}(b_{dist}, a^{PBDE}) = R_B(b_{dist}, a^{PBDE}) + \sum_{z \in Z} p(z|b, a^{PBDE}) \tilde{Q}_{H-1}(b^{a^{PBDE}, z}, a^{H,2}) \tag{5.64}$$

where $a^{H,2}$ is the second primitive action in the best long macro-action $\tilde{a}^H$.

To prove this is a lower bound to the optimal value of a^{PBDE} we first need to show that PBDE can

exactly compute the expected reward for any given primitive action. Since the dynamics and observation models are linear Gaussian, then the well-known Kalman filter can be used to exactly perform belief updating. Similarly, the posterior distribution over beliefs b_{dist} after each action can be computed exactly. Calculating the expected reward of primitive action a for a distribution of beliefs b_{dist} requires integrating over this posterior distribution:

$$R_B(b_{dist}, a) = \int_s \int_b R_S(s,a) b(s) b^a_{dist}(b) dbds. \tag{5.65}$$

Recall from the prior section that the posterior distribution over beliefs can be factored into a Gaussian distribution over the belief means μ (Equation 5.18), and a Dirac delta distribution over the belief covariances Σ (since all beliefs will have identical covariances):

$$b_{dist}(\mu, \Sigma) = \mathcal{N}(\mu | m_a, \Sigma^\mu_a) \delta(\Sigma, \Sigma_a) \tag{5.66}$$

where m_a is the mean of the belief means after primitive action a, Σ^μ_a is the covariance of the belief means after primitive action a, and Σ_a is the covariance of a belief state after primitive action a.

As the belief state itself is a Gaussian,

$$b(s) = \mathcal{N}(s | \mu, \Sigma), \tag{5.67}$$

we can re-express the reward as

$$R_B(b_{dist}, a) = \int_s \int_{\mu,\Sigma} R_S(s,a) \mathcal{N}(s|\mu,\Sigma) \mathcal{N}(\mu | m_a, \Sigma^\mu_a) \delta(\Sigma, \Sigma_a) ds d\mu d\Sigma \tag{5.68}$$

$$= \int_s \int_\mu R_S(s,a) \mathcal{N}(s|\mu,\Sigma_a) \mathcal{N}(\mu | m_a, \Sigma^\mu_a) d\mu ds, \tag{5.69}$$

where the second line follows due to the Dirac delta distribution on the belief covariances. Expanding out the formula for $\mathcal{N}(s|\mu,\Sigma)$ we see it is identical to the formula for $\mathcal{N}(\mu|s,\Sigma)$:

$$\mathcal{N}(s|\mu,\Sigma) = \frac{1}{\sqrt{2\pi}|\Sigma|^{N_d/2}} exp(-\frac{1}{2}(s-\mu)\Sigma^{-1}(s-\mu)^T) \tag{5.70}$$

$$= \frac{1}{\sqrt{2\pi}|\Sigma|^{N_d/2}} exp(-\frac{1}{2}(\mu-s)\Sigma^{-1}(\mu-s)^T) \tag{5.71}$$

$$= \mathcal{N}(\mu|s,\Sigma). \tag{5.72}$$

Therefore, we can substitute the equivalent expression to yield

$$R_B(b_{dist}, a) = \int_s \int_\mu R_S(s,a) \mathcal{N}(\mu|s,\Sigma_a) \mathcal{N}(\mu | m_a, \Sigma^\mu_a) d\mu ds. \tag{5.73}$$

Completing the square in the exponent, we re-express the product of the above two Gaussians as

$$R_B(b_{dist}, a) = \int_s \int_\mu R_S(s,a)\mathcal{N}(s|m_a, \Sigma_a + \Sigma_a^\mu)\mathcal{N}(\mu|c, C)d\mu ds. \quad (5.74)$$

where $C = (\Sigma_a^{-1} + (\Sigma_a^\mu)^{-1})^{-1}$ and $c = C(m_a(\Sigma_a^\mu)^{-1} + \mu\Sigma_a^{-1})$. We then integrate over μ to get

$$R_B(b_{dist}, a) = \int_s R_S(s,a)\mathcal{N}(s|m_a, \Sigma_a + \Sigma_a^\mu)ds. \quad (5.75)$$

If the reward model itself is a weighted sum of N_r Gaussians,

$$R_S(s,a) = \sum_{j=1}^{N_r} w_j \mathcal{N}(s|\zeta_j, \Upsilon_j), \quad (5.76)$$

then the integral in Equation 5.75 can be evaluated in closed form as

$$R_B(b_{dist}, a) = \int_s \sum_{j=1}^{N_r} w_j \mathcal{N}(s|\zeta_j, \Upsilon_j)\mathcal{N}(s|\mu_a, \Sigma_a + \Sigma_a^\mu)ds \quad (5.77)$$

$$= \sum_{j=1}^{N_r} w_j \mathcal{N}(\zeta_j|\mu_i, \Upsilon_j + \Sigma_a + \Sigma_a^\mu) \int_s \mathcal{N}(s|c_1, C_1), \quad (5.78)$$

where we have again completed the square in the exponent, and defined new constants $C_1 = (\Upsilon_j^{-1} + (\Sigma_a + \Sigma_a^\mu)^{-1})^{-1}$ and $c_1 = C_1(\zeta_j\Upsilon_j^{-1} + \mu_i(\Sigma_a + \Sigma_a^\mu)^{-1})$. Integrating we obtain an analytic expression for the expected reward of a primitive action under a distribution of beliefs:

$$R_B(b_{dist}, a) = \sum_{j=1}^{N_r} w_j \mathcal{N}(\zeta_j|\mu_i, \Upsilon_j + \Sigma_a + \Sigma_a^\mu). \quad (5.79)$$

A similar closed-form expression is available if the reward model is a polynomial functions of the state,

$$R_S(s,a) = \sum_{j=1}^{N_r} w_j s^j, \quad (5.80)$$

instead of a weighted sum of Gaussians. Substituting Equation 5.80 into Equation 5.75 yields

$$R_B(b_{dist}, a) = \int_s \sum_{j=1}^{N_r} w_j s^j \mathcal{N}(s|\mu_a, \Sigma_a^\mu + \Sigma_a)ds$$

$$= \sum_{j=1}^{N_r} w_j \int_s s^j \mathcal{N}(s|\mu_a, \Sigma_a^\mu + \Sigma_a)ds. \quad (5.81)$$

Therefore, evaluating the expected reward involves calculating the first N_r moments of a Gaussian distribu-

tion. Each of these moments is an analytic expression of the Gaussian mean and covariance.[3] So, for reward models that are either a weighted sum of Gaussians, or which are polynomial functions of the state space, the expected reward of a macro-action can be exactly computed.

Equations 5.79 and 5.81 prove that we can exactly compute the expected reward $R_B(b_{dist}, a)$ for any primitive action, which allows us to exactly evaluate Equation 5.64, the value of a $L\tilde{H}$-length macro-action that starts with action a^{PBDE}. This value is necessarily a lower bound on the optimal value $Q^*_{L\tilde{H}}(b, a^{PBDE})$ which allows any future actions instead of only those in the macro-action. In addition, Lemma 1 guarantees the $L\tilde{H}$-step finite horizon optimal value is a lower bound to the infinite horizon optimal value. Putting these results together, we immediately derive the theorem result. □

Performance analysis of conditional macro-action planning (PBD)

While Theorem 1 provides formal bounds on the values computed by the non-sampled PBD variant (PBDE), PBDE is unlikely to perform well in practice because it selects a sequence of macro-actions in an open loop fashion, without conditioning on observation nodes in the forward search tree. In contrast, the PBD, MAD and MAC algorithms follow a conditional instead of unconditional/open-loop plan by sampling beliefs from the posterior belief distribution, and branching on each sampled belief.

We now turn our attention to the PBD algorithm. The $L\tilde{H}$-horizon value of a root action a^{PBD} coming from macro action $\tilde{a}^{PBD}$ in the standard PBD algorithm is:

$$\tilde{Q}_{\tilde{H}}(b, a^{PBD}) = \tilde{Q}_L(b, a^{PBD}) + \frac{1}{N_s} \sum_{i=1}^{N_s} \max_{\tilde{a}^* \in \tilde{\mathcal{A}}} \tilde{Q}_{\tilde{H}-1}(b_i, \tilde{a}^*) \tag{5.82}$$

where N_s beliefs b_i are sampled from the posterior distribution of beliefs arising after macro-action $\tilde{a}^{PBD}$, and the best next macro-action is chosen for each sampled belief.

In general, if the set of sampled beliefs constitutes an particle representation that is identical to the true posterior distribution of beliefs, then Equation 5.82 is guaranteed to be a lower bound of the belief-action value of a^{PBD} because the expectation over the sampled beliefs will match the true distribution over the potential observation sequences. To make this clearer, let us consider an example where a macro-action $\tilde{a}^{PBD}$ consists of 2 primitive actions, and there are 2 different observations, yielding $2^2 = 4$ unique observation sequences that could be received during macro-action $\tilde{a}^{PBD}$. Further assume that for a particular initial belief, the probability of each observation sequence is 0.25, 0.4, 0.25, and 0.1 respectively. Instead of enumerating the four observation sequences, imagine 100 beliefs were sampled from the posterior distribution of beliefs after $\tilde{a}^{PBD}$. Each sampled belief corresponds implicitly to an observation sequence, and assume that 25 samples (0.25 of the total number of samples) came from the first sequence, 40 from the second, 25 from the third, and 10 from the fourth. In this case the set of sampled beliefs form an empirical distribution that

[3]The Gaussian distribution is completely described by its first two moments; all higher order moments are simply functions of the first two moments.

is identical to the true posterior distribution of beliefs had we enumerated all observation sequences, and weighed each possible posterior belief by the probability of the associated observation sequence. In such cases, the value computed using the sampled belief distribution is a lower bound because the macro-actions restrict the policy class considered.

However, it is unlikely that the sampled belief distribution is identical to the true posterior distribution over beliefs, and it would be hard to know for any finite number of belief samples if this set exactly captured the true posterior distribution over beliefs. As the number of samples N_s goes to infinity, the sampled posterior distribution will becomes arbitrarily close to the true distribution, and Equation 5.82 will become a lower bound on the optimal belief-action value. However, for a finite number of belief samples N_s, Equation 5.82 cannot be guaranteed to be an upper or lower bound on the true action value. This is because even if the value of each sampled belief is guaranteed to be a lower bound, the distribution over sampled beliefs may disproportionately include beliefs that lead to higher rewards.

Therefore, the belief-action values of both the PBD algorithm which samples beliefs from the posterior distribution over beliefs after a macro-action, and the MAC/MAD algorithms that sample observation sequences, are unknown to be upper or lower bounds on the optimal values of the root actions. However, we do expect them to generally perform better than PBDE, since PBD, MAC and MAD all condition, implicitly or explicitly, on the received observations when constructing a policy, unlike PBDE.

Similarly, for a significant class of observation and dynamics models, it will not be possible to exactly update the belief state and maintain a Gaussian representation over the resulting posterior. Instead, as we described in Section 5.1.4, we take the popular approach of approximating the resulting distribution as a Gaussian by linearizing about the resulting reward function. Linearization makes the posterior distribution of beliefs an approximation of the true posterior distribution: we will label this approximation $\hat{b}_{dist}$. The degree of error in the reward estimate depends critically on the interaction of the domain-specific reward formulation, and the encountered belief states:

$$|R_B(\hat{b}_{dist}, a) - R_B(b_{dist}, a)| = \left| \sum_{i=1}^{L} \int_s \int_b R_S(s, a) b(s) (\hat{b}_{dist} - b_{dist}) dbds \right|. \tag{5.83}$$

In general, this error could be significant. However, our experimental results will demonstrate that using sampling and linearizing within a macro-action forward search can still outperform prior approaches where considering further horizons can improve action selection.

In essence, our approach trades the exact solutions of exhaustive forward search procedures for a more restricted policy class that allows us to efficiently search to longer horizons and approximately compute the values of those longer-horizon policies. The potential benefit of this tradeoff is made clearer by reconsidering the bounds of Lemma 1, which provides an upper bound to the error between the optimal infinite-horizon value and the H-step horizon value which is a direct function of the horizon H. Consider the case when the discount factor is 0.95, and the problem domain satisfies the conditions for exact belief updating as

described in Section 5.1.1. If we approximate the infinite-horizon optimal value by only a 3-step ($H = 3$) horizon plan, then an upper bound on the error between the optimal infinite-horizon and the optimal 3-step horizon value evaluates to $15.48R_{max}$ or $0.77V_{max}$ where V_{max} is defined as $R_{max}/(1-\gamma)$. In contrast, if we instead can afford to compute a 20-step horizon plan ($H = 20$), then the upper bound on the error between the infinite-horizon optimal value and finite-horizon optimal value reduces to $0.34V_{max}$, which is less than half the upper bound on the error when using $H = 3$. This simple example illustrates the potential benefit of planning out to longer horizon in terms of the accuracy of the computed value functions relative to the optimal infinite-horizon value. Since our underlying assumption is that better estimates of belief-action values will lead to better policies, computing the value of longer-horizon policies may have significant performance advantages. Even approximately estimating the values of long-horizon policies for models where the posterior belief distribution cannot be calculated exactly can outperform full forward search, which often becomes computationally intractable after a short horizon. This claim will be supported experimentally later in the next chapter.

5.2.2 Computational complexity

One of the central contributions of our work is providing an efficient macro-action forward search algorithm that can scale to long horizons and large problems. We first analyze the computational complexity of searching using an analytic parametric distribution of the result of a macro-action; performance bounds were provided in Theorem 1. We then consider how this computational cost changes when observation sequences are sampled and a discrete representation is used, such as in the MAD algorithm approach.

Complexity of Gaussian belief updating for a length L macro-action

The computation for the posterior distribution over beliefs resulting from a macro-action was presented in Equation 5.60, and consists of a set of matrix multiplications and inversions. Matrix multiplication is an $\mathcal{O}(D^2)$ computation, where D is the state space dimension. Matrix inversion can be done in $\mathcal{O}(D^3)$ time. Therefore the computational cost of performing a single update of the posterior over belief states is an $\mathcal{O}(D^3)$ operation. This update must be performed for each primitive action in a length-L macro-action $\tilde{a}$, resulting in a computational cost of

$$\mathcal{O}(LD^3) \tag{5.84}$$

for a single macro-action.

Complexity of analytically computing the expected reward of a macro-action

The second component of the computational cost comes when we evaluate the expected reward of a macro-action. If the reward is a weighted sum of N_r Gaussians, as specified by Equation 5.76, this operation involves

evaluating the value of $N_r L$ Gaussians at particular fixed points. Evaluating a D-dimensional Gaussian at a single point is an $\mathcal{O}(D^3)$ operation, due to the inverse covariance that must be computed. The cost for performing this operation $N_r L$ times is simply $\mathcal{O}(N_r L D^3)$. Therefore the total cost for evaluating the expected reward of a macro-action when the belief can be represented exactly as a Gaussian and the reward model is a weighted sum of N_r Gaussians is:

$$\mathcal{O}(LD^3(N_r + 1)). \tag{5.85}$$

If instead the reward model is a N_r-th degree polynomial function of the state, then the expected reward calculation consists of the cost of calculating the N_r-moments of a D-dimensional Gaussian distribution (Equation 5.81). Assume without loss of generality that we are computing the N_r-th central moment of a D-dimensional Gaussian: a non-central moment can always be converted into a central moment by adding and subtracting a mean term. Let the N_r-th central moment denote moments of the form $E[(s_1 - E[s_1])^2(s_2 - E[s_2]) \ldots (s_D - E[s_D])]$ or $E[(s_2 - E[s_2])^{N_r}]$, and σ_{ij} denote the ij-th entry of the covariance matrix. From Triantafyllopoulos (2003) we know that if N_r is odd, the central N_r-th moments are zero, and if N_r is even ($N_r = 2k$) any N_r-th central moments can be decomposed into a sum over products of k covariance terms. For example, for a four-dimensional Gaussian, one of the fourth central moments ($k = 2, 4 = 2k$) is

$$E[(s_1 - \mu_1)(s_2 - \mu_2)(s_3 - \mu_3)(s_4 - \mu_4)] = \sigma_{12}\sigma_{34} + \sigma_{14}\sigma_{23} + \sigma_{13}\sigma_{24} = \sum_{1,2,3,4} \sigma_{ij}\sigma_{kl} \tag{5.86}$$

where the sum is taken over all permutations of product pairs (in this case, $12/34, 14/23, 13/24$). For any $2k$-th central moment,

$$E[(s_{i_1} - E[s_{i_1}])(s_{j_1} - E[s_{j_1}]) \ldots (s_{i_k} - E[s_{i_k}])(s_{j_k} - E[s_{j_k}])] = \sum \sigma_{i_1 j_1}\sigma_{i_2 j_2} \ldots \sigma_{i_k j_k} \tag{5.87}$$

where the sum is again taken over all permutations of product pairs. This sum yields $(N_r - 1)!/(2^{k-1}(k-1)!)$ terms which consist of covariance elements to the power of at most k. For a particular central moment, this cost is independent of the dimension of the state space. Therefore the cost is dominated by the number of terms, which grows at slightly less than $\mathcal{O}(N_r!)$. There will also be an additional cost if the original polynomial was not a central moment calculation, which will involve at most N_r D-dimensional matrix multiplications, yielding a cost of $\mathcal{O}(N_r D^2)$. In summary, the cost of computing the expected reward when the reward is a polynomial function will be

$$\mathcal{O}(L(D^3 + N_r! + N_r D^2)). \tag{5.88}$$

Complexity of open Loop macro-action planning (PBDE)

Performing forward search requires both updating the posterior distribution of beliefs, and computing the expected reward, recursively out to a fixed macro-action depth horizon $\tilde{H}$. Consider first the computational complexity when we can only select a single macro-action ($\tilde{H} = 1$). In this case, we only need to compute the expected posterior distribution and reward of each of $|\tilde{\mathcal{A}}|$ macro-actions. From Equations 5.84, 5.85, and 5.88, the computational complexity for calculating the posterior distribution over beliefs and expected reward for a single action is $\mathcal{O}(LD^3)$, so for $|\tilde{\mathcal{A}}|$ macro actions the total cost is simply $\mathcal{O}(|\tilde{\mathcal{A}}|LD^3)$. More generally, for a horizon of $\tilde{H}$ macro-actions, PBDE planning consists of considering all $|\tilde{\mathcal{A}}|^{\tilde{H}}$ potential sequences of macro-actions, with an associated computational complexity of

$$\mathcal{O}(|\tilde{\mathcal{A}}|^{\tilde{H}} LD^3 N_r). \tag{5.89}$$

Therefore the cost scales as a cubic function of the number of state dimensions, and exponentially with the macro-action search depth, where the base is the number of macro-actions. This cost is independent of the number of observations, and scales linearly with the number of primitive actions (L) per macro-action. For long-horizon searches, if the number of macro-actions is small, this will be a significant savings over standard full primitive-action forward search approaches.

Complexity of conditional macro-action planning (PBD)

Because PBDE is an open-loop planner, there is no way to choose different macro-actions given particular observation sequences. PBD does provide this capability by sampling beliefs from the posterior distribution over beliefs at each depth, which is equivalent to sampling observation sequences, and conditioning the subsequent macro-actions of the plan on each sampled belief. We next consider the additional computational cost incurred in performing belief sampling.

Sampling beliefs from the posterior distribution over beliefs requires sampling from a multivariate Gaussian over the distribution of belief means, which we accomplish by computing the Cholesky decomposition of the covariance matrix, $\Sigma = AA^T$, an $\mathcal{O}(D^3)$ operation. Each belief mean is generated by first constructing a D-dimensional vector q, consisting of D independent samples from a standard (scalar) normal distribution. A sample from the desired multivariate Gaussian $\mathcal{N}(s|\mu, \Sigma)$ is simply $\mu + Aq$. Sampling N_s times involves the one-time cost of computing the Cholesky decomposition plus the matrix-vector multiplication for each sample, yielding a cost of

$$\mathcal{O}(D^3 + N_s D^2). \tag{5.90}$$

This procedure is performed at every branch point in the forward search tree (in other words, at all macro-action nodes except those at the tree leaves). For concreteness, consider a horizon of two macro-actions

($\tilde{H} = 2$). After expanding out each of the $|\tilde{\mathcal{A}}|$ macro-actions, we will sample N_s beliefs. From each resulting belief state, we will again expand each of the $|\tilde{\mathcal{A}}|$ macro-actions: refer back to Figure 5-3b for an illustration. The computational complexity is now the sum of the cost at horizon one and two:

$$\mathcal{O}(|\tilde{\mathcal{A}}|(LD^3N_r + N_sD^2 + D^3) + |\tilde{\mathcal{A}}|^2N_sLD^3N_r) = \mathcal{O}(|\tilde{\mathcal{A}}|(N_sD^2 + D^3) + |\tilde{\mathcal{A}}|^2N_sLD^3C), \quad (5.91)$$

where the second expression is derived by considering only the higher order terms. In general, the computational complexity of selecting an action using PBD when considering a future horizon of $\tilde{H}$ macro-actions is

$$\mathcal{O}(|\tilde{\mathcal{A}}|^{\tilde{H}-1}N_s^{\tilde{H}-2}(N_sD^2 + D^3) + |\tilde{\mathcal{A}}|^{\tilde{H}}N_s^{\tilde{H}-1}LD^3C). \quad (5.92)$$

Complexity of PBD with arbitrary reward models

For arbitrary reward models it will not be possible to analytically compute the expected reward. Instead the expected reward for each primitive action a within the macro-action $\tilde{a}$ can be approximated by sampling D-dimensional states and estimating the expected reward by averaging the reward of each sampled state.

Note that if the rewards are bounded, for a given ϵ and δ, sampling a sufficient number of samples $N_s = f(\epsilon, \delta)$, guarantees the estimate of the expected reward of a primitive action is is ϵ-close to the true expected value, with probability at least $1 - \delta$. The proof of this is a simple application of Hoeffding's inequality (Hoeffding, 1963). If N_s is set such that the estimated reward of each primitive action is $\frac{\epsilon}{L}$ close to the true expected primitive action reward with probability at least $1 - \frac{\delta}{\epsilon}$, then the triangle inequality and union bound guarantee that the expected reward of the entire length-L macro-action is ϵ-close to the true expected reward for the macro-action with probability at least $1 - \delta$.

The cost of sampling N_s states involves sampling N_s times from a multivariate Gaussian, which is an $\mathcal{O}(D^3 + D^2)$ operation (from Equation 5.90). Assuming that calculating the reward for each sample takes time linear in the state dimension, then sampling rewards adds an additional

$$\mathcal{O}(D^3 + ND^2D) = \mathcal{O}(D^3(N + 1)) \quad (5.93)$$

cost to each primitive action within a macro-action, yielding a total complexity of PBD planning with reward sampling of:

$$\mathcal{O}(|\tilde{\mathcal{A}}|^{\tilde{H}}N_s^{\tilde{H}}LD^3 + |\tilde{\mathcal{A}}|^{\tilde{H}}N_s^{\tilde{H}}LD^2). \quad (5.94)$$

Complexity of macro-actions with observation sequence sampling & parametric beliefs

An alternative to analytically computing the posterior distribution over beliefs after a macro-action is to use a particle approximation of this distribution by sampling observation sequences directly. If observation

sequences are sampled, then even for a single macro action, a belief update must be performed for each of the N_z sampled L-length observation trajectories, yielding a computational cost of $\mathcal{O}(|\tilde{\mathcal{A}}|N_z LD^3)$ for a single macro-action, assuming that the expected reward can be calculated analytically. At longer horizons ($H > 1$) the number of observation trajectories will scale exponentially with the horizon. The computational complexity of sampling observation sequences when the expected reward can be analytically evaluated is

$$\mathcal{O}(|\tilde{\mathcal{A}}|^{\tilde{H}}(N_z)^{\tilde{H}}LD^3). \tag{5.95}$$

If instead the reward must be numerically estimated, then an additional N_s samples are required to evaluate the expected reward at each belief, which, from Equation 5.90, increases the computational complexity to

$$\mathcal{O}(|\tilde{\mathcal{A}}|^{\tilde{H}}(N_z)^{\tilde{H}}LD^3 + |\tilde{\mathcal{A}}|^{\tilde{H}}(N_z)^{\tilde{H}}N_s LD^2). \tag{5.96}$$

It is useful to compare these results to the standard PBD algorithm which uses an analytic expression over the posterior distribution of beliefs. The complexity will depend on the relative number of observation sequences sampled versus the number of beliefs sampled. However, an interesting situation to examine is when the number of observation sequences sampled is equal to the number of beliefs sampled in the standard PBD case, which is identical to the number of states sampled to evaluate numerically the expected reward. Let us call this number N_s. Assuming that the number of observation sequences sampled is the same as the number of beliefs sampled in the PBD algorithms is particularly intuitive, as both algorithms try to instantiate a particle-like approximation of the resulting posterior distribution of beliefs after a macro-action. In this situation the computational complexity of evaluating observation sequences is

$$\mathcal{O}(|\tilde{\mathcal{A}}|^{\tilde{H}}N_s^{\tilde{H}}LD^3) \tag{5.97}$$

which is approximately N_s larger than the computational complexity of the PBD algorithm[4]

$$\mathcal{O}(|\tilde{\mathcal{A}}|^{\tilde{H}}N_s^{\tilde{H}-1}LD^3). \tag{5.98}$$

This relationship is also true when the reward must be evaluated numerically.

Complexity of macro-actions with observation sequence sampling & discrete-state spaces

For convenience, we reproduce the complexity analysis, presented in Chapter 3, of the generic macro-action algorithms that compute neither the distribution of posterior beliefs nor the expected rewards analytically, for both factored and non-factored state spaces.

Recall that for problem domains with a factored discrete-state representation, we assume that each of the

[4]The exact relation involves considering the lower order terms of the computational complexity.

Algorithm	Eqn.	Computational Complexity
PBDE	5.89	$\mathcal{O}(\lvert\tilde{\mathcal{A}}\rvert^H LD^3 N_r)$
PBD, analytic expected reward	5.92	$\mathcal{O}(\lvert\tilde{\mathcal{A}}\rvert^H N_s^{H-1} LD^3 N_r)$
PBD, numeric expected reward	5.94	$\mathcal{O}(\lvert\tilde{\mathcal{A}}\rvert^H N_s^H LD^3 + \lvert\tilde{\mathcal{A}}\rvert^H N_s^H LD^2)$
MAC, analytic reward	5.95	$\mathcal{O}(\lvert\tilde{\mathcal{A}}\rvert^H N_s^H LD^3 N_r)$
MAC, numeric reward	5.96	$\mathcal{O}(\lvert\tilde{\mathcal{A}}\rvert^H N_s^H LD^3 + \lvert\tilde{\mathcal{A}}\rvert^H N_s^{H+1} LD^2))$
MAD with factored state	5.99	$\mathcal{O}(\lvert\tilde{\mathcal{A}}\rvert^H N_s^H Lg^2 D)$
MAD with non-factored states	5.100	$\mathcal{O}(\lvert\tilde{\mathcal{A}}\rvert^H N_s^H Lg^{2D})$

Table 5.1: Computational complexity of selecting an action using PBD algorithm and closely related alternatives. D is the number of state dimensions, H is the forward search horizon (as measured in macro-actions), and g is the number of discrete states per state dimension. To make the comparison between the approaches clearer, we have expressed the complexity as a function of N_s which is used interchangeably to represent the number of beliefs sampled per macro-action, the number of states sampled to numerically estimate the expected reward, and the number of observation sequences sampled per macro-action. To simplify the expressions we have only kept the dominant terms.

D state dimensions can be divided into g uniformly-spaced grid cells. Because the belief update of each dimension can be done independently, even though the computational cost of a belief update is a quadratic function of the state space, belief updating has a cost of $\mathcal{O}(g^2 D)$. Similarly, computing the expected reward requires D dot products between g-length vectors, resulting in a computational complexity of

$$\mathcal{O}(|\tilde{\mathcal{A}}|^{H/L} N_z^{H/L} L(g^2 D + gD)). \tag{5.99}$$

when both the distribution of posterior beliefs and expected rewards are not computed analytically.

For a fully non-factored state space, the g^D states would result in a computational complexity of

$$\mathcal{O}(|\tilde{\mathcal{A}}|^{H/L} N_z^{H/L} L(g^{2D} + g^D)), \tag{5.100}$$

since each belief update and expected reward calculation would cost $\mathcal{O}(g^{2D})$ and $\mathcal{O}(g^D)$ respectively.

Table 5.1 summarizes the computational complexity of the different algorithmic approaches. If a large number of samples are necessary to get good experimental results, we expect that the standard PBD algorithm will run significantly faster for the same N_s than sampling observation sequences.

5.3 Conclusion

In this chapter, we have proposed novel algorithms that exploit the inherent structure of different problem domains to perform efficient macro-action forward search. Our PBD algorithm leverages our result that, when the agent's belief can be represented as a Gaussian distribution, we can analytically compute the distribution over the set of posterior beliefs that could result from a single macro-action. The analytic computation removes the need to explicitly enumerate the observation sequences, resulting in significant computational

savings. In discrete environmental models where we cannot analytically capture the resulting distribution of beliefs after a macro-action, our MAD forward search planner that was presented in Chapter 3 generates observation sequences to produce a particle approximation of the resulting distribution of beliefs. The MAD algorithm also easily exploits any factored structure that may exist in the problem to perform efficient belief updates, thereby minimizing computational cost. Finally, we presented theoretical evaluations of the performance and computational cost of our macro-action algorithms.

In the next chapter, we present experimental results when applying the PBD, MAC and MAD algorithms to a variety of problem domains, spanning multiple research communities.

Chapter 6

PBD experiments

In this chapter, we test the algorithms that were presented in Chapter 5 on planning under uncertainty problems from two different research communities: the ISRS problem that we presented in Chapter 4, as well as a target-tracking example that originates from the sensor resource management domain. Despite the different origins and state space representations of these problems, they both require searching over a long horizon in order to generate good policies. Our macro-action forward search approach outperforms existing approaches in both settings. We also demonstrate our algorithm in a target-tracking problem on an actual helicopter platform, underscoring the applicability of our algorithm to real-world domains.

In both problems we compare the PBD algorithm to state-of-the-art approaches from the relevant research community. We also compare our macro-action forward search algorithms which sample observation trajectories (MAC and MAD). The PBDE algorithm was omitted as we found that the algorithm performed poorly, despite the fact that its policy is guaranteed to be a lower bound, due to the reasons discussed in Section 5.1.2. Furthermore, since Chapters 5 and 6 focus on the ability to compute the distribution of posterior beliefs rather than on how the macro-actions are generated, we omit a comparison with the PUMA algorithm from Chapter 4.

6.1 RockSample

In Chapter 3, we proposed the Information RockSample (ISRS) problem, and showed that our semi-conditional planner, the MAD algorithm, obtained better performance relative to other state-of-the-art POMDP solvers. In this section, we revisit the ISRS problem, and compare the performance of the PBD and MAC algorithms with POMDP techniques in the literature. For our PBD and MAC algorithms, we approximate the agent's belief of each rock's value as a Gaussian distribution over the [0,1] state space, and take advantage of the efKF presented in Section 5.1.4 to represent the RockSample problem's Bernoulli observation model.

Recall that r_t is the agent's position at time t, RB_i is the location of the information beacon associated

with rock i, $z_{i,t}$ is a binary observation of the value of rock i at time t, and $s_{i,t}$ is the true value of rock i at time t. Then if we let $d_{i,t} = \| r_t - RB_i \|_2$, then the Bernoulli observation function (Equation 3.1) can be written as follows.

$$p(z_{i,t}|RV_{i,t} = s_{i,t}, r_t, RB_i) \tag{6.1}$$

$$= (0.5 + (s_{i,t} - 0.5)2^{-d_{i,t}/D_0})^{z_{i,t}}(0.5 - (s_{i,t} - 0.5)2^{-d_{i,t}/D_0})^{1-z_{i,t}} \tag{6.2}$$

$$= \exp(z_{i,t} \ln \frac{0.5 + (s_{i,t} - 0.5)2^{-d_{i,t}/D_0}}{0.5 - (s_{i,t} - 0.5)2^{-d_{i,t}/D_0}} + \ln(0.5 - (s_{i,t} - 0.5)2^{-d_{i,t}/D_0})) \tag{6.3}$$

$$= \exp(z_{i,t}\theta_t - \beta_t(\theta_t)). \tag{6.4}$$

We therefore have the parameters of the exponential family observation model

$$\theta_{i,t} = W(s_{i,t}, r_t, RB_i) \tag{6.5}$$

$$= \ln \frac{0.5 + (s_{i,t} - 0.5)2^{-d_{i,t}/D_0}}{0.5 - (s_{i,t} - 0.5)2^{-d_{i,t}/D_0}} \tag{6.6}$$

$$\beta_{i,t} = -\ln(0.5 - (s_{i,t} - 0.5)2^{-d_{i,t}/D_0}) \tag{6.7}$$

$$= \ln(\exp(\theta_{i,t}) + 1). \tag{6.8}$$

We can then derive the derivatives $Y_{i,t}$ and $\ddot{\beta}_{i,t}$

$$Y_t = \frac{\partial W(s_{i,t}, r_t, RB_i)}{\partial s_{i,t}}\bigg|_{s_{i,t}=\hat{m}_{i,t}} \tag{6.9}$$

$$= \frac{\partial}{\partial s_{i,t}} \ln \frac{0.5 + (s_{i,t} - 0.5)2^{-d_{i,t}/D_0}}{0.5 - (s_{i,t} - 0.5)2^{-d_{i,t}/D_0}}\bigg|_{s_{i,t}=\hat{m}_{i,t}} \tag{6.10}$$

$$= \frac{2^{-d_{i,t}/D_0}}{0.5 + (\hat{m}_{i,t} - 0.5)2^{-d_{i,t}/D_0}} \cdot \frac{1}{0.5 - (\hat{m}_{i,t} - 0.5)2^{-d_{i,t}/D_0}} \tag{6.11}$$

where $\hat{s}_{i,t}$ is the mean of the belief used for linearization. Since

$$\Rightarrow \qquad \beta_{i,t} = \ln(\exp(\theta_{i,t}) + 1), \tag{6.12}$$

then

$$\ddot{\beta}_{i,t} = \frac{\partial^2 b_{i,t}}{\partial \theta_{i,t}^2}\bigg|_{\theta_{i,t}=\hat{\theta}_{i,t}} \tag{6.13}$$

$$= \frac{\exp(\hat{\theta}_{i,t})}{\exp(\hat{\theta}_{i,t}) + 1} - \frac{\exp(2\hat{\theta}_{i,t})}{(\exp(\hat{\theta}_{i,t}) + 1)^2}. \tag{6.14}$$

We presented the results in Chapter 3, but repeat them here for convenience. The results in this chapter once again use a domain-specific implementation of the ISRS problem, unlike the experimental results in

	ISRS[8,5]		
	Avg rewards	Online time (s)	Offline time (s)
QMDP	1.11 ± 0.43	0.0001	3.03
HSVI2	6.78 ± 2.46	0.051	1000
SARSOP	8.46 ± 2.46	0.070	25000
RTBSS (d5, s10)	9.78 ± 1.69	17.64	0
MAC (d3,s50)	13.68 ± 1.86	15.39	0
PBD (d3,s50)	14.49 ± 1.73	1.26	0
MAD (d3,s50)	**15.88** ± 1.58	4.81	0
Fully observable	21.37	N.A.	N.A.

Table 6.1: ISRS results. For the forward search algorithms, the numbers in brackets represent the search depth (d) and number of posterior beliefs obtained (s) at the end of each action/macro-action. Online time indicates the average time taken by the planner to return a decision at every timestep. Standard error values are shown.

Chapter 4, which examined a domain-independent implementation of the algorithm. Specifically, our semi-conditional planners are given access to hand-coded macro-actions as detailed in Section 3.4.1. Table 6.1 compares the performance of the different algorithms in the ISRS problem, and includes the performance of all three macro-action algorithms. Each algorithm was tested on 10 sets of hidden rock values, and each scenario was tested 20 times. The forward search algorithms were allowed to search out to a pre-defined depth, which refers to the primitive action depth in the RTBSS algorithm, and the macro-action depth in the macro-action algorithms (MAC, PBD and MAD). In addition, when referring to the pre-defined number of samples that were used to obtain posterior beliefs after every action/macro-action, we again abuse the language here slightly by using samples to refer to observations in the RTBSS algorithm, observation sequences in the MAD and MAC algorithms, and to samples from the posterior belief distribution in the PBD algorithm.

All three macro-action algorithms do significantly better than the other benchmark solvers, demonstrating that approaches that can search deeper can yield better policies in this domain. For the ISRS domain, MAD does particularly well since it exploits the fully-factorizable, discrete-state problem specification, whereas the parametric approaches must approximate the world models during planning. However, the performance difference between the PBD and MAD algorithms is not significant. In addition, for the same planning horizon and the same number of posterior belief samples, the PBD algorithm enjoys the greatest computational efficiency amongst the macro-action forward search algorithms, since it is able to avoid performing belief updates for each observation sequence sample, and instead can directly compute the distribution of posterior beliefs to sample from.

Table 6.2 compares the different rewards obtained by the macro-action algorithms for different macro-action depths, as well as the time taken by the planner to return a decision at every timestep. The sharp performance jump that occurs when the macro-action search depth is increased from 2 to 3 emphasizes the need to search to a longer horizon in the ISRS problem before a good policy can be generated. However, the computational cost of the algorithms also increases exponentially with the macro-action search depth.

Next we examine the relative performance and computational cost of PBD, MAC and MAD, as the num-

	Depth 1, Samples 50		Depth 2, Samples 50		Depth 3, Samples 50		Depth 4, Samples 20	
	Avg rewards	Online time (s)	Avg rewards	Online time (s)	Avg rewards	Online time (s)	Avg rewards	Online time (s)
MAC	4.61	0	**9.63**	0.022	13.68	15.39	15.55	660.50
PBD	4.61	0	7.73	**0.002**	14.49	**1.26**	15.54	**75.06**
MAD	4.61	0	7.51	0.0083	**15.88**	4.81	**18.27**	225.74

Table 6.2: Performance of macro-action algorithms with varying macro-action planning horizon on ISRS. At depth 4, a smaller number of posterior beliefs were sampled for computational reasons

	5 Samples		50 Samples		100 Samples		500 Samples	
	Avg rewards	Online time (s)	Avg rewards	Online time (s)	Avg rewards	Online time (s)	Avg rewards	Online time (s)
MAC	12.76	0.15	13.68	15.39	12.47	58.90	12.94	1732.52
PBD	12.92	**0.035**	14.49	**1.26**	14.56	**4.52**	15.36	**108.64**
MAD	**15.31**	0.056	**15.88**	4.81	**15.57**	20.72	**16.32**	552.64

Table 6.3: Performance of macro-action algorithms in ISRS with different number of samples

	ISRS[100,30]	
	Avg rewards	Online time (s)
MAC(d3,s5)	42.64	310.05
PBD(d3,s5)	43.68	**60.81**
MAD(d3,s5)	**51.70**	101.92
Fully obs.	66.61	N.A.

Table 6.4: Performance of macro-action algorithms on a larger ISRS problem.

ber of samples change (Table 6.3). Sampling allows the macro-action forward search algorithms to consider different conditional plans based on the prior macro-action and potential observation sequence implicitly received. As predicted by our earlier computational complexity analysis, PBD scales best of the three algorithms as the number of samples increases, since it does not have to perform belief updates along each sampled trajectory explicitly. In general, performance improves with more samples, although the improvement was not statistically significant in the ISRS problem. These results suggest that for the ISRS problem, to obtain a good policy, it is more critical to plan to a long horizon, rather than sample many posterior beliefs for each macro-action. However, when a decision-making under uncertainty problem requires a large number of posterior beliefs to be sampled after every macro-action, the PBD algorithm results in a lower time complexity for the same number of samples. Once again, MAD has a slight performance edge due to the approximation of the discrete ISRS problem with continuous variables implicit in PBD, but the difference again is not significant.

The macro-action forward search nature of our algorithm allows us to scale to much larger versions of the RockSample problem, since unlike offline techniques, it is unnecessary to generate a policy that spans the entire belief space. We implemented the macro-action algorithms on an ISRS problem with a 100 by 100 grid and 30 rocks, a problem domain that far exceeds any problem that can be solved by a traditional POMDP solver. Table 6.4 compares the results of the three macro-action algorithms to the fully observable value,

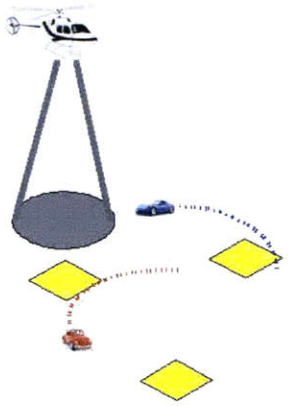


Figure 6-1: TARGETREPORT problem. A helicopter must track multiple targets moving with noisy dynamics. The field-of-view of the agent's sensor (shaded circle) increases with the agent's altitude.

which provides a strict upper bound of the maximum possible reward for the problem.[1] Such large problems also underscore the value of having macro-actions to limit the branching factor of the forward search.

6.2 Target monitoring

We next consider a target-tracking problem related to those studied in the sensor resource management literature (Scott et al., 2009). In contrast to the target-tracking problem that was used for algorithmic comparisons in Chapter 4, this target-tracking problem is extended to continuous state spaces, and the helicopter agent can operate in a full 3D environment. These extensions result in a more realistic an richer problem domain.

In this problem (Figure 6-1), a helicopter agent again has to track multiple targets that are moving independently with noisy dynamics. The helicopter operates in 3D space, while the targets move on the 2D ground plane. The helicopter is equipped with a downward-facing camera for monitoring the environment, and if a target is within the field-of-view of the camera sensor, the agent receives a noisy observation of the location and orientation of the target. We assume for simplicity that the observations of each target are unique, and therefore ignore the data association problem that has been addressed elsewhere in the literature.

The noise associated with the agent's observation of a target depends on the agent's position relative to the target. When the helicopter is close to the ground it can only observe a small region, but can determine the position of objects within that small region to a high level of accuracy. When the helicopter flies at a higher altitude, it can view a wider region of the environment, but its measurements will be less precise. Similarly, the closer the helicopter is to a particular target, the more accurate the helicopter's observation of that target

[1] Standard error values for the larger RockSample problems are not presented due to the small number of trials that were performed. These results are primarily meant to illustrate the applicability of the macro-action algorithms on much larger POMDP problems.

is expected to be. Reflecting this intuition, we use a Gaussian observation model where the noise covariance Σ_{zi} is a function of the position of the helicopter and target i:

$$\begin{bmatrix} z_{xi} \\ z_{yi} \\ z_{\theta i} \end{bmatrix} = f\left(\begin{bmatrix} x_i \\ y_i \\ \theta_i \end{bmatrix}\right) + \mathcal{N}(0, \Sigma_{zi}) \tag{6.15}$$

$$\Sigma_{zi} = g(x_i, y_i, x_a, y_a, h_a), \tag{6.16}$$

where x_i, y_i, θ_i is the pose of target i, while x_a, y_a, h_a correspond to the agent's position and height in the environment. $z_{xi}, z_{yi}, z_{\theta_i}$ is the observation of target i in image coordinates.

The covariance function itself is specified as

$$g(x_i, y_i, x_a, y_a, h_a) = C_1 h_a + C_2 \frac{\left(\begin{bmatrix} x_i \\ y_i \end{bmatrix} - \begin{bmatrix} x_a \\ y_a \end{bmatrix}\right)\left(\begin{bmatrix} x_i \\ y_i \end{bmatrix} - \begin{bmatrix} x_a \\ y_a \end{bmatrix}\right)^T}{h_a} + C_3, \tag{6.17}$$

where C_1, C_2 and C_3 are constants.

In the generic belief update expression where the target position, $s_i = [x_i; y_i; \theta_i]$, is unknown,

$$b'(s_i') \propto p(z|s_i', a, \Sigma_{zi}) \int_{s_i} p(s_i'|s_i, a) b(s_i) ds_i \quad s.t. \quad \int_{s_i'} b'(s_i') ds_i' = 1, \tag{6.18}$$

which means that each possible s_i' would be associated with a different covariance Σ_{zi}. Performing this integration exactly would not keep the distribution Gaussian. Instead, we approximate the observation model by computing a single expected covariance $\hat{\Sigma}_{zi}$ given the current belief distribution:

$$\hat{\Sigma}_{zi} = E[\Sigma_{zi}] = \int_{s_i} b(s_i) \Sigma_{zi}(s_i) ds_i. \tag{6.19}$$

Substituting in the exact expressions for the covariance function and the belief after an action is taken but before incorporating the measurement, $b^a(s) \sim \mathcal{N}(s_i|\overline{\mu}, \overline{\Sigma})$, we get:

$$E[\Sigma_{zi}] = \int \mathcal{N}\left(\begin{bmatrix} x_i \\ y_i \end{bmatrix} \middle| \overline{\mu}_{xy}, \overline{\Sigma}_{xy}\right)\left(C_1 h_a - \frac{C_2}{h_a}\left(\begin{bmatrix} x_i \\ y_i \end{bmatrix} - \begin{bmatrix} x_a \\ y_a \end{bmatrix}\right)\left(\begin{bmatrix} x_i \\ y_i \end{bmatrix} - \begin{bmatrix} x_a \\ y_a \end{bmatrix}\right)^T + C_3\right) dx_i dy_i. \tag{6.20}$$

and by adding and subtracting $\overline{\mu}_{xy}$ from the second term, reduces to

$$E[\Sigma_{zi}] = C_1 h_a + \frac{C_2}{h_a}\left(\overline{\mu}_{xy} - \begin{bmatrix} x_a \\ y_a \end{bmatrix}\right)\left(\overline{\mu}_{xy} - \begin{bmatrix} x_a \\ y_a \end{bmatrix}\right)^T + \frac{C_2}{h_a}\overline{\Sigma}_{xy} \tag{6.21}$$

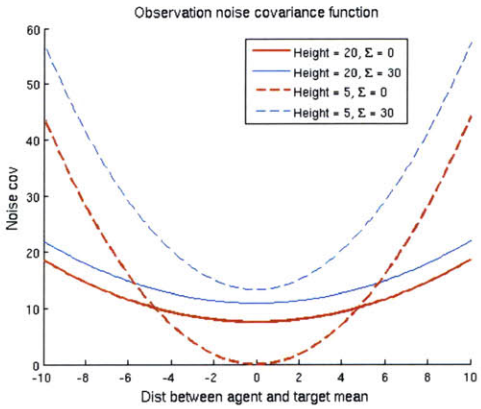


Figure 6-2: The observation noise covariance is a function of the height of helicopter, the distance between the helicopter and the mean of the target belief, and the covariance of the target belief. At lower altitudes, the helicopter can make better observations of targets close to it, but has a limited field of vision. At higher heights, the helicopter can see a larger area but even close targets are more noisily observed.

where $\overline{\mu}_{xy}, \overline{\Sigma}_{xy}$ refer to the translational components of the agent's belief.

In contrast to simpler observation models, our observation model has the desirable characteristic that if a target's location is very uncertain, i.e. its covariance $\overline{\Sigma}_{xy}$ is very large, then even if the target's mean location is close to the helicopter's mean location, the expected benefit of receiving an observation (in terms of reducing the target's uncertainty) is still small. This property results from the above derivation of $E[\Sigma_{zi}]$, which includes the current target covariance $\overline{\Sigma}_{xy}$. Figure 6-2 provides an illustration of the expected covariance for different locations of the target relative to the agent, agent heights, and target belief covariances.

The agent's pose is fully observable, though the actions that it takes are subject to a small amount of additive Gaussian noise. As a result, unlike the RockSample domains, the open-loop nature of macro-actions means that the planner cannot perfectly predict the vehicles' pose at the end of the macro-action. Each target's motion is determined by its translational and rotational velocities. The model provides the agent with a prior over these velocities, but at every timestep, the target's true velocities are additive functions of these fixed input controls and Gaussian noise. In the parametric formulation, the agent maintains a Gaussian belief over each target's state, and in order to compare MAD, we discretise the continuous state spaces of the agent's and targets' positions, and maintain a probability distribution over each discrete target state. Due to computational memory constraints, for a $100m$ by $100m$ by $20m$ target-monitoring problem in the x, y and z directions, we were limited to a discretization with $10m$ resolution in the x, y directions, $5m$ in the z direction, and $45°$ angular resolution.

We focus on a decision-theoretic version of the original sensor resource management problem in Scott et al. (Scott et al., 2009), where at each timestep the agent must decide if each of the targets is inside an area of interest. These areas of interest are indicated by the yellow squares in Figure 6-1. The agent receives a

	1 Target		2 Targets		3 Targets		8 Targets	
	Avg rewards	Online time (s)	Avg rewards	Online time (s)	Avg rewards	Online time (s)	Avg rewards	Online time (s)
Greedy	-21.50	0.0657	-26.50	0.19	-18.00	0.46	-95.00	2.01
WT-Single	**65.14**	0.00075	-27.03	0.00068	-23.52	0.00080	-71.17	0.00063
WT-Macro	64.64	0.00076	-19.05	0.00042	-10.53	0.00037	-52.98	0.00025
NBO	-5.80	0.051	-10.78	0.21	-8.27	0.63	-5.98	5.78
MAD(d2,s3)	1.27	1.66	0.97	8.46	-1.86	26.96	27.36	432.13
MAC(d2,s10)	41.73	4.73	46.67	22.13	37.89	70.91	83.86	711.67
PBD(d2,s10)	36.21	0.89	**68.00**	4.33	**55.78**	13.02	**120.80**	132.50

Table 6.5: TARGETREPORT Results. Run for 200 timesteps.

positive reward if it successfully reports that a target is in an interest region, a negative reward if it wrongly decides that the target is in the region, and no reward if it decides that the target is not in the region, regardless of the target's actual state. Small costs are incurred for the agent's motion. We call this the TARGETREPORT problem.

Macro-actions are generated by computing the sequence of actions that will enable the agent to hover at different altitudes over the means of each target belief, assuming that there is no process noise in the agent's actions, as well as a hovering macro-action that consists of hovering at the agent's current location for a few timesteps. We compare the macro-action algorithms to a range of intuitive strategies and prior approaches. The first algorithm is the greedy strategy, which returns the primitive action that will produce the largest expected reward in the next timestep. The next two approaches are the Worst Target (WT) policies, which are hand-coded policies that sends the agent to the target with the largest uncertainty. The intuition here is that generally, the agent's goal is to localize the targets in the environment. The two algorithms differ based on whether the agent chooses a new target to travel to after each timestep (WT-single), or re-plans only after it has reached the target it had initially chosen (WT-macro). Finally, we compared our algorithm to the nominal belief optimization (NBO) algorithm proposed by Scott et al. (2009). The NBO algorithm also assumes a Kalman Filter model for the target-tracking problem, but rather than considering the entire distribution of posterior beliefs, only the most likely posterior belief after an action is considered. In this algorithm, the most likely posterior belief for a Gaussian belief update is given by the posterior mean without incorporating any observations, and the covariance by performing the covariance update while linearizing about the most likely mean at each step. Although the original algorithm uses an optimization approach to search for action sequences, here we modify the NBO algorithm by adopting a forward search approach, evaluating a macro-action based on the most likely posterior belief.[2]

Table 6.5 presents results for the TARGETREPORT problem, comparing the algorithms in scenarios with different number of targets. These results demonstrate that the PBD algorithm, with its closed form representation of the distribution of posterior beliefs after an action, finds a significantly better policy than alternate

[2]As noted by the authors, the NBO algorithm focuses on a new method for approximating the Q-value, rather than on the optimization techniques. While they adopt a generic search approach for performing the optimization, the authors also point to forward-search POMDP algorithms as good search techniques in which their Q-value approximations could be incorporated. Our use of forward search with the NBO Q-value approximation should therefore be considered an implementation artifact that does not affect the results.

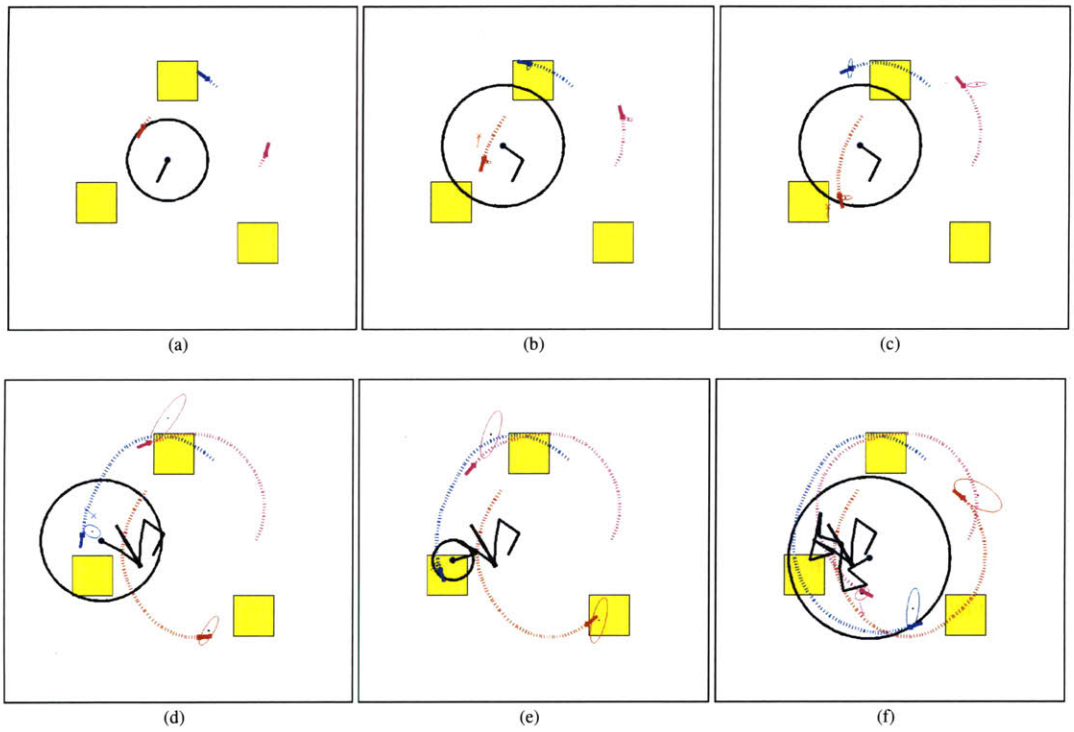


Figure 6-3: Snapshots of the PBD policy being executed. The black circle indicates the field-of-view of the agent's sensor, which is directly proportionate to the agent's height. The size of the error ellipses indicate the agent's uncertainty associated with each target at each timestep. The agent alternates between flying at a high altitude to maximize the likelihood of observing targets (b,f) and focusing on a single target that is near/has entered an area of interest (e).

approaches. Figure 6-3 demonstrates a typical policy executed by the PBD algorithm. The agent begins in the middle of the grid world, and approaches a target at a high altitude (Figure 6-3b), maximizing the likelihood of localizing that target. If none of the targets seem to be approaching a region of interest, the agent hovers in the same position to conserve energy (Figure 6-3c). When one of the targets may potentially be entering a region of interest, the agent focuses on that target, tracking it carefully to ensure that it knows when the target is exactly in the region of interest (Figure 6-3d,e). The agent subsequently travels to a high altitude and repeats the process of localizing another target with potential rewards (Figure 6-3f).

Considering the entire distribution of posterior beliefs, rather than just the maximum likelihood posterior belief, is valuable because the agent is able to reason that there is a possibility that the target could be within a region of interest. This is in contrast to the NBO approach, which only considers the most likely posterior belief, and will therefore seek to localize the target only if the mean of its belief appears to be heading into a region of interest. While the consideration of the entire distribution of posterior beliefs necessarily incurs greater computational cost, we demonstrate in Section 6.4 that we are able to track two targets in real-time using an implementation of the PBD algorithm that has not been optimized for speed.

Table 6.5 also shows that because the PBD algorithm directly computes the distribution of posterior beliefs after a macro-action, the computational cost of the PBD algorithm is significantly lower than the MAC algorithm. The MAC algorithm suffers a greater computational cost as it generates the set of posterior beliefs after a macro-action by sampling observation sequences and explicitly performs belief updates along each sample trajectory. In addition, because the TARGETREPORT problem has a state space that is fundamentally continuous, the resolution of the state space discretization that was achievable given computational memory constraints was still unable to capture the inherent characteristics of the target-tracking problem, resulting in the poor performance of MAD in the TARGETREPORT problem.

In the single-target case, we also observed the result that the PBD algorithm does worse than the hand-coded policy of the agent traveling to the target with the largest uncertainty (WT-single). When the problem only involves a single target, such a policy equates to having the agent hover over the sole target at every step, which is the optimal policy in the single target case. In contrast, we observe that the MAC and PBD algorithms return policies that result in the agent periodically leaving the target to fly to a higher altitude, resulting in greater noise in the observations and corresponding loss of rewards on average. By restricting the MAC and PBD algorithms to plan with macro-actions, we are restricting the agent to plan with open-loop action sequences in order to perform deeper search, rather than a conditional plan that is conditioned on the observations after each *primitive* action. Even though the agent re-plans after every timestep, without this conditional plan, an agent executing the MAC or PBD algorithms will execute the "safe" policy and fly to a higher altitude, which maximizes the likelihood of keeping the target well-localized when it is unable to condition its actions based on subsequent observations. This example highlights the tradeoff we make by considering a smaller class of policies (those that can be expressed as chains of macro-actions) compared to the full policy set. While in simple problems, such as a single-target TARGETREPORT problem, the policy restriction can clearly be a limitation, our macro-action algorithms perform significantly better than the other benchmark approaches when there are multiple targets, in scenarios that are arguably more complicated and require more sophisticated planning algorithms.

6.3 Benchmark problems directly from the literature

For completeness, we also examine the performance of our macro-action algorithms on variants of the Rock-Sample and target-tracking problems that are more commonly addressed in their respective research communities. The FieldVisionRockSample (FVRS) problem is equivalent to an instantiation of ISRS where the information beacons are located in the same position as the rock they represent. Table 6.6 shows that the MAD algorithm performs similarly to existing offline POMDP solvers, while the MAC and PBD algorithms do slightly worse due to the approximations necessary to perform the parametric belief updates. Nevertheless, in the FVRS problem formulation, information gathering actions are similar to reward exploitation actions, making long-horizon planning unnecessary for generating optimal policies. Table 6.6 shows that a naïve

(a)

	FVRS[8,5]		
	Avg rewards	Online time (s)	Offline time (s)
QMDP	18.58 ± 0.49	0	3.00
HSVI2(150s)	18.46 ± 2.53	0.021	150
SARSOP(150s)	**21.04** ± 0.076	0.0063	150
RTBSS(d5, s10)	20.12 ± 0.48	19.04	0
MAC(d2,s200)	17.72 ± 1.29	0.035	0
PBD(d2,s200)	19.85 ± 0.43	0.004	0
MAD(d2,s200)	20.69 ± 0.27	0.015	0
Fully obs.	21.37	N.A.	N.A.

(b)

	FVRS[100,30]	
	Avg rewards	Online time (s)
MAC(d3,s5)	59.51	104.04
PBD(d3,s5)	59.57	**19.48**
MAD(d3,s5)	**65.87**	32.17
Fully obs.	66.61	N.A.

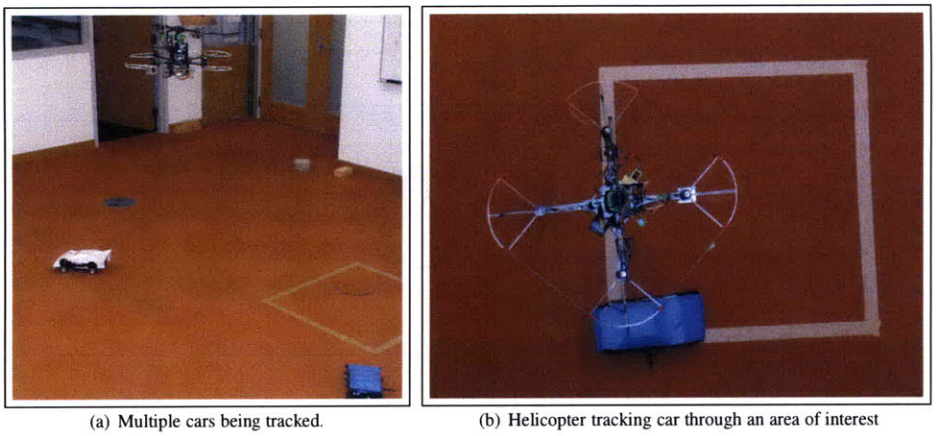
Table 6.6: FVRS results. For the forward search algorithms, the numbers in brackets represent the search depth (d) and number of posterior beliefs obtained (s) at the end of each action/macro-action. Standard error values for FVRS[8,5] are shown.

	1 Target		2 Targets		3 Targets		8 Targets	
	Avg rewards	Online time (s)	Avg rewards	Online time (s)	Avg rewards	Online time (s)	Avg rewards	Online time (s)
Greedy	-79.92	0.0042	-1009.56	0.012	-1665.99	0.0085	-4441.55	0.065
WT-Single	**-6.73**	0.00070	**-995.91**	0.00098	-1710.82	0.00053	-4690.30	0.00080
WT-Macro	-6.77	0.00074	-1011.23	0.00061	-1700.29	0.00023	-4560.98	0.00030
NBO	-6.75	0.029	-1014.00	0.18	-1589.10	0.19	-4380.32	3.81
MAD(d2,s3)	-595.00	2.40	-1227.91	10.67	-1808.48	33.13	-4816.39	513.82
MAC(d2,s10)	-6.76	2.97	-1061.82	22.12	**-1552.59**	23.14	-4394.71	571.36
PBD(d2,s10)	-6.77	0.25	-1077.24	1.67	-1580.79	1.71	**-4361.81**	39.46

Table 6.7: TARGETSURVEY problem. Run for 200 timesteps.

QMDP solver does reasonably well on the FVRS problem. Since QMDP assumes that the problem becomes fully observable after a single action, its good performance suggests that it is unnecessary to explicitly account for persistent uncertainty when making decisions in the FVRS problem. These results provide some justification for why we believe the ISRS problem is a more interesting problem domain for planning under uncertainty in partially observable domains.

Within the target-monitoring domain, target-surveying is a popular application where an agent is rewarded for maintaining a small uncertainty over the targets positions. As discussed in Section 2.1.4, the sensor resource management community often uses an information-theoretic reward function when formulating the target-surveying problem. A reward model which matches this objective is when the reward is proportional to the negative sum of the trace of the target belief covariances. We refer to this problem as the TARGETSURVEY problem, and Table 6.7 shows that the MAC and PBD algorithms perform approximately as well as NBO, which only considers the most likely posterior belief. This is unsurprising as the reward function is directly a function of the covariance, which is independent of the actual observations received. In this case, it is satisfying that maintaining a distribution over the posterior beliefs, which are explicitly computed in the MAC and PBD algorithms, does not lead to a loss in performance, but also does not provide an advantage over the simpler NBO approach.



(a) Multiple cars being tracked.

(b) Helicopter tracking car through an area of interest

Figure 6-4: TARGETREPORT demonstration with helicopter. The helicopter has to simultaneously track two cars and report whenever either car enters an area of interest.

6.4 Real-world helicopter experiments

Finally, as a proof of concept, we demonstrate the PBD algorithm on a live instantiation of the TARGETREPORT problem. We simulate the MAV08 scenario that was presented in Chapter 1 by having an aerial vehicle guide ground units to a hostage building while avoiding an enemy guard vehicle. Our aerial vehicle therefore had to plan paths in order to be able to monitor the different ground objects and report whenever any of them arrived at an area of interest.

We demonstrate this scenario on the autonomous quadrotor helicopters that we have previously developed. We mounted a downward-facing camera that makes observations of the target. Since target detection is not the focus of this thesis, each of the ground vehicles had a known, distinctive color, making them easy to detect and distinguish with a simple blob detection algorithm. Given the helicopter's position in the world and the image coordinates of the detected object, we can recover an estimate of the position and orientation of a target observation in global coordinates. Nevertheless, the helicopter only receives an observation of the target when the target is within the camera's field-of-view, and although the helicopter platform hovers relatively stably, slight oscillations persist, which result in noisier observations when the helicopter is flying at higher altitudes. Hence, the helicopter has to choose actions that balance between obtaining more accurate observations at low altitudes and a larger field-of-view by flying high.

Two ground vehicles were driven autonomously in the environment with open-loop control, and the helicopter had to plan actions that would accurately localize both targets. To replicate the TARGETREPORT problem, we marked out three areas of interest where the helicopter had to predict at every timestep if the targets are within those areas (Figure 6-4c). We applied the PBD algorithm to plan paths for the helicopter that maximize the likelihood that it can accurately report whenever a target is in an area of interest. However, rather than sending open-loop control actions to the helicopter, as we did in the simulation experiments, for

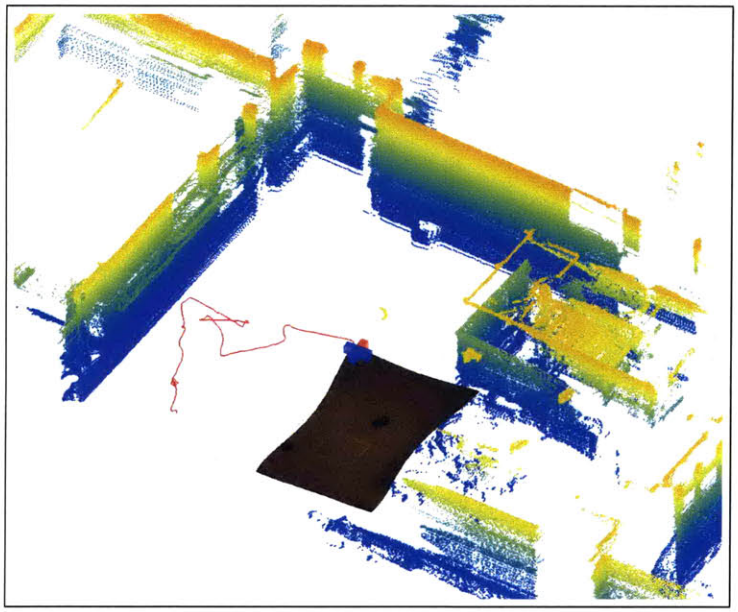


Figure 6-5: The helicopter (blue/red cross) uses an onboard laser scanner to localize itself. A downward pointing camera is used to observe the ground targets. In this figure, the camera image from the onboard camera is projected onto the ground plane.

	# Target entry detections	# True target entries	Flight time (s)	Dist. traveled (m)
WT-Single	1	7	484.15	243.36
NBO	1	4	435.25	247.01
PBD	**4**	6	474.64	282.51

Table 6.8: Performance of algorithms on real-world helicopter experiment. Ground truth was found using an overhead video camera.

safety reasons we closed the loop around the position of the helicopter, sending desired waypoints that we wanted the helicopter to navigate to. Nevertheless, the helicopter's true state in the world is actually partially observable, and the helicopter has to rely on an onboard laser scanner to localize its position in the environment.

Figure 6-5 shows a 3D view of the helicopter performing autonomous target tracking of the ground targets. As the helicopter flies around the environment, it obtains observations of the target, which are then used to update the agent's belief of the targets. Figure 6-6 provides snapshots of the helicopter executing a plan that is computed online by the PBD algorithm. The helicopter exhibits similar behaviors to those that were observed in the simulation experiments. The helicopter alternates between the two targets in the environment to report when either target is in an area of interest. When the agent had a large uncertainty over a particular target's location, it would also fly to a higher altitude in order to increase its sensor field-of-view, thereby maximizing the likelihood that it will be able to re-localize the targets. A video of the complete system in action is available at: `http://groups.csail.mit.edu/rrg/videos.html`.

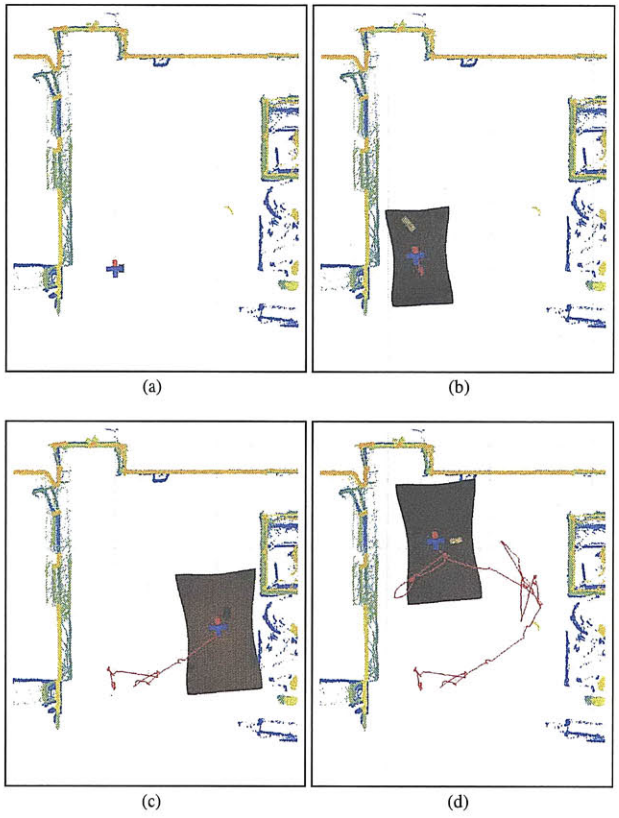


Figure 6-6: Bird's eye-view snapshots of the helicopter's trajectory (red), based on policy generated by the PBD algorithm. The helicopter (blue/red cross) alternates between observing the white (b,d) and blue (c) cars in order to accurately report when either car is in an area of interest. The area of the field-of-view of the agent's camera sensor varies directly with the height that the agent is flying at.

As a coarse measure of achieved reward, we evaluated how well the helicopter running PBD did at reporting when a target entered an area of interest, and compared it to the WT-Single and NBO algorithms. The ground truth of the number of times the targets actually entered the areas of interests in each trial was found by using a video camera mounted overhead above the environment. Table 6.8 indicates that the PBD algorithm did a much better job of reporting the targets' positions than both the WT-Single and NBO algorithms. In particular, we observed that both the WT-Single and NBO algorithms seldom took advantage of the ability to increase the agent's sensor field-of-view by having the agent fly to a higher altitude. An agent applying these two algorithms therefore had a higher probability of losing track of the targets completely.

6.5 Conclusion and future work

This chapter builds upon the previous chapter and presents experimental results evaluating the performance of our macro-action algorithms. Our algorithms were applied to problem domains that span multiple research communities, and consistently performed better than prior approaches in problems where it is necessary to perform far look-ahead and/or it is necessary to explicitly account for the distribution of posterior beliefs. Finally, we demonstrated our algorithm on a real robotic helicopter, underscoring the applicability of our algorithm for planning in real-world, long-horizon, partially observable domains.

Our work has led to a number of interesting directions for future work. Our experience with the single-target TARGETREPORT problem suggests that in some scenarios, having access to the set of primitive actions is important, at least for a couple of look-ahead steps. One possible extension to the algorithms in this chapter would therefore be to perform a short naïve factored full forward search, and then use the macro-actions as a value heuristic at the leaves. In this setup, it remains an open question as to how to balance between expensive, but more accurate full forward search, and the faster, restricted-policy-class macro-actions.

Chapter 7

MMPBD: Posterior belief distribution for multi-modal Gaussian beliefs

In Chapter 5, we showed that for problem domains where the agent's belief is representable as a uni-modal Gaussian distribution, the sufficient statistics of the agent's posterior belief after a macro-action is executed can be directly computed without having to enumerate the possible observations. This property enables the agent to plan more efficiently with macro-actions, thereby allowing the planner to either consider a larger set of candidate macro-actions or search to a further horizon within the same amount of planning time.

Unfortunately, the uni-modal Gaussian assumption for the agent's belief can be highly restrictive. For example, problems such as an autonomous agent navigating over varying terrain, a robotic arm grasping complex objects, or an agent tracking targets moving along a road network have highly complex and non-linear transition and observation models, making it impossible to maintain the agent's belief as a uni-modal Gaussian distribution. For example, when a target is moving along a road network and reaches a road junction, it can choose to move along either of the roads leading out from that junction. This means that the agent may have to maintain a belief over multiple road segments, and hence it would be inadequate for a planner to restrict the agent's belief to uni-modal Gaussian beliefs.

It is possible to argue that it is more straightforward to represent the complicated dynamics of the above-mentioned problem domains by operating in discrete-state environments, and use state-transition matrices that adhere to a POMDP problem formulation. Nevertheless, using linear Gaussian system models and Gaussian belief representations, if valid, can result in efficient belief-updating and planning, as demonstrated in Chapter 5. In this chapter, we extend the PBD algorithm from Chapter 5 to problem domains where the agent's belief have to be represented as multi-modal Gaussian distributions. When the problem domain's transition model can be described as a switching state-space dynamics model, and the observation model has a bi-modal characteristic of generating observations with both positive and negative information, we show that we can similarly approximate the distribution of posterior beliefs at the end of a macro-action in an

efficient manner, thereby enabling the planner to search deeper. We present the Multi-modal Posterior Belief Distribution (MMPBD) algorithm, an online, forward-search, planning-under-uncertainty algorithm for planning when the agent's belief is a multi-modal Gaussian distribution over the state space.

After presenting a generic problem formulation that motivates the need to represent the agent's beliefs as multi-modal Gaussian distributions, we provide the necessary equations for performing belief updates in these problem domains, in order to obtain posterior beliefs after actions are executed and observations are obtained. We then show how the distribution of posterior beliefs after a macro-action can be computed for multi-modal Gaussian beliefs, making explicit additional approximations that enable efficient computation of this distribution of posterior beliefs. Finally, we present a variation of the target-tracking problem presented in earlier chapters by constraining the target to move along a road-network, thereby making it imperative that the agent's belief is represented as a multi-modal Gaussian. Simulation results compare our algorithm to both the greedy strategy and a forward search strategy that does not incorporate macro-actions. Finally, we demonstrate our algorithm on an actual quadrotor helicopter tracking two autonomous ground vehicles.

7.1 Problem formulation

Recall from Chapter 5 that when the transition and observation models of a problem domain are both linear with Gaussian noise, and the agent's initial belief is a uni-modal Gaussian distribution over the state space, then the agent's posterior belief always remains a uni-modal Gaussian distribution at every timestep. Furthermore, the Kalman filter provides a closed form, efficient means of performing the belief update.

We have presented the equations of the linear Gaussian transition and observation models previously in Chapter 5, but restate them here for convenience:

$$s_t = A_t s_{t-1} + B_t a_{t-1} + \epsilon_t \qquad \epsilon_t \sim \mathcal{N}(0, P_t) \tag{7.1}$$

$$z_t = C_t s_t + \delta_t \qquad \delta_t \sim \mathcal{N}(0, Q_t) \tag{7.2}$$

where ϵ_t and δ_t are the Gaussian noise associated with the transition and observation models respectively.

These equations can alternatively be written in the following form:

$$p(s_t|s_{t-1}, a_{t-1}) = \mathcal{N}(s_t; A_t s_{t-1} + B_t a_{t-1}, P_t) \tag{7.3}$$

$$p(z_t|s_t) = \mathcal{N}(z_t; C_t s_t, Q_t) \tag{7.4}$$

where $\mathcal{N}(a; b, c)$ implies that the variable a is normally distributed with mean b and covariance c.

7.1.1 Transition model

Unfortunately, for many problem domains, the linear Gaussian model presented above is highly restrictive, and does not allow the true transition and observation functions of the problem domain to be adequately represented. In this chapter, we provide a problem formulation that includes transition and observation models that are more general in applicability than the linear Gaussian model, and which motivates the need for a multi-modal Gaussian representation of the agent's belief

Specifically, for the transition model, we borrow from Brunskill et al. (2008) and adopt a switching state-space dynamics model that enables us to represent multi-modal state-dependent dynamics. Switching state-space dynamics models are popular in the controls community for representing systems with complex dynamics (Ghahramani & Hinton, 2000), and these models are also known as hybrid models and jump-linear systems. Switching state-space dynamics models consist of a set of linear Gaussian state transition models, and at each timestep, a hidden, discrete switching state indexes which linear Gaussian model is being used to update the hidden continuous-state vector. For example, in a road-constrained target-tracking problem, the targets' locations make up the continuous-state vector, while the discrete switching state indicates which road segment each target is on or can choose to get onto. In addition, the switching state transitions are conditioned only on the previous continuous-state vector, and is independent of the switching state at the previous timestep. The continuous-state vector therefore fully describes the state of the system, whereas the discrete switching state only indicates which of the linear Gaussian transition models are applicable based on the continuous-state vector. The switching state is therefore purely an artifact of the way we have defined our state-transition model. For example, in the context of the target-tracking problem, whether a target can transition to a different road segment at every timestep depends solely on whether the target resides near a road junction.

Formally, the transition model can be written as a collection of state-dependent multi-modal dynamics, as follows:

$$p(s_t|s_{t-1}, a_{t-1}) = \sum_{S_i \in \mathcal{SS}} p(s_t|s_{t-1}, S_i, a_{t-1}) p(S_i|s_{t-1}, a_{t-1}) \tag{7.5}$$

where S_i is a discrete switching state, $\mathcal{SS}$ is the set of discrete switching states, $p(s_t|s_{t-1}, S_i, a_{t-1})$ describe the dynamics for switching state S_i, and $p(S_i|s_{t-1}, a_{t-1})$ is the probability that S_i describes the problem dynamics, conditioned on the previous state s_{t-1} and action a_{t-1}. The switching state probabilities $p(S_i|s_{t-1}, a_{t-1})$ for any state s_{t-1} and action a_{t-1} sums to 1. Note that the despite the presence of switching states S in the transition model, the state space of the problem domain remains independent of the switching state, since the switching state at time t is a function of only the previous continuous state vector s_{t-1} and the action executed a_t, and is independent of the previous switching state at time $t-1$.

For each switching state S_i, the individual dynamics remain linear Gaussian, and can be expressed ac-

cording to

$$p(s_t|s_{t-1}, S_i, a_{t-1}) = \mathcal{N}(s_t; A_{t,i}s_{t-1} + B_{t,i}a_{t-1}, P_{t,i}) \tag{7.6}$$

Compared to the linear Gaussian model, the more complicated transition model implies that the agent's belief will not always be representable as a uni-modal Gaussian belief. Instead, we assume that the agent's belief is represented as a multi-modal Gaussian distribution over the state space, which can be expressed as follows:

$$b_t \sim \sum_j w_{t,j}\mathcal{N}(\mu_{t,j}, \Sigma_{t,j}) \tag{7.7}$$

where $\mu_{t,j}$ and $\Sigma_{t,j}$ are the means and covariances of mode j of the agent's belief at time t, and $w_{t,j}$ is the weight of mode j.

7.1.2 Observation model

Many observation sensors, especially those that robotic agents are typically equipped with, can be described as emitting observations that have a bi-modal characteristic. Sensors such as pinhole cameras, laser range-finders, radar sensors, etc. all generate noisy observations of the world around them, but due to their limited range, field-of-view and noisy characteristics, the sensors cannot provide omniscient information of every object of interest in the environment. No matter what the object of interest is, e.g. navigational landmarks, moving targets etc., the agent may not always get a positive observation of the object of interest, but may instead obtain an observation with negative information, which we refer to as negative observations.

Following Hoffman et al. (2006), negative information is obtained from an ascertained absence of an expected sensor reading at any timestep. Note that our definition of negative information is different from the definition often adopted within the statistics literature (Bradlow, 1996), where observations with negative information results in the agent becoming more uncertain after incorporating the observation than it was before. Instead, our definition of negative observations still results in positive information gain, though often of a smaller amount, since a negative observation only eliminates a possible state and leaves many possible alternatives unevaluated. Nevertheless, if these negative observations are accumulated over time, they can still reveal significant information on the state of the world. Together with positive observations, negative observations allow a much richer set of observation models to be expressed, and when the agent's belief is represented as a multi-modal Gaussian distribution, negative observations also facilitate the extinguishing of some of the Gaussian modes.

Our observation model therefore has a bi-modal characteristic of generating either a positive or negative observation at every timestep. Positive observations are generated according to a linear Gaussian model, while the negative observation typically has a fixed sensor signature. The type of observation that is emitted

at every timestep is a function of the state s_t, though there is some non-zero probability that any positive observation is a false positive (f_P), or that the negative observations are false negatives (f_N).

More formally, given a decomposition of the state space $\mathcal{S}$ into two subsets $\mathcal{S}_{pos}$ and $\mathcal{S}_{neg}$, corresponding to the set of states that should emit positive and negative observations respectively, the observation model can be described according to

$$\text{if} \quad s_t \in \mathcal{S}_{pos}, \qquad p(z_t|s_t) = \begin{cases} (1 - f_N)\mathcal{N}(z_t; C_t s_t, Q_t) & z_t \in \mathcal{Z}_{pos} \\ f_N & z_t \in \mathcal{Z}_{neg} \end{cases} \tag{7.8}$$

$$\text{if} \quad s_t \in \mathcal{S}_{neg}, \qquad p(z_t|s_t) = \begin{cases} f_P & z_t \in \mathcal{Z}_{pos} \\ 1 - f_P & z_t \in \mathcal{Z}_{neg} \end{cases} \tag{7.9}$$

7.2 Belief updating for multi-modal Gaussian distributions

The transition and observation models presented above are both extensions of the linear Gaussian model, while a multi-modal Gaussian belief representation is essentially a weighted sum of uni-modal Gaussian beliefs. Given the similarities of both the system models and the belief representations to the linear Gaussian model, we can leverage aspects of the Kalman filter algorithm to perform belief updates in the multi-modal case. Algorithm 8 presents our modified version of the Kalman filter for performing multi-modal Gaussian belief updates. This algorithm is similar to the sum of Gaussians Kalman filter from the literature (Sorenson & Alspach, 1971).

7.2.1 Transition update

Following Sorenson et al. (1971), at each iteration, modes within one standard deviation of each other are first collapsed into a single mode, so as to keep the belief representation as compact as possible. The Gaussian parameters of the new mode are computed by re-fitting a Gaussian to the original modes, while the new weight is found by summing the weights of the original modes.

Each of the resulting modes is then propagated forward in time based on the transition dynamics of the problem domain. Given the switching state dynamics model, the equations for the transition update depend on the distribution of the switching states given the action a_t taken from each Gaussian mode $(\mu_{t-1,j}, \Sigma_{t-1,j})$. As the agent's state is not fully observable, we make an approximation by using the mean of each mode $\mu_{t-1,j}$, such that the probability of each resulting switching state S_i is $p(S_i|\mu_{t-1,j}, a_t)$.

In addition, to prevent an unbounded growth in the number of modes, we assume that for most of the state space, only one switching state has a non-zero probability, i.e. $p(S_i|s_{t-1}, a_{t-1}) = 1$ for some S_i. This assumption implies that the problem domain is linear Gaussian for most of the state space, except for a few states where the dynamics actually exhibit switching characteristics. For example, in the road-constrained target-tracking problem, for most of each road segment, only a single switching state has non-zero probability. Only at the ends of the road segments are there multiple switching states that have non-zero probability.

Algorithm 8 BELIEFUPDATE() for multi-modal Gaussian beliefs

Require: Belief state b_{t-1}, action a_t, observation z_t

Collapse modes that are within 1 std dev.
(Transition update)
for each mode j **do**
  if $\max_i p(S_i|\mu_{t-1,j}, a_t) = 1$ **then**
    $\overline{\mu}_{t,j} = A_{t,i}\mu_{t-1,j} + B_{t,i}a_{t-1}, \qquad \overline{\Sigma}_{t,j} = A_{t,i}\Sigma_{t-1,j}A_{t,i}^T + P_{t,i}$
  else
    for all i such that $p(S_i|\mu_{t-1,j}, a_{t-1}) > 0$ **do**
      $\overline{w}_{t,j,i} = w_{t-1,j}p(S_i|\mu_{t,j}, a_{t-1})$
      $\overline{\mu}_{t,j,i} = A_{t,i}\mu_{t-1,j} + B_{t,i}a_{t-1}, \qquad \overline{\Sigma}_{t,j,i} = A_{t,i}\Sigma_{t-1,j}A_{t,i}^T + P_{t,i}$
    end for
  end if
end for
Re-number all modes e.g. $\overline{w}_{t,j,i} \rightarrow \overline{w}_{t,j'} \quad \forall i, j$
(Observation update)
if Positive observation **then**
  Associate z_t to a set of modes $\mathcal{W}_z$
  for each mode j **do**
    if mode belongs to $\mathcal{W}_z$ **then**
      $K_{t,j} = \overline{\Sigma}_{t,j}C_t^T(C_t\overline{\Sigma}_{t,j}C_t^T + Q_t)^{-1}$
      $\mu_{t,j} = \overline{\mu}_{t,j} + K_{t,j}(z_t - C_t\overline{\mu}_{t,j}), \qquad \Sigma_{t,j} = (C_t^T Q_t^{-1} C_t + \overline{\Sigma}_{t,j}^{-1})^{-1}$
    else
      $w_{t,j} = f_P\overline{w}_{t,j}.$
    end if
  end for
else {Negative observation}
  for each mode j **do**
    Truncate and refit $(\overline{\mu}_{t,j}, \overline{\Sigma}_{t,j})$
    $w_{t,j} = \overline{w}_{t,j}(1 - (1 - f_N)\times$frac. truncated$)$
  end for
end if
Re-normalize weights $w_{t,j}\forall j$

If the probability mass is concentrated in a single switching state S_i, i.e. $p(S_i|\mu_{t-1,j}, a_{t-1}) = 1$, then the mode statistics $\overline{\mu}_{t,j}$ and $\overline{\Sigma}_{t,j}$ are found by propagating the prior belief according to the linear Gaussian model associated with switching state S_i,

$$\overline{\mu}_{t,j} = A_{t,i}\mu_{t-1,j} + B_{t,i}a_{t-1}, \qquad \overline{\Sigma}_{t,j} = A_{t,i}\Sigma_{t-1,j}A_{t,i}^T + P_{t,i} \tag{7.10}$$

$$\overline{w}_{t,j} = w_{t,j} \tag{7.11}$$

while the weights of the mode $\overline{w}_{t,j}$ remain unchanged.

In contrast, if multiple switching states have non-zero probability mass, then additional modes are created corresponding to each of the switching states that have non-zero probability of occurrence. The weights of each of the new modes $\overline{w}_{t,j,i}$ can be found using the weight of the old mode $w_{t,j}$ and the switching state

probabilities, according to

$$\overline{w}_{t,j,i} = w_{t-1,j} \;\; p(S_i|\mu_{t-1,j}, a_{t-1}) \qquad \forall \;\; S_i \in \mathcal{SS} \;\; \text{s.t.} \;\; p(S_i|\mu_{t-1,j}, a_{t-1}) > 0 \tag{7.12}$$

Similarly, the mode statistics are again propagated according to the linear Gaussian models of switching state S_i

$$\overline{\mu}_{t,j,i} = A_{t,i}\mu_{t-1,j} + B_{t,i}a_{t-1}, \qquad \overline{\Sigma}_{t,j,i} = A_{t,i}\Sigma_{t-1,j}A_{t,i}^T + P_{t,i} \tag{7.13}$$

7.2.2 Observation update

The observation update depends on whether a positive or negative observation is obtained. If a positive observation is obtained, we first perform a data association step linking the observation to modes that could be associated to the observation. The specific data association procedure is problem specific and depends on the properties of the observation variable z_t and the observation matrix C_t. Assuming that $C_t^T C_t$ is invertible, z_t is first mapped into the state space by using the pseudo-inverse of C_t, according to

$$\hat{z}_t = (C_t^T C_t)^{-}1 C_t^T z_t \tag{7.14}$$

Modes can then be associated to the observation z_t by finding those modes that have high probability of emitting observation z_t. Specifically, we say that mode j is associated with observation z_t if

$$\mathcal{N}(\hat{z}_t|\overline{\mu}_{t,j}, \overline{\Sigma}_{t,j}) > \zeta \tag{7.15}$$

where ζ is a user-defined threshold for associating a mode to the observation. $\mathcal{N}(a|b,c)$ is a Gaussian function over the state space with mean b and covariance c. As presented in Equation 5.70, the value can be computed according to:

$$\mathcal{N}(a|b,c) = \frac{1}{\sqrt{2\pi|c|}^{N_d/2}} exp(-\frac{1}{2}(a-b)c^{-1}(a-b)^T) \tag{7.16}$$

Having found the modes that are associated with observation z_t, the Kalman filter observation update equations are then used to update the linked modes, according to

$$K_{t,j} = \overline{\Sigma}_{t,j}C_t^T(C_t\overline{\Sigma}_{t,j}C_t^T + Q_t)^{-1} \tag{7.17}$$

$$\mu_{t,j} = \overline{\mu}_{t,j} + K_{t,j}(z_t - C_t\overline{\mu}_{t,j}) \tag{7.18}$$

$$\Sigma_{t,j} = (C_t^T Q_t^{-1} C_t + \overline{\Sigma}_{t,j}^{-1})^{-1} \tag{7.19}$$

The weights of all other modes (those not associated with the observation) are reduced to f_P of the

original weights before the observation update, i.e. $w_{t,j} = f_P \overline{w}_{t,j}$. f_P is the probability that the observation is a false positive. The mode statistics of these other modes remain unchanged. The weights of all the modes are subsequently re-normalized.

If instead the observation contains negative information, the agent must still incorporate that information into its belief. We first find the modes that should be associated with observation z_t, according to Equation 7.15. For each of these modes, we truncate the portion of the mode that has close proximity to $\hat{z}_t$, before refitting a new Gaussian around the rest of the mode.

In addition, we compute the new weight $w_{t,j}$ by calculating the relative percentage of the belief that was truncated, according to

$$w_{t,j} = \overline{w}_{t,j}(1 - (1 - f_N) \times \text{frac. truncated}) \tag{7.20}$$

where f_N is the probability that the negative observation obtained was a false negative. The weights are then re-normalized in order to redistribute the fraction of the truncated mode's weight that had been taken away. Finally, modes with low weights are pruned from the agent's belief via a variation of ratio pruning (Schmitt et al., 2002), so as to keep the belief representation tractable.

7.3 Efficient planning under uncertainty with macro-actions

The planning under uncertainty problem requires the agent to plan its next action at every timestep in order to minimize the cost incurred. The cost (or negative reward) can be defined either over the belief space R_B or over the state space R_S. In this chapter, we adopt $R_B(b_t, a)$ as the reward function, since it is more general of the two reward functions (see Section 2.1.1 for a detailed explanation).

7.3.1 Linear Gaussian models, uni-modal Gaussian beliefs (Review)

In Chapter 5, we have shown that when the agent's belief is representable as a uni-modal Gaussian belief, we can directly compute the distribution of posterior beliefs without having to enumerate the possible observations. Specifically, given a prior Gaussian belief (μ_t, Σ_t), the resultant distribution of posterior beliefs can be characterized by a distribution of posterior means and a fixed covariance. We restate the main equations here for convenience

The distribution of means is itself a normal distribution according to

$$\mu_{t+T} \sim \mathcal{N}(f(\mu_{t-1}, \{A, B, a\}_{t-1:t+T-1}), \sum_{\tau=t}^{t+T} \overline{\Sigma}_\tau C_\tau K_\tau^T) \tag{7.21}$$

$$\Rightarrow \mu_{t+T} \sim \mathcal{N}(m_{t+T}, \Sigma_{t+T}^{\mu}) \tag{7.22}$$

where $f(\mu_{t-1}, \{A, B, a\}_{t-1:t+T-1})$ is the deterministic transformation of the means according to $\mu_{t+k} =$

$A_{t+k}\mu_{t+k-1} + B_{t+k}a_{t+k-1}$.

For the covariance update, we can also collapse multi-step covariance updates into a single step. Let $\Sigma_{t-1} = B_{\Sigma,t-1}C_{\Sigma,t-1}^{-1}$. At every step, the complete covariance update can be written as:

$$\Psi_t = \begin{bmatrix} B_\Sigma \\ C_\Sigma \end{bmatrix}_t = \begin{bmatrix} 0 & I \\ I & CQ_tC \end{bmatrix}_t \begin{bmatrix} 0 & A^{-T} \\ A & RA^{-T} \end{bmatrix}_t \begin{bmatrix} B_\Sigma \\ C_\Sigma \end{bmatrix}_{t-1}, \tag{7.23}$$

The covariance update is only dependent on the model parameters of the problem, and is independent of the observations that may be obtained. This property enables us to collapse multi-step covariance updates into a single step $\Psi_{t+1:T} = \prod_{i=1}^{T} \Psi_i$, recovering the posterior covariance Σ_{t+T} from $\Sigma_{t+T} = B_{\Sigma,t+T}C_{\Sigma,t+T}^{-1}$. For a uni-modal Gaussian belief, we therefore have a closed-form method for calculating the distribution of posterior beliefs after the agent executes a macro-action, and can then sample Gaussian parameters from this distribution of distributions to instantiate posterior beliefs for deeper forward search.

7.3.2 Approximations for multi-modal beliefs

For problem domains where multi-modal Gaussian beliefs are needed, we also seek to perform belief updates over a macro-action without having to consider the possible observations that could be obtained. Recall from Equation 7.7 that a multi-modal Gaussian belief can be parameterized with a set of mean-covariance pairs and a set of weights. We would therefore like to compute the joint distribution over the weights and mode statistics, which we will then sample from in order to obtain posterior beliefs for deeper forward search. Two assumptions are necessary for efficient updating of the agent's multi-modal beliefs:

1. A distance-varying covariance function that is a Gaussian function over the state space, so as to approximate the bimodal characteristic of the observation model,

2. The observations are accurate enough that over the span of a macro-action, the the agent will obtain a positive observation at least once if the true state of the world should merit such an observation.

Distance-varying, Gaussian-based observation noise covariance function

A single mode j in a multi-modal Gaussian belief is represented as a Gaussian distribution, $\mathcal{N}(\mu_{t,j}, \Sigma_{t,j})$. As per our problem formulation (Equation 7.2), the agent has a sensor that provides observations that could have either positive or negative information. The possibility of having both noisy position observations *and* a negative observation makes it appear difficult to consider the observation space without branching on the observations.

We propose using a modified noise covariance function to unify the two observation modes. Ignoring false positive and false negative rates for the moment, a positive observation is generated when the hidden state belongs to a particular subset of the state space. This positive observation is perturbed by Gaussian

noise $\delta_t \sim \mathcal{N}(0, Q_t)$. The covariance of the Gaussian noise, Q_t, is then used to calculate the Kalman gain $K_{t,j}$ and update the Gaussian parameters $(\mu_{t,j}, \Sigma_{t,j})$ according to

$$K_{t,j} = \overline{\Sigma}_{t,j} C_t^T (C_t \overline{\Sigma}_{t,j} C_t^T + Q_t)^{-1} \tag{7.24}$$

$$\mu_{t,j} = \overline{\mu}_{t,j} + K_{t,j}(z_t - C_t \overline{\mu}_{t,j}) \tag{7.25}$$

$$\Sigma_{t,j} = (C_t^T Q_t^{-1} C_t + \overline{\Sigma}_{t,j}^{-1})^{-1} \tag{7.26}$$

When the agent's belief has little support within the subset of states that would generate a positive observation, i.e. given the agent's belief, a negative observation is highly likely, then we observe that the belief update according to Algorithm 8 will result in a posterior belief that remains essentially unchanged from the prior belief before the observation update, since little of the belief will be truncated by obtaining a negative observation. We also note that for a traditional Kalman filter update, if the noise covariance is very large, then the Kalman filter equations (Equation 7.24 - 7.26) result in a posterior belief that also remains essentially unchanged.

When a particular mode of the agent's belief has strong support within the subset of states that would generate a positive observation, a negative observation will instead result in the weight of the mode converging towards zero. Under such circumstances, the update of the individual mode with negligible weights is irrelevant. We delay the discussion of weights updating to the next sub-section.

Therefore, we can unify the two observation modes by using an observation noise covariance that varies over the state space. Specifically, we represent the set of observations using the following covariance function:

$$Q_t(s_t) = C_{1,t} - C_{2,t}\mathcal{N}(s_t | C_{3,t}, C_{4,t}) \tag{7.27}$$

$C_{1,t}$ is the noise covariance value when s_t is of a large difference from $C_{3,t}$, which is itself defined over the state space in a similar manner to s_t. In the target-tracking problem, for example, $C_{3,t}$ refers to the agent's own location in the state space, assuming that the agent's state is fully observable. $C_{1,t} - C_{2,t} \geq 0$ determines the noise covariance of the observation when $s_t = C_{3,t}$. Both $C_{1,t}$ and $C_{2,t}$ are assumed to be positive, implying that the noise covariance grows with increasing distance between s_t and $C_{3,t}$. $C_{4,t}$ is a scaling constant that indicates how quickly the noise covariance increases with distance.

Despite the possibility of obtaining negative observations, the Gaussian parameters of each mode j of the agent's belief can therefore be calculated by using a single covariance function (Equation 7.27). Since the observation noise covariance, Q_t, is now a function of the hidden states s_t, we must integrate over the agent's

belief to compute and use the expected noise covariance $\Sigma_{t,z,j} = E[Q_t]$ for each mode

$$\Sigma_{t,z,j} = \int_{s_t} \mathcal{N}(s_t|\overline{\mu}_{t,j}, \overline{\Sigma}_{t,j})(C_{1,t} - C_2\mathcal{N}(s_t|C_{3,t}, C_{4,t})ds_t \tag{7.28}$$

$$= C_1 - C_2 \int_{s_t} \mathcal{N}(s_t|\overline{\mu}_{t,j}, \overline{\Sigma}_{t,j})\mathcal{N}(s_t|C_{3,t}, C_{4,t})ds_t \tag{7.29}$$

We note that in a Normal distribution, for every value of $s_t, C_{3,t}$, $p(s_t|C_{3,t}, C_{4,t}) = p(C_{3,t}|s_t, C_{4,t})$, which implies that the above equation can be written as:

$$\Sigma_{t,z,j} = C_1 - C_2 \int_{s_t} \mathcal{N}(s_t|\overline{\mu}_{t,j}, \overline{\Sigma}_{t,j})\mathcal{N}(C_{3,t}|s_t, C_{4,t})ds_t \tag{7.30}$$

$$= C_1 - C_2\mathcal{N}(s_t|\overline{\mu}_{t,j}, \overline{\Sigma}_{t,j} + C_{4,t}) \tag{7.31}$$

This calculation of $\Sigma_{t,z,j}$ assumes that the belief is known, i.e., there is a deterministic value of the mean and covariance. However, when performing a belief update after a macro-action, we obtain a distribution over the means (Equation 7.21). Assuming $\overline{\mu}_{t,j} \sim \mathcal{N}(m_{t,j}, S_{t,j})$, the noise term can then be calculated according to:

$$\Sigma_{t,z,j} = C_1 - C_2 \int_{s_t} \int_{\overline{\mu}_{t,j}} \mathcal{N}(C_{3,t}|s_t, C_{4,t})\mathcal{N}(s_t|\overline{\mu}_{t,j}, \overline{\Sigma}_{t,j})\mathcal{N}(\overline{\mu}_{t,j}|m_{t,j}, S_{t,j})d\overline{\mu}_{t,j}ds_t \tag{7.32}$$

$$= C_1 - C_2 \int_{s_t} \mathcal{N}(C_{3,t}|s_t, C_{4,t})\mathcal{N}(s_t|m_{t,j}, \overline{\Sigma}_{t,j} + S_{t,j})ds_t \tag{7.33}$$

$$= C_1 - C_2\mathcal{N}(C_{3,t}|m_{t,j}, C_{4,t} + \overline{\Sigma}_{t,j} + S_{t,j}) \tag{7.34}$$

The expected noise from an observation therefore depends not only on the means of the Gaussian mode, but also on the uncertainty. The larger the uncertainty, the more uniformly noisy the observation is expected to be, which then corresponds to a lower expectation of being able to reduce the uncertainty associated with a particular mode.

More importantly, individual modes can now be updated in a consistent, closed-form manner. After executing a macro-action, each mode of the agent's belief can be represented by a distribution of means (Equation 7.21), as well as a fixed covariance (Equation 7.23), with Q_t replaced by $\Sigma_{t,z,j}$.

Updating weights of multi-modal distribution

Having discussed how the individual modes of the agent's multi-modal Gaussian belief can be updated without enumerating the list of possible observations, we now turn to updating the weights of the individual modes that make up the agent's posterior belief. Unfortunately, it is not reasonable for us to update the weights without incorporating the observations, since by the very nature of the belief updating process, the weights will vary drastically depending on whether the agent obtains a positive observation over the course of the macro-action.

In order to reduce the branching factor due to the number of possible observations, we introduce the notion of a "macro-observation" when updating the weights, corresponding to whether negative observations are received when the multi-step action sequence is executed. For each multi-modal belief, we maintain two different sets of weights, W_Y and W_N, corresponding to the two separate scenarios where the agent does and does not receive at least a single positive observation while executing a macro-action.

Recall from Section 7.2 that when a positive observation is obtained, the weights can be updated depending on whether the mode is question is associated with the positive observation. Before re-normalization, the weights of the linked modes are left unchanged, while the modes that are not associated with the observation are reduced to f_P of the original weights, or $w_{t,j} = f_P \overline{w}_{t,j}$. Therefore, when f_P is relatively small, the weights of the non-associated modes will tend to zero, while the entire probability mass will be distributed amongst the linked modes, according to their original weights.

When predicting the possible observations for planning, one option is to sample possible data associations and recover the corresponding weights of the modes. Alternatively, we can leverage the distance-varying covariance function that was described in the previous sub-section to update the posterior weights of all modes directly.

Given the data association process described in Section 7.2, modes are more likely to be associated with the observation z_t if the distance between the mean of the mode $\overline{\mu}_{t,j}$ and the mapped observation onto the state space $\hat{z}_t$ is small. Similarly, the distance-varying noise covariance function $\Sigma_{t,z,j}$ also exhibits the property that the noise covariance of the observation model varies over different parts of the state space, with greater noise covariance as the distance increases between $C_{3,t}$ and s_t.

In addition, it is known that the Fisher information associated with the observation model measures the certainty of a state estimate due to measurement data alone (Maybeck & Siouris, 1980). In the context of a multi-modal Gaussian belief, the Fisher information associated with the observation model for each mode therefore corresponds to the certainty that an observation is associated with a particular mode. By using the distance-varying covariance function when computing the Fisher information, the Fisher information associated with the observation model can also be used to update the weights of each mode, according to

$$w_{t,j} \propto C_t \Sigma_{t,z,j}^{-1} C_t^T \overline{w}_{t,j} \tag{7.35}$$

where $C_t \Sigma_{t,z,j}^{-1} C_t^T$ is the Fisher information associated with the observation model for the Gaussian mode j and C_t is the observation matrix. We then re-normalize the weights in order to obtain W_Y, which corresponds to the set of weights that are obtained when the agent receives at least one positive observation over the course of the macro-action..

On the other hand, when a negative observation is obtained, modes that have support within the subset of the state space that should have generated positive observations should be truncated and a new Gaussian refit around the rest of the mode. As we have presented in Section 7.2, the weights of truncated modes can

be updated according to

$$w_{t,j} = \overline{w}_{t,j}(1 - (1 - f_N) \times \text{frac. truncated}) \tag{7.36}$$

Unfortunately, no closed-form solution exists for performing such a truncation. As a result, we take advantage of the intuition that if negative observations are obtained throughout the course of a macro-action, a mode that would otherwise have generated positive observations will have a posterior weight that tends towards zero if it instead repeatedly obtains negative observations. We therefore compute the set of weights associated with negative observations, W_N, by setting the weights of those modes that would have been truncated to zero, and re-normalize the rest of the weights.

Separately, we calculate the probability α of the agent obtaining a positive observation while executing the macro-action. This probability can be obtained by computing the average percentage of the agent's belief that has support within the field-of-view of the agent's sensor over the course of the macro-action, assuming that the macro-action had been executed in an open-loop fashion. α therefore describes the probability that the agent's multi-modal weights will be distributed according to W_Y, as opposed to W_N, assuming that the agent does not incorporate any of the observations obtained while executing the macro-action. Our distribution over the weights is therefore approximated with two sets of weights W_Y and W_N, as well as the probability α. We then sample from these two sets of weights according to α to instantiate posterior beliefs for deeper forward search.

Given that the series of belief updates associated with a macro-action have been approximated with a macro-observation, it would appear that an alternative approach would be to reduce the graph map structure into a series of discrete cells, with each cell depicting a connected edge and/or node. However, such a simplification of the problem would result in actions with varying time durations, and performing a belief update using an asynchronous discrete-event model is non-trivial, as evidenced by the computationally complex Partially Observable Semi-Markov Decision Processes (POSMDP) (Mahadevan & Khaleeli, 1999) framework.

7.4 Multi-modal posterior belief distribution (MMPBD) algorithm

Algorithm 9 summarizes the Multi-Modal Posterior Belief Distribution (MMPBD) algorithm for efficient macro-action planning in partially-observable domains, and closely follows the macro-action forward search framework (Algorithm 2) presented in Chapter 3.

At every timestep, this forward search algorithm generates the list of macro-actions based on the agent's current state. Each of these macro-actions can be thought of as open-loop policies of varying lengths, independent of the observations that the agent receives while executing the macro-action. In this chapter, we assume that the macro-actions are provided by a domain expert, rather than being generated automatically, as was done in Chapter 4.

Algorithm 9 Multi-Modal Posterior Belief Distribution (MMPBD) algorithm

Require: Agent's initial belief b_0, Macro-action search depth H_f, No. belief samples per macro-action N_z

1: $t \leftarrow 0$
2: **while not** EXECUTIONEND() **do**
3: Generate macro-action list A_{seq}
4: **for** $a_{seq,i} \in A_{seq}$ **do**
5: $Q(a_{seq,i}, b_t) =$ ROLLOUTMACRO$(a_{seq,i}, b_t, H_f, N_z)$
6: **end for**
7: Execute first action $\hat{a}_t$ of $\hat{a}_{seq} = \text{argmax}\, Q(b_t, a_{seq})$
8: Obtain new observation z_t and cost C_t
9: b_{t+1} = BELIEFUPDATE(b_t, a_t, z_t) (Algorithm 8)
10: $t \leftarrow t + 1$
11: **end while**

The planner then evaluates each of the macro-actions, returning the macro-action with the minimum expected cost. The agent then executes the first action of this minimum cost macro-action. The agent updates its belief based on the action taken and observation received via Algorithm 8, and repeats the planning process at the next timestep. For reasons discussed in Section 3.3, the planner replans after every timestep even though the planning process only considers macro-actions, thereby enabling the planner to take into account the latest observation and information available.

Algorithm 10 summarizes the procedure for calculating the expected cost of each macro-action. After computing the probability α of obtaining a positive observation during the macro-action, the distribution of posterior beliefs for both the individual modes and the weights are calculated according to Section 7.3.2.

After calculating the expected immediate cost associated with the macro-action, the agent then samples from the posterior belief distribution to instantiate beliefs for deeper forward search. Given that the covariance of each mode is constant (Equation 7.23), we perform importance sampling only on the posterior mean distribution and the weights (Line 17, 18 of Algorithm 10), and associate them with the posterior covariances to obtain samples of posterior beliefs. These beliefs are then used to perform an additional layer of depth-first search, and this process repeats for a pre-determined search depth H_f (Line 20 - 22).

At the leaf nodes i.e. $h = 0$, a value heuristic is used to provide an estimate of being at the belief node (Ross et al., 2008). Because we perform a forward search out to a fixed macro-action depth, rather than a fixed primitive-action depth, the belief nodes may have expanded out to different timesteps. As opposed to using the value heuristic to estimate the cost to go beyond the planning horizon at each of the leaf nodes, our value heuristic merely seeks to normalize the computed expected values over the number of timesteps. Our value heuristic therefore accumulates the cost of the agent's belief out to a predefined timestep, and for these additional timesteps, the planner assumes that the beliefs are updated by only using the transition update equations (Equations 2-12 of Algorithm 8) with actions that are randomly generated, but without incorporating any observation.

Algorithm 10 ROLLOUTMACRO()

Require: Action sequence a_{seq}, belief state b_t, remaining search depth h, no. belief samples per macro-action N_z

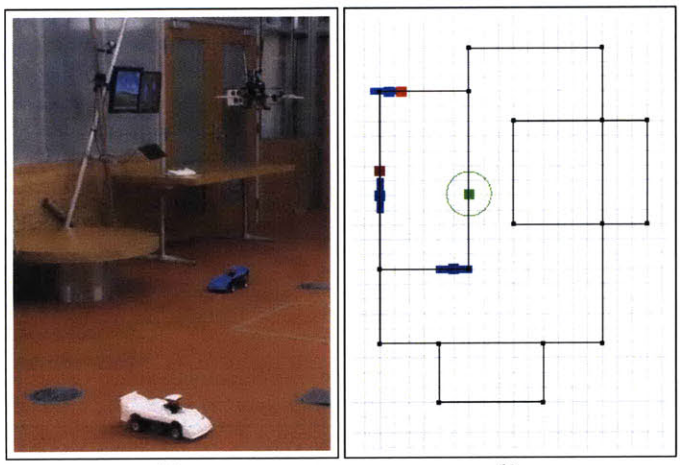
if $h = 0$ **then**
  Perform Kalman filter transition update out to predefined time depth.
  return $\sum_t R_B(b_t, a_{random})$
else {Expand Macro-Action}
  Compute prob. α of obtaining positive observation along a_{seq}
  for each step in $a_k \in a_{seq}, k = \{1, \ldots, |a_{seq}|\}$ **do**
    Collapse modes that are within 1 std dev. apart
    for each mode j in belief b_t **do**
      Compute params $(m_{t+k,j}, \Sigma^{\mu}_{t+k,j}, \Sigma_{t+k,j})$ (Equation 7.22, 7.23)
      Split modes if $\max_i p(S_i|m_{t+k,j}, a_k) < 1$
      Update weights $w_{y,t+k,j}$, $w_{n,t+k,j}$ (Section 7.3.2)
    end for
    $V = V + \alpha R_B(\{W_{Y,t+k}, \Sigma_{t+k}\}, a_k)$
    $V = V + (1-\alpha) R_B(\{W_{N,t+k}, \Sigma_{t+k}\}, a_k)$
  end for
  for $p = 1$ to N_z **do**
    Gen. samples with weights $\{W_Y, W_N\}$ accord. to α
    Sample belief modes from $(m_{t+a_{seq}}, S_{t+a_{seq}}, \Sigma_{t+a_{seq}})$
    Obtain action sequence list A^{next}_{seq}
    for $a^{next}_{seq,p} \in A^{next}_{seq}$ **do**
      $Q(b_p, a^{next}_{seq,p})$ = ROLLOUTMACRO($a^{next}_{seq,p}$,b_p,$h-1$,N_z)
    end for
    $V = V + \gamma \frac{1}{N_z} \text{argmax}_{a^{next}_{seq,p}} Q(b_p, a^{next}_{seq,p})$
  end for
  return V
end if

7.5 Target-tracking

To demonstrate the performance of the MMPBD algorithm, we formalize the road-constrained target-tracking problem, where a helicopter agent (Figure 7-1a) is tasked with maintaining surveillance over multiple targets ($n > 1$) along a road network. In contrast to variations of the target-tracking problem presented in earlier chapters, both the agent and targets are constrained to move along a road network. Therefore, not only are the targets' motion noisy, but also at the road junctions, the targets can choose to move along different road segments, which would result in very different target trajectories. The helicopter agent is also once again constrained to move in a 2D plane, similar to the tracking problem in Chapter 4. However, the reward function in this problem is more similar to the TARGETSURVEY problem in Chapter 6.

A map of the urban environment is known *a priori*, and we assume that the road network can be reduced to a graph with edges and nodes, representing roads and junctions respectively (Figure 7-1b). Both the agent (green square) and the targets (red, brown squares) are constrained to move along the graph, and can move in either direction along an edge (black lines), as well as move along any of the roads that meet at a road junction (black node).

The road-constrained target-tracking problem therefore has a dynamics model that matches the switching



(a) (b)

Figure 7-1: (a) Our quadrotor helicopter tracking multiple ground vehicles. (b) Road-network target-tracking problem. A road network is constructed comprising roads (edges) and junctions (nodes). Both the agent (green) and targets (red/brown) are constrained to travel along the road network. The agent is equipped with a sensor for detecting targets, and receives a noisy positive observation if a target is within the sensor's field-of-view (green circle). Due to partial observability, the agent maintains a belief (blue/dark blue bars) of the targets' poses. If it believes that the target has previously moved past a road junction, it may need to represent its belief the target as a multi-modal Gaussian belief (dark blue bars).

state-space dynamics model presented in Section 7.1.1. The motion of the agent and targets can be assumed to be linear Gaussian along each road segment, whereas at each road junction, the target may choose a new edge with probability $\frac{1}{k-1}$ for a node of degree k, corresponding to the $k-1$ switching states that have non-zero probability. The road network also makes it critical that the agent's belief of each target can be represented as a multi-modal Gaussian belief (blue/dark blue bars), since it is possible that the agent's belief of the target resides on multiple road segments. For example, in Figure 7-1b, the agent believes that one of the targets has previously past a road junction, and therefore the agent's belief of the target's location must be represented as a bi-modal Gaussian belief (dark blue bars). Finally, for computational convenience, we assume that some portion of each Gaussian mode, is allowed to "spill" beyond the ends of each road segment, or alternatively that the road junctions are not exactly at the ends of each road segment. This relaxation allows the planner to avoid having to plan with truncated Gaussians, and can instead restrict the agent's belief to a compact set.

We assume that the targets' poses and dynamics are independent of the agent's, i.e., the target is indifferent or unaware that it is being pursued. In addition, we assume that the agent has knowledge of the targets' mean speed and start poses, and can travel faster than the targets. Even though the agent is aware of the targets' start poses, this target-tracking problem still involves a target-search component because it can subsequently lose track of a target, especially when the modes split at a junction.

The agent is able to accurately localize itself in the environment, but does not have perfect information about the targets' poses. Beyond the sensors used for self-localization, the agent has a limited range,

downward-pointing camera sensor that obtains observations of a target's pose if it is within a certain field-of-view (green circle in Figure 7-1b). Given the height that the helicopter agent is flying at and the intrinsic parameters of the camera sensor, we can recover the effective range r_a of the camera sensor. The agent obtains a noisy observation of a target if the target is within r_a distance from the agent, and a $null$ observation if the target is further than r_a distance away. The observation space is therefore continuous when the agent receives an observation, but also has a bi-modal characteristic due to the $null$ observation. However, some of these observations may be false positives and negatives, depending on the values of f_P and f_N respectively.

Finally, we measure the cost of each target's uncertainty using the weighted sum of the covariances associated with each of the target's Gaussian modes. At each timestep, the agent incurs a cost (or negative reward) according to its belief b_t and the distance traveled (action taken) in the last timestep d

$$R_B(b_t, d) = -\sum_i \sum_j w_{t,i,j}\, tr(\Sigma_{t,i,j}) - \beta d \tag{7.37}$$

where β controls the relative importance of minimizing the distance traveled.

7.6 Simulation experiments

We implemented our MMPBD algorithm on the road-constrained target-tracking problem in several simulated environments, such as those shown in Figure 7-1b. For this set of experiments, the agent only had to track two targets, although the algorithm can be directly extended to more targets. Furthermore, as the focus of this chapter is on the multi-modal nature of the agent's beliefs, we represent each of the modes in the target-tracking problem as a 1D Gaussian, though the algorithm is easily extendable to multi-variate Gaussian representations.

The set of macro-actions available to the agent includes the sequence of actions necessary to travel down an edge to a node, as well as hover at a position for a fixed number of timesteps. Figure 7-2 provides snapshots of a simulation run of the MMPBD algorithm. The problem begins with the agent heading in the general direction of the targets (7-2a, b). Because one of the targets will be arriving at a branching point, the agent hovers at the branching point to observe which outgoing edge is chosen by the target (7-2b). The agent then moves over to localize the other target (7-2c). This behavior shows that the agent can anticipate when modes will split at a junction, and search deep enough to recognize that if it does not localize the target at the junction, it will be harder to localize the targets subsequently. In addition, the agent typically focuses on localizing a single target until the uncertainty of the other targets is large enough that it offsets the cost of traveling to those uncertain targets (7-2d), or if the modes of the other targets are about to split (7-2e). The cost of motion also biases the agent to choose shorter paths (7-2f).

We compared our algorithm to two other strategies — a greedy strategy and a forward search strategy without macro-actions. For the greedy strategy, the agent always travels towards the mode that results in the

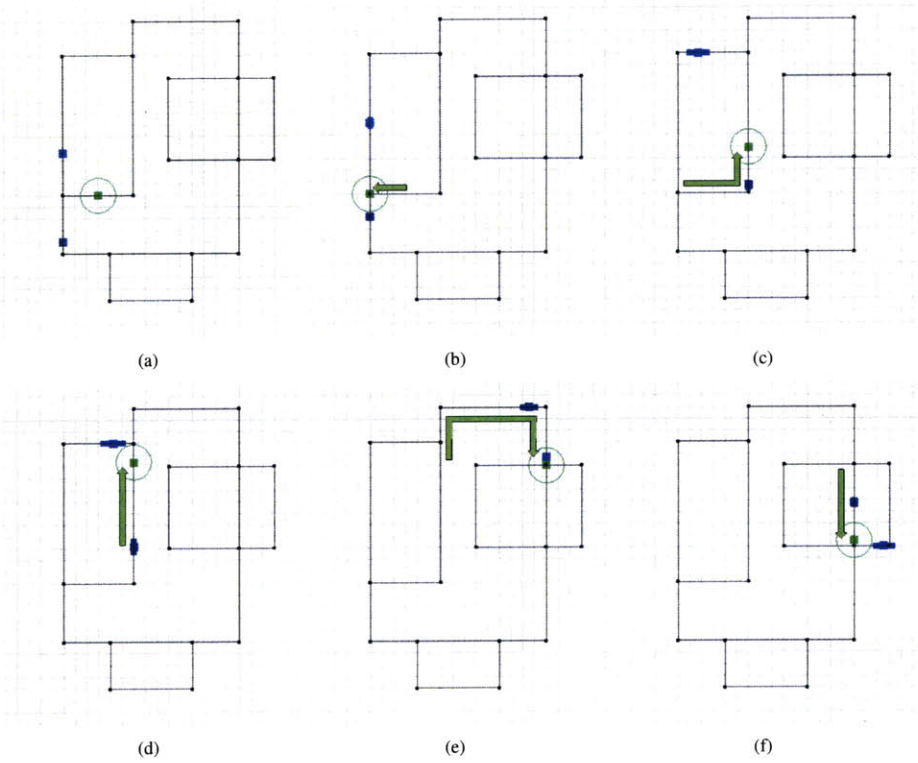


Figure 7-2: Snapshots of a simulation run with the MMPBD algorithm. The agent (green) is able to follow targets (d), anticipate mode splitting (b,e), and intercept targets (f). The green arrows indicate the agent's movement since the previous figure.

smallest negative rewards, which roughly translates to the mode with the largest weighted covariance and a small cost-to-travel. This strategy causes the agent to oscillate between the different targets, but cannot anticipate when modes may split at road junctions.

We also implemented a forward search algorithm that did not have access to macro-actions. Primitive actions are evaluated by performing belief updates according to Algorithm 8. However, the continuous observation space of the target-tracking problem would have made the branching factor too large to be computationally feasible. Instead, for every action that was evaluated, we only branched on whether an observation was obtained or not, thereby giving the naive forward search algorithm access to the macro-observations that were discussed in Section 7.3.2.

Figure 7-3 provides an indication of how the different strategies perform by showing how many modes are needed to represent the agent's belief for a typical run. Initially, all three strategies obtain similar performance, oscillating between the targets to keep them well-localized. However, as the modes approach the road junctions, the greedy strategy does not anticipate when the modes will split. Similarly, the naive forward search strategy cannot search deep enough to realize that it would be harder for the agent to localize the target

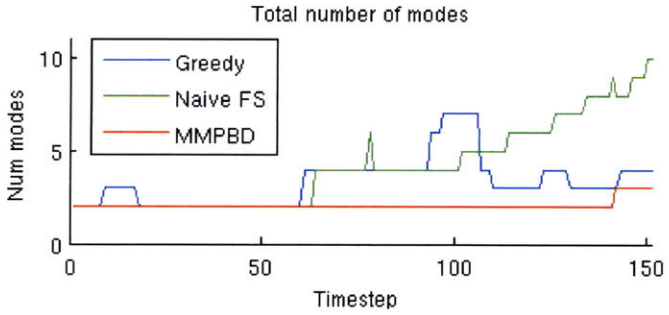


Figure 7-3: Comparison of different strategies over a single run. Typically, the fewer modes in the agent's belief, the better localized the targets.

after the modes have split. The increase in the number of target modes for these benchmark algorithms makes it harder for the agent to reduce the uncertainty at subsequent timesteps.

Table 7.1 summarizes the performance of the three strategies over 10 runs, for 200 timesteps each. Although the MMPBD algorithm results in the longest distance traveled, the agent is better able to localize the targets, ensuring that its belief of each target is uni-modal most of the time. In contrast, the naive forward search algorithm causes the agent to travel the shortest distance because it focuses on trying to localize a single target.

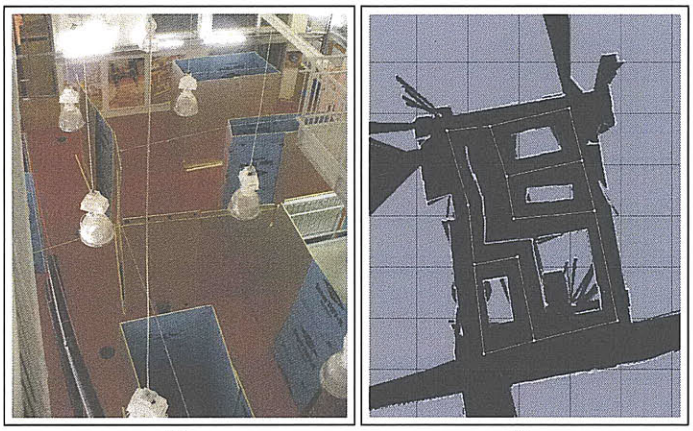
	Dist. Traveled	Ave. Modes	Total Cost
MMPBD	138.76	**1.0810**	**-51.7698**
Greedy	133.52	1.5240	-61.2492
Naive FS	**112.20**	1.7747	-85.1101

Table 7.1: Performance of different planners over 10 runs.

7.7 Real-world experiments

Finally, as a proof of concept, we demonstrated the MMPBD algorithm on the indoor quadrotor helicopter (Figure 7-1a) that we have developed, tasking it to track two ground vehicles. We set up a mock-up of a road network indoors (Figure 7-4a), with two autonomous cars driven at approximately constant speeds in the environment. We made use of a publicly available GMapping algorithm (Grisetti, Stachniss, & Burgard, 2007) to build a map of the environment (Figure 7-4b), before performing a skeletonization of the map to construct the road-network. Since neither target detection, nor data association, is the focus of this thesis, we made the targets easy to detect by using distinct colors for each of the targets, and used a simple blob detection algorithm to detect them.

Figure 7-5 shows snapshots of the helicopter's path during a target-tracking run, as well as the images cap-



(a) (b)

Figure 7-4: (a) Mock-up of road network constructed indoors. (b) Graph structure extracted from SLAM map.

tured from the onboard camera. The helicopter exhibited behaviors similar to those discussed in Section 7.6, oscillating between the different targets to keep them well-localized.

7.8 Related work

The switching state-space model is a popular technique for approximating systems with complex dynamics. The transition dynamics of the discrete switching states are typically either modeled as functions of the previous continuous states (Brunskill et al., 2008) or as Markov Decision Processes (Ghahramani & Hinton, 2000). SSMs have been used for various applications, including planetary rover operation (Blackmore et al., 2007), locomotion over rough terrain (Brunskill et al., 2008) and human motion (Pavlovic et al., 2001). However, most of the work on SSMs has focused on model parameter learning, rather than for planning in partially observable domains. One exception is the work by Brunskill et al. (2008), which proposes a point-based POMDP planning algorithm for solving continuous-state POMDPs using an SSM. However, their algorithm uses a different observation model, and the problem domains experimented on had relatively few states.

Focusing on the specific problem domain of target-tracking, we have presented a brief discussion of the target-tracking algorithms developed by the sensor resource management community in Section 2.4. Nevertheless, a sub-class of problems known as road-constrained target-tracking is often used to describe problems where the agent's belief of the targets' pose could have multiple modes. In cases of multiple-hypothesis tracking of a single target, the belief updating is traditionally done with particle filters (Hue et al. (2002), Agate and Sullivan (2003), Streller (2008)). An exception is the work by Ulmke et al. (2006), which explores both the Gaussian sum approximation and particle filter approaches. However, these algorithms typically focus on

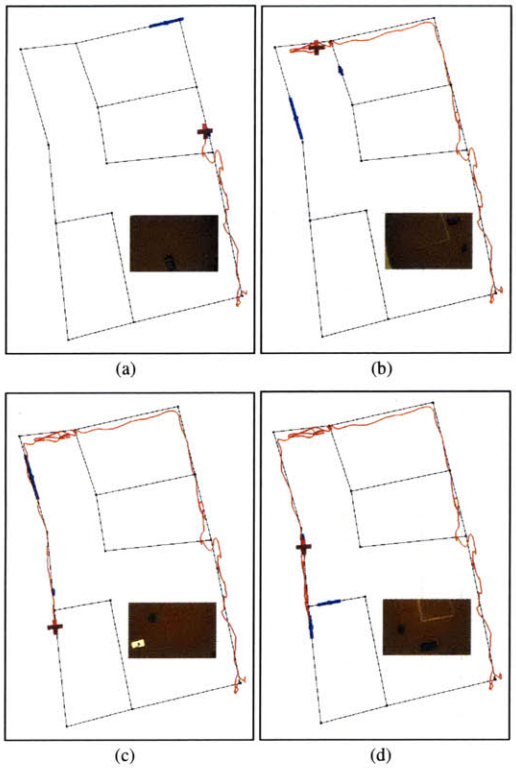


Figure 7-5: Real-world target-tracking demonstration. The path traveled by the helicopter (red/brown) is shown in red. The helicopter oscillates between both target beliefs (blue/dark blue) to localize the targets. Inset: Image captured from downward-pointing camera.

the problem of performing accurate belief updating and data association, rather than on the decision-making for the agent.

Finally, the target-tracking problem has also been addressed by the other planning communities that were mentioned in Chapter 2, such as the POMDP and control communities. Hsu et al. (2008) present a POMDP formulation of the target-tracking problem, and applies SARSOP to generate an offline policy. However, this formulation focuses on tracking a single target. Bertucelli and How (2006a) similarly employ Bayesian techniques for performing search task allocation for multiple vehicles under uncertainty. Finally, Geyer (2008) recently proposed an algorithm that addresses the decision-making problem for the agent by using macro-actions with a particle filter. However, due to the target-search nature of the problem, generating open-loop plans alone is sufficient for finding the target once.

7.9 Conclusion

In this chapter, we demonstrated how the distribution of posterior beliefs can be efficiently calculated when the problem domain has a switching state dynamics model and the agent's belief is represented as a multi-modal Gaussian distribution over the state space. This ability to compute the distribution of posterior beliefs for a more general class of beliefs enables us to perform planning under uncertainty with macro-actions efficiently for a larger class of problem domains.

With this ability to plan efficiently with multi-modal beliefs, we have demonstrated the value of probabilistic planning for tracking targets in an urban environment. By using a multi-modal Gaussian representation of the agent's beliefs, we can plan efficiently by considering multi-step action sequences, and we have demonstrated the performance of our algorithm in both simulation and on actual hardware. Future work includes analyzing the errors induced by the approximations introduced, as well as comparing the algorithms on larger environments with more targets.

Chapter 8

Conclusion

8.1 Summary

We began this thesis with the MAV08 competition scenario, a hostage-rescue mission that required an autonomous MAV to track multiple ground targets and determine which areas were booby-trapped with explosive mines. The MAV could only rely on its local sensors to make observations of the environment, thereby requiring it to plan its future actions in order to maximize the likelihood of mission success.

In this thesis, we developed a number of planning under uncertainty algorithms to enable the MAV agent to accomplish the tasks mentioned above. Specifically, we have presented a new approach for sequential decision-making in stochastic and uncertain domains known as semi-conditional planning. By adopting a forward search framework and conditioning the agent's actions at the end of each temporally-extended, open-loop macro-action, rather than after every action, we have been able to restrict the policy space in exchange for conditional planning out to a longer horizon.

Beyond the general semi-conditional framework presented in Chapter 3, we proposed three algorithms that address two technical challenges for efficient planning under uncertainty with macro-actions — how to automatically generate the macro-actions, and how to efficiently obtain the set of possible posterior beliefs after a macro-action.

Our first algorithm, PUMA, automatically constructs macro-actions and iteratively refines them as computational planning time is made available. These macro-actions are generated automatically by grounding it in an MDP solution. As the optimal macro-action length is not always immediately apparent, our algorithm iteratively refines the macro-action branches as more computational planning time is made available. This refinement process progressively increases the amount of conditional branching in the tree, and in the limit PUMA computes a fully-conditional policy and provides a guarantee on the global optimality of the computed policy. On several large domains requiring far look-ahead, PUMA achieves promising experimental results relative to state-of-the-art offline planners and a semi-conditional planner that uses hand-selected

macro-actions. In addition, PUMA is also able to achieve good performance on domains that require fully-conditional planning.

Our second algorithm, PBD, addresses the second technical challenge of efficiently computing the set of posterior beliefs after a macro-action is executed. Different observation sequences result in different posterior beliefs, and it is computationally expensive to compute the set of posterior beliefs individually. Instead, when the agent's beliefs can be represented as a uni-modal Gaussian distribution, and the transition and observation models are linear Gaussian, we have showed that we can compute an analytic form of the set of posterior beliefs. When the linear Gaussian assumption does not hold, we have shown that an approximate version of the belief update exists for problems with observation models that belong to the exponential family of distributions. Using this analytic computation of the posterior beliefs results in computational efficiencies when planning with macro-actions, and our experimental results demonstrate that our PBD algorithm is computationally cheaper than the generic macro-action approach that samples observation sequences after every macro-action.

Our third algorithm, MMPBD, relaxes the assumptions made by the second algorithm that restricts the agent's beliefs to uni-modal Gaussian beliefs. We motivate this algorithm with a target-tracking problem along a road network, and show that we are still able to efficiently compute the resultant set of posterior beliefs after macro-actions are executed. We have demonstrated the value of probabilistic planning for tracking targets in an urban environment. By using a multi-modal Gaussian representation of the agent's beliefs, we can plan efficiently by considering multi-step action sequences, and we have demonstrated the performance of our algorithm in both simulation and on actual hardware.

Finally, the experimental results that we have been able to demonstrate in these chapters, both in simulation and on actual robotics platforms, demonstrate the applicability of our semi-conditional approach to real world-problems. We have separately demonstrated the ability of our algorithms to solve the individual tasks required of the MAV in the MAV08 competition scenario, including tracking multiple targets in both open spaces and along road networks, as well as detecting the explosiveness of possible mines in an open field. The ability of our algorithms to search out to a long horizon makes this approach extremely valuable for planning in real-world, partially observable problem domains, suggesting that our approach holds significant promise for being applicable on real-world partially observable planning problems in the near future.

It is worth noting that the two classes of problems that we have chosen from the literature, rock sampling for scientific exploration and target-tracking, both happened to generally assume that the agent's position was fully observable throughout the problem, rather than a more generic planning under uncertainty problem where all the states of the world are partially observable. The only exception is our variation of the ISRS problem in Section 3.5, where the agent's state is also partially observable. Regardless, for all our problem domains, the noise in the controller prevents the planner from being able to predict the observable state variables perfectly. Since the macro-actions are generated as sequences of primitive actions that are executed in an open-loop fashion, we believe that the macro-action algorithms that we have proposed are similarly

applicable even when the agent's own state is partially observable.

8.2 Future work

We believe that the semi-conditional planning framework that we have presented in this thesis holds significant promise for overcoming existing challenges in planning under uncertainty in partially observable environments, and can scale up to solve many real-world problems. In this section, we detail a number of possible future research directions:

8.2.1 Automatic generation of macro-actions

For the PUMA algorithm presented in Chapter 4, we currently use standard MDP solution techniques to generate macro-actions, computing a policy over the entire (potentially very large) state space. It would be interesting to investigate a more scalable alternative, using efficient open-loop forward search algorithms like the rapidly-exploring random trees (RRT) to find a feasible path (and corresponding macro-action) between two points in high-dimensional search spaces. RRTs generate admissible trajectories in large, high-dimensional, fully-observable environments very efficiently, but are unable to deal with the uncertainty induced by partial-observability in the environment. Using an RRT to propose candidate macro-actions and evaluating these macro-actions within a forward search framework allows the planner to circumvent this limitation..

In addition, current POMDP approaches generally assume that each time an algorithm is executed, there is no experience that the planner can draw upon to improve its performance. A promising area of research would therefore be to explore whether good macro-actions can be learnt from previous runs on the same problem domain, either using a non-parametric distribution over actions, or using similarity metrics that will cluster and aggregate the macro-actions.

Another interesting issue is deciding the length of the macro-action after it has been refined. We currently fix the length of the macro-action after refinement to be half the original length, but it may be possible to obtain better performance by explicitly considering the best point along the macro-action to perform additional conditional planning. We are also exploring other splitting criteria for deciding the best macro-action to refine next. It remains an open question as to how to balance between expensive, but more accurate full forward search, and the faster, restricted policy class macro-actions.

Similarly, rather than generating possible macro-actions that are of a pre-defined length, it may be possible to determine the length of each macro-action that is to be evaluated, which is equivalent to determining the points where the planner should condition the agent's subsequent actions on the observations received. As we have discussed in Section 3.5, conditioning on the observations obtained is necessary when the optimal policy dictates that different subsequent actions should be executed when different observations are received. An approach for determining when a macro-action should cease for conditional planning to take place is to perform a myopic exploration of the expected immediate reward of taking all possible actions after each

action along the macro-action when different observations are obtained. We can then calculate the difference in expected rewards between executing the same action regardless of the observation that was obtained, and one that conditions the subsequent actions on the received observations. This heuristic may provide an indication of whether it is valuable for the planner to branch on the different observations at particular beliefs.

Finally, we would like to explore the use of closed policies for macro-actions, in a way that can possibly be automatically developed. As we discussed in Section 3.2.1, the options framework for the fully observable planning problem consists of input and termination conditions that are defined over the state space of the problem; a natural extension to partially-observable worlds would be to define the option's input and termination conditions over the belief space.

In the fully-observable scenario, the transition probabilities and expected rewards of each option can be analytically computed, which enables them to be regarded as primitive actions when planning. It is therefore worth exploring whether the transition probabilities, observation probabilities and expected rewards can similarly be analytically computed for the partially-observable variant of options. Alternatively, it may be possible for us to define the initiation and termination conditions as a function of the observation, or sequence of observations, that the agent obtains at the previous timesteps.

8.2.2 Performance bounds and computational analysis

In this thesis, we sought to establish a number of performance bounds and computational complexity analysis for the semi-conditional algorithms presented. An issue worth exploring further is whether it is possible to maintain bounds on the macro-actions in order to selectively expand the tree; existing bounds typically require computing an explicit value function which is very expensive in large state spaces, and we are interested in investigating alternate approaches for bounding the value in high-dimensional, potentially factored, problems.

In addition, although in this thesis we have adopted the forward search approach for planning under uncertainty with macro-actions, it is worth exploring if we can use macro-actions efficiently within the framework of the offline, point-based POMDP approaches, which have been popular within the literature. As discussed in Section 3.6.1, existing offline techniques that employ macro-actions use them to guide the sampling of belief points, but then perform standard value backups on each of the belief points to estimate the value function across the belief space (Theocharous & Kaelbling, 2003; Kurniawati et al., 2009). Instead, by directly incorporating the macro-actions into the value backup process, and by sampling observation sequences of the same length as the macro-actions, an offline planner may be able to perform semi-conditional planning in a dynamic programming fashion and avoid conditioning on every primitive action during the value backup.

8.2.3 Alternative applications for macro-actions

Our experience with the single-target TARGETREPORT problem suggests that in some scenarios, having access to the set of primitive actions is important, at least for a small number of look-ahead steps. One

possible extension to this paper would therefore be to perform a short naïve factored full forward search, and then use the macro-actions as a value heuristic at the leaves. Such an approach would be similar to the class of Receding Horizon Control techniques presented in Section 2.3.

Finally, given our success with applying the semi-conditional planning algorithms on our autonomous quadrotor helicopter, future work includes implementing the algorithms on other autonomous robotic platforms, such as the CSAIL robotic forklift. For instance, the manipulation task of picking up pallets in a partially observable environment closely resembles the simulated ISRS problem domain that we have used in this thesis.

Bibliography

Agate, C., & Sullivan, K. (2003). Road-constrained target tracking and identification using a particle filter. In *Proceedings of the SPIE Conference on Signal and Data Processing of Small Targets*, Vol. 5204.

Andre, D., & Russell, S. (2002). State abstraction for programmable reinforcement learning agents. In *Proceedings of the National Conference on Artificial Intelligence*.

Astrom, K. (1965). Optimal control of Markov decision processes with incomplete state estimation. *Journal of Mathematical Analysis and Applications, 10*, 174–205.

Athans, M. (1971). The role and use of the stochastic linear-quadratic-Gaussian problem in control system design. *IEEE Transactions on Automatic Control, 16*(6), 529–552.

Atrash, A., & Pineau, J. (2006). Efficient planning and tracking in POMDPs with large observation spaces. In *AAAI-06 Workshop on Empirical and Statistical Approaches for Spoken Dialogue Systems*.

Bachrach, A., He, R., & Roy, N. (2009). Autonomous flight in unstructured and unknown indoor environments. In *Proceedings of the European Micro Air Vehicle Conference and Competition*.

Badgwell, T. (1997). Robust model predictive control of stable linear systems. *International Journal of Control, 68*(4), 797–818.

Barndorff-Nielsen, O. (1979). Information and exponential families in statistical theory. *Bull. Amer. Math. Soc. 1, 667-668.*

Bellingham, J., Richards, A., & How, J. (2002). Receding horizon control of autonomous aerial vehicles. In *Proceedings of the American Control Conference*.

Bellman, R. E. (1957). *Dynamic programming*. Princeton University Press, Princeton, NJ, USA.

Bertsekas, D. (2000). *Dynamic Programming and Optimal Control, vol. 1 & 2, 2nd*. Athena Scientific.

Bertsekas, D., & Castanon, D. (1999). Rollout algorithms for stochastic scheduling problems. *Journal of Heuristics, 5*(1), 89–108.

Bertsekas, D., & Tsitsiklis, J. (1991). An analysis of stochastic shortest path problems. *Mathematics of Operations Research, 16*(3), 580–595.

Bertsekas, D., & Tsitsiklis, J. (1996). Neuro-dynamic programming.. Belmont, MA: Athena Scientific.

Bertuccelli, L., & How, J. (2006a). Bayesian forecasting in multi-vehicle search operations. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*.

Bertuccelli, L., & How, J. (2006b). Search for dynamic targets with uncertain probability maps. In *American Control Conference*.

Blackmore, L. (2006). A probabilistic particle control approach to optimal, robust predictive control. In *Proceedings of the AIAA Guidance, Navigation and Control Conference*.

Blackmore, L., Gil, S., Chung, S., & Williams, B. (2007). Model learning for switching linear systems with autonomous mode transitions. In *IEEE Conference on Decision and Control*.

Bonet, B., & Geffner, H. (2002). Solving stochastic shortest-path problems with RTDP. Tech. rep., Universidad Simon Bolivar.

Botea, A., Enzenberger, M., Muller, M., & Schaeffer, J. (2005). Macro-FF: Improving AI planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research, 24*, 581–621.

Boyen, X., & Koller, D. (1998). Tractable inference for complex stochastic processes. In *Proceedings of the Uncertainty in Artificial Intelligence*, Vol. 98.

Bradlow, E. (1996). Teacher's corner: Negative information and the three-parameter logistic model. *Journal of Educational and Behavioral Statistics*, *21*(2), 179.

Brooks, A., Makarenko, A., Williams, S., & Durrant-Whyte, H. (2006). Parametric POMDPs for planning in continuous state spaces. *Robotics and Autonomous Systems*, *54*(11), 887–897.

Brunskill, E., Kaelbling, L., Lozano-Perez, T., & Roy, N. (2008). Continuous-state POMDPs with hybrid dynamics. In *Symposium on Artificial Intelligence and Mathematics*.

Carlin, A., & Zilberstein, S. (2008). Observation compression in DEC-POMDP policy trees. In *AAMAS 2008 Workshop on Multi-Agent Sequential Decision Making in Uncertain Domains*.

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

Cassandra, A., Kaelbling, L., & Kurien, J. (1996). Acting under uncertainty: Discrete Bayesian models for mobile robot navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Charlin, L., Poupart, P., & Shioda, R. (2008). Automated hierarchy discovery for planning in partially observable environments. In *Advances in Neural Information Processing Systems*.

Cheng, H. (1988). *Algorithms for partially observable Markov decision processes*. Ph.D. thesis, University of British Columbia.

Choi, H., & How, J. (2008). Continuous motion planning for information forecast. In *Proceedings of the IEEE Conference on Decision and Control*, pp. 1721–1728.

Darrell, T., & Pentland, A. (1996). Active gesture recognition using partially observable Markov decision processes. In *Proceedings of the 13th International Conference on Pattern Recognition*, Vol. 3.

Dietterich, T. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, *13*(1), 227–303.

Dorato, P., & Vlack, D. (1987). *Robust control*. IEEE Press Piscataway, NJ, USA.

Durbin, J., & Koopman, S. (2000). Time series analysis of non-Gaussian observations based on state space models from both classical and Bayesian perspectives. *Journal of the Royal Statistical Society: Series B (Methodological)*, *62*(1), 3–56.

Dynkin, E. (1965). Controlled random sequences. *Theory of probability and its applications*, *10*, 1.

Foka, A., & Trahanias, P. (2007). Real-time hierarchical pomdps for autonomous robot navigation. *Robotics and Autonomous Systems*, *55*(7), 561–571.

Garcia, C. (1984). Quadratic dynamic matrix control of nonlinear processes: An application to a batch reaction process. In *Proceedings of the AIChE annual meeting*.

Geffner, H., & Bonet, B. (1998). Solving large POMDPs using real time dynamic programming. In *Fall AAAI Symposium on POMDPs*.

Geyer, C. (2008). Active target search from UAVs in urban environments. In *Proceedings of the International Conference on Robotics and Automation*.

Ghahramani, Z., & Hinton, G. (2000). Variational learning for switching state-space models. *Neural Computation*, *12*, 831–864.

Grisetti, G., Stachniss, C., & Burgard, W. (2007). Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, *23*(1), 34–46.

Hansen, E., & Zhou, R. (2003). Synthesis of hierarchical finite-state controllers for POMDPs. In *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling*.

Hansen, E. A., & Feng, Z. (2000). Dynamic programming for POMDPs using a factored state representation. In *Artificial Intelligence Planning Systems*.

Hansen, E., Bernstein, D., & Zilberstein, S. (2004). Dynamic programming for partially observable stochastic games. In *Proceedings of the National Conference on Artificial Intelligence*.

Hauskrecht, M., Meuleau, N., Kaelbling, L., Dean, T., & Boutilier, C. (1998). Hierarchical solution of Markov decision processes using macro-actions. In *Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence*.

He, R., Bachrach, A., Achtelik, M., Geramifard, A., Gurdan, D., Prentice, S., Stumpf, J., & Roy, N. (2010). On the design and use of a micro air vehicle to track and avoid adversaries. *International Journal of Robotics Research, 29*(10), 529–546.

He, R., Prentice, S., & Roy, N. (2008). Planning in information space for a quadrotor helicopter in a GPS-denied environments. In *Proceedings of the International Conference on Robotics and Automation*.

Hernandez-Gardiol, N., & Mahadevan, S. (2001). Hierarchical memory-based reinforcement learning. *In Advances in Neural Information Processing Systems*.

Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association, 58*(301), 13–30.

Hoey, J., & Poupart, P. (2005). Solving POMDPs with continuous or large discrete observation spaces. In *International Joint Conference on Artificial Intelligence*.

Hoey, J., Poupart, P., Boutilier, C., & Mihailidis, A. (2005). POMDP models for assistive technology. In *Proceedings of AAAI Fall Symposium on Caring Machines: AI in Eldercare*.

Hoffmann, J., Spranger, M., Gohring, D., Jungel, M., & Burkhard, H. (2006). Further studies on the use of negative information in mobile robot localization. In *Proceedings of the IEEE International Conference on Robotics and Automation*.

Howard, R. (1971). *Dynamic probabilistic systems: Semi-Markov and decision processes*. New York: John Wiley and Sons, Inc.

Hsiao, K., Lozano-Pérez, T., & Kaelbling, L. (2008). Robust belief-based execution of manipulation programs. In *Proceedings of the Eighth International Workshop on the Algorithmic Foundations of Robotics*.

Hsu, D., Lee, W., & Rong, N. (2008). A point-based POMDP planner for target tracking. In *Proceedings of the International Conference on Robotics and Automation*.

Hue, C., Le Cadre, J., Perez, P., & IRISA, R. (2002). Sequential Monte Carlo methods for multiple target tracking and data fusion. *IEEE Transactions on Signal Processing, 50*(2), 309–325.

Iba, G. (1989). A heuristic approach to the discovery of macro-operators. *Machine Learning, 3*(4), 285–317.

Kaelbling, L., Littman, M., & Cassandra, A. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence, 101*, 99–134.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering, 82*(Series D), 35–45.

Kearns, M., & Koller, D. (1999). Efficient reinforcement learning in factored MDPs. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Kothare, M., Balakrishnan, V., & Morai, M. (1994). Robust constrained model predictive control using linear matrix inequalities. In *American Control Conference*.

Krause, A., & Guestrin, C. (2007). Near-optimal observation selection using submodular functions. In *Proceedings of the 22nd Conference on Artificial Intelligence (AAAI)*.

Kreucher, C., Hero III, A., Kastella, K., & Chang, D. (2004). Efficient methods of non-myopic sensor management for multitarget tracking. In *Proceedings of the 43rd IEEE Conference on Decision and Control*.

Kurniawati, H., Du, Y., Hsu, D., & Lee, W. (2009). Motion planning under uncertainty for robotic tasks with long time horizons. In *Proceedings of the 13th International Symposium of Robotics Research (ISRR)*.

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

Kuwata, Y., & How, J. (2004). Three dimensional receding horizon control for UAVs. In *AIAA Guidance, Navigation, and Control Conference and Exhibit.*

Littman, M., Cassandra, A., & Pack Kaelbling, L. (1995a). Learning policies for partially observable environments: Scaling up. In *Proceedings of the 12th International Conference on Machine Learning.*

Littman, M., Cassandra, A., & Kaelbling, L. (1995b). Learning policies for partially observable environments: Scaling up. *Readings in Agents*, 495–503.

Madani, O., Hanks, S., & Condon, A. (1999). On the undecidability of probabilistic planning and infinite-horizon partially observable Markov decision problems. In *Proccedings of the National Conference on Artificial Intelligence.*

Mahadevan, S., & Khaleeli, N. (1999). Robust mobile robot navigation using partially-observable semi-Markov decision processes. Tech. rep..

Maybeck, P., & Siouris, G. (1980). Stochastic models, estimation, and control, Volume I. *IEEE Transactions on Systems, Man and Cybernetics*, *10*(5), 282–282.

Mayne, D. Q., Rawlings, J. B., Rao, C. V., & Scokaert, P. O. M. (2000). Constrained model predictive control: Stability and optimality. *Automatica*, *36*, 789–814.

McAllester, D., & Singh, S. (1999). Approximate planning for factored POMDPs using belief state simplification. In *Proceedings of the Uncertainty in Artificial Intelligence.*

McGovern, A. (1998). acQuire-macros: An algorithm for automatically learning macro-actions. In *NIPS 98 Workshop on Abstraction and Hierarchy in Reinforcement Learning.*

McGovern, A., & Barto, A. (2001). Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proceedings of the International Conference on Machine Learning.*

Newton, M., Levine, J., Fox, M., & Long, D. (2007). Learning macro-actions for arbitrary planners and domains. In *Proceedings of the International Conference on Automated Planning and Scheduling.*

Paquet, S., Chaib-draa, B., & Ross, S. (2006). Hybrid POMDP Algorithms. In *Proceedings of The Workshop on Multi-Agent Sequential Decision Making in Uncertain Domains (MSDM-2006), Hakodate, Hokkaido, Japan.*

Paquet, S., Tobin, L., & Chaib-draa, B. (2005). An online POMDP algorithm for complex multiagent environments. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 970–977. ACM New York, NY, USA.

Pavlovic, V., Rehg, J., & MacCormick, J. (2001). Learning switching linear models of human motion. *Advances in Neural Information Processing Systems.*

Perkins, T., & Barto, A. (2003). Lyapunov design for safe reinforcement learning. *The Journal of Machine Learning Research*, *3*, 832.

Pineau, J., Gordon, G., & Thrun, S. (2003a). Point-based value iteration: An anytime algorithm for POMDPs. In *Proceedings of the International Joint Conference on Artificial Intelligence.*

Pineau, J., Gordon, G., & Thrun, S. (2003b). Policy-contingent abstraction for robust robot control. In *Proceedings of the 19th Annual Conference on Uncertainty in Artificial Intelligence (UAI).*

Pineau, J., & Thrun, S. (2002). An integrated approach to hierarchy and abstraction for POMDPs. Tech. rep..

Porta, J., Vlassis, N., Spaan, M., & Poupart, P. (2006). Point-based value iteration for continuous POMDPs. *Journal of Machine Learning Research*, *7*, 2329–2367.

Poupart, P. (2005). *Exploiting structure to efficiently solve large scale partially observable Markov decision processes.* Ph.D. thesis, Computer Science department, University of Waterloo.

Precup, D. (2000). *Temporal abstraction in reinforcement learning.* Ph.D. thesis, UMass Amherst.

Prentice, S., & Roy, N. (2007). The belief roadmap: Efficient planning in linear POMDPs by factoring the covariance. In *Proceedings of the 13th International Symposium of Robotics Research (ISRR).*

Puterman, M. (1994). *Markov decision processes: discrete stochastic dynamic programming.* John Wiley & Sons, Inc. New York, NY, USA.

Rawlings, J. (2000). Tutorial overview of model predictive control. *IEEE Control Systems Magazine, 20*(3), 38–52.

Richards, A., & How, J. (2005). Robust model predictive control with imperfect information. In *Proceedings of the American Control Conference.*

Ross, S., & Chaib-draa, B. (2007). AEMS: An anytime online search algorithm for approximate policy refinement in large POMDPs. In *Proceedings of The 20th Joint Conference in Artificial Intelligence (IJCAI).*

Ross, S., Pineau, J., Paquet, S., & Chaib-Draa, B. (2008). Online planning algorihtms for pomdps. *Journal of Articial Intelligence Research, 32*, 663–704.

Roy, N., & Gordon, G. (2003). Exponential family PCA for belief compression in POMDPs. *Advances in Neural Information Processing Systems*, 1667–1674.

Roy, N., Pineau, J., & Thrun, S. (2000). Spoken dialogue management using probabilistic reasoning. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics.*

Roy, N., & Thrun, S. (2000). Coastal navigation with mobile robots. In *Advances in Neural Information Processing Systems (NIPS).*

Schmitt, T., Beetz, M., Hanek, R., Buck, S., et al. (2002). Watch their moves: Applying probabilistic multiple object tracking to autonomous robot soccer. In *Proceedings of the National Conference on Artificial Intelligence.*

Scott, A., Harris, Z., & Chong, E. (2009). A POMDP framework for coordinated guidance of autonomous UAVs for multitarget tracking. *EURASIP Journal on Advances in Signal Processing, 2009.*

Shani, G., Brafman, R. I., & Shimony, S. E. (2007). Forward search value iteration for pomdps. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI).*

Sim, H. S., Kim, K.-E., Kim, J. H., Chang, D.-S., & Koo, M.-W. (2008). Symbolic heuristic search value iteration for factored pomdps. In *Proceedings of the Association for the Advancement of Articial Intelligence.*

Smallwood, R., & Sondik, E. (1973). The optimal control of partially observable Markov processes over a finite horizon. *Operations Research, 21*(5), 1071–1088.

Smith, T., & Simmons, R. (2004). Heuristic search value iteration for POMDPs. In *Proceedings of the Uncertainty in Artificial Intelligence.*

Smith, T., & Simmons, R. (2005). Point-based POMDP algorithms: Improved analysis and implementation. In *Proceedings of the conference on Uncertainty in Artificial Intelligence.*

Sondik, E. (1971). The optimal control of partially observable Markov processes.. *DTIC Research Report AD0730503.*

Sorenson, H., & Alspach, D. (1971). Recursive Bayesian estimation using Gaussian sums.. *Automatica, 7*, 465.

Spaan, M., & Vlassis, N. (2005). Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research, 24*, 195–220.

Stengel, R., & Ryan, L. (1991). Stochastic robustness of linear time-invariant control systems. *IEEE Transactions on Automatic Control, 36*(1), 82–87.

Stolle, M., & Precup, D. (2002). Learning options in reinforcement learning. *Lecture Notes in Computer Science*, 212–223.

Streller, D. (2008). Road map assisted ground target tracking. In *Proceedings of the 11th International Conference on Information Fusion.*

Sutton, R., & Barto, A. (1999). Reinforcement learning. *Journal of Cognitive Neuroscience, 11*(1), 126–134.

Sutton, R., Precup, D., & Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence, 112*(1), 181–211.

Theocharous, G. (2002). *Hierarchical learning and planning in partially observable markov decision processes*. Ph.D. thesis, Michigan State University.

Theocharous, G., & Kaelbling, L. (2003). Approximate planning in POMDPs with macro-actions. *Advances in Neural Information Processing Systems 16 (NIPS)*.

Theocharous, G., Rohanimanesh, K., & Mahadevan, S. (2001). Learning hierarchical partially observable Markov decision process models for robot navigation. In *Proceedings of the International Conference on Robotics and Automation*.

Triantafyllopoulos, K. (2003). On the central moments of the multidimensional Gaussian distribution. *The Mathematical Scientist, 28*, 125–128.

Ulmke, M., Koch, W., & Fgan-Fkie, W. (2006). Road-map assisted ground moving target tracking. *IEEE Transactions on Aerospace and Electronic Systems, 42*(4), 1264–1274.

Washington, R. (1997). BI-POMDP: Bounded, incremental partially-observable Markovmodel planning. In *Proceedings of the 4th European Conference on Planning (ECP)*.

West, M., Harrison, P., & Migon, H. (1985). Dynamic generalized linear models and Bayesian forecasting. *Journal of the American Statistical Association*, 73–83.

Wiering, M., & Schmidhuber, J. (1997). HQ-learning. *Adaptive Behavior, 6*(2), 219–246.

Yu, C., Chuang, J., Computing, S., Math, C., Gerkey, B., Gordon, G., & Ng, A. (2005). Open-loop plans in multi-robot POMDPs. Tech. rep., Stanford CS Department.

Zhang, N. L., & Zhang, W. (2001). Speeding up the convergence of value iteration in partially observable Markov decision processes. *Journal of Artcial Intelligence Research, 14*, 1–28.

Zhou, E., Fu, M., & Marcus, S. (2008). A density projection approach to dimension reduction for continuous-state POMDPs. In *Proceedings of the 47th IEEE Conference on Decision and Control*.