Exploiting Structure to Efficiently Solve Large Scale Partially Observable Markov Decision Processes

by

Pascal Poupart

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Abstract

Exploiting Structure to Efficiently Solve Large Scale Partially Observable Markov Decision Processes

Pascal Poupart
Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto
2005

Partially observable Markov decision processes (POMDPs) provide a natural and principled framework to model a wide range of sequential decision making problems under uncertainty. To date, the use of POMDPs in real-world problems has been limited by the poor scalability of existing solution algorithms, which can only solve problems with up to ten thousand states. In fact, the complexity of finding an optimal policy for a finite-horizon discrete POMDP is PSPACE-complete. In practice, two important sources of intractability plague most solution algorithms: large policy spaces and large state spaces.

On the other hand, for many real-world POMDPs it is possible to define effective policies with simple rules of thumb. This suggests that we may be able to find *small* policies that are *near optimal*. This thesis first presents a *Bounded Policy Iteration* (BPI) algorithm to robustly find a good policy represented by a small finite state controller. Real-world POMDPs also tend to exhibit structural properties that can be exploited to mitigate the effect of large state spaces. To that effect, a *value-directed compression* (VDC) technique is also presented to reduce POMDP models to lower dimensional representations.

In practice, it is critical to *simultaneously* mitigate the impact of complex policy representations and large state spaces. Hence, this thesis describes three approaches that combine techniques capable of dealing with each source of intractability: VDC with BPI, VDC with Perseus (a randomized point-based value iteration algorithm by Spaan and Vlassis [136]), and state abstraction with Perseus. The scalability of those approaches is demonstrated on two problems with more than 33 million states: synthetic network management and a real-world system designed to assist elderly persons with cognitive deficiencies to carry out simple daily tasks such as hand-washing. This represents an important step towards the deployment of POMDP techniques in ever larger, real-world, sequential decision making problems.

Acknowledgements

I would like to acknowledge the contributions of many people who have helped me directly and indirectly in the accomplishment of this work. First and foremost, I am indebted to my supervisor and mentor, Craig Boutilier, who has played a key role by his continuous support, his invaluable guidance and his insightful advice for my research as well as my career.

I am also grateful to Jesse Hoey for carefully reading and commenting an early draft of this thesis. I also thank my external examiner, Michael Littman and the members of my thesis committee, Richard Zemel, Fahiem Bacchus and Christina Christara for their helpful comments. More generally, I thank the department of Computer Science at the University of Toronto for the stimulating environment to learn and carry out research.

Finally, my special thanks go to my wife, Ching-Huei Wu, for her patience and continuous support. I also thank my parents, Francine and Real Poupart, for their encouragement and moral support as well as Heather Watson and Peter Josic for providing me with a quiet and stable environment in their home.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The design of automated systems capable of accomplishing complicated tasks is at the heart of computer science. Such systems can be viewed abstractly as taking inputs from the environment and producing outputs toward the realization of some goals. An important problem is the design of good control policies that produce suitable outputs based on the inputs received. For instance, a thermostat is an automated system that regulates the temperature of a room by controlling a heating device based on information provided by heat sensors. For such a simple system, a reactive control policy can maintain the temperature more or less constant by turning on and off the heating device when the temperature is below or above some target.

For more complicated systems, effective control policies are often much harder to design. Consider a system designed to assist elderly persons suffering from memory deficiencies. Memory loss can severely hamper the ability of a person to accomplish simple activities of daily living such as dressing, toileting, eating, taking medication, etc. An automated system could help a person regain some autonomy by guiding a person with some audio-prompts that remind the person of the next steps in the course of an activity. Suppose the system is equipped with sensors (e.g., video-cameras and microphones) to monitor the user, and actuators (e.g., speakers) to communicate with the user. The design of a suitable prompting strategy is far from obvious.

In particular, the information provided by the sensors tends to be inaccurate due to the noisy nature of image and sound processing. Furthermore, that information may be incomplete due to the limited scope of the sensors. For example, although cameras and microphones allow the system to observe movements and utterances made by a user, they do not reveal the intentions nor the state of mind of people. Ideally, if the system could read minds, the design of effective prompting strategies could be eased significantly. Instead, the system must infer the state of the user based on the limited and noisy information provided by sensors.

The effects of actuators may also be quite uncertain. For example, users may not always follow the prompts depending on their mood, their physical or mental weariness, etc. The system should then have the ability to take into account this uncertainty in its strategy. When uncertainty in the action effects is due to unknown features of the world, it may be possible to reduce that uncertainty by designing strategies that learn and adapt to the environment as they proceed. For instance, each user may exhibit a different personality for which different prompting strategies are better suited. Initially the system may not know the personality of a user; however, as interaction proceeds, it should be able to learn about the user and tailor its prompting strategy accordingly.

This also suggests that actions may have long term effects, which further complicates the design of control policies. For instance, in order to learn about a user, the system may sometimes give a prompt that does not help achieve immediate progress, but that allows it to gain useful information by observing how the user reacts. This information can then be used to better tailor the policy in the future. Hence, the control policy of automated systems should also be able to reason about the sequentiality of actions, including their short-term and long-term effects, by balancing the exploitation of available information for short-term progress with the exploration of new information for long-term progress.

The design of control policies is also complicated by the fact that automated systems often pursue multiple (possibly conflicting) objectives. The primary goal of an automated assistant is to help the user to complete tasks as often and as quickly as possible. The system may then be tempted to prompt at each step regardless of the user's state just to make sure that she stays on course. However, such a strategy may irritate the user. Thus, we would like a system that effectively guides a user while prompting only when necessary to retain as much user autonomy as possible. In general, control policies should be able to tradeoff conflicting objectives based on their relative importance.

1.1 Decision-Theoretic Planning

Partially observable Markov decision processes (POMDPs) provide a natural framework for modeling complex control problems with partial observability, uncertain action effects, incomplete knowledge of the environment dynamics and multiple interacting objectives.

Uncertainty in the action effects and the state of the world are modeled probabilistically. Objectives are encoded with utility functions whose magnitude is indicative of their relative importance. Following the principle of maximum expected utility, it is possible to optimize a control strategy to achieve the desired objectives. Furthermore, optimal policies naturally learn about unknown components of the environment dynamics by optimizing the combined value of the information gained by exploration with the utility of immediate action effects.

POMDPs were initially formalized by the control theory and operations research communities [32, 2, 81, 134, 90, 26, 74] to optimally control stochastic dynamical systems. More recently, the artificial intelligence community also considered POMDPs for planning under uncertainty [22, 55]. To date, a wide range of sequential decision problems such as robot navigation [67, 21, 140, 91], preference elicitation [10], stochastic resource allocation [87, 80], maintenance scheduling [116], spoken-dialog systems [98, 145, 144] and many others have been modeled using POMDPs.

Despite the considerable expressivity of POMDPs, their use in real-world systems remains limited due to the intractability of the solution algorithms for finding good control policies. In fact, the problem of computing an optimal policy is PSPACE-complete [99] and an ϵ-optimal policy is NP-hard [75] for discrete POMDPs. In practice, the hardness of POMDPs arises from the complexity of policy spaces and the potentially large number of states.

Nervertheless, real-world POMDPs tend to exhibit a significant amount of structure, which can often be exploited to improve the scalability of solution algorithms. In particular, many POMDPs have simple policies of high quality. Hence, it is often possible to quickly find those policies by restricting the search to some class of compactly representable policies. In addition, when states correspond to the joint instantiation of some random variables (features), it is often possible to exploit various forms of probabilistic independence (e.g., conditional independence and context-specific independence), decomposability (e.g., additive separability) and sparsity in the POMDP dynamics to mitigate the impact of large state spaces. Hence, this thesis focuses on solution algorithms for discrete POMDPs that exploit problem-specific structure to achieve better scalability.

1.2 Contributions

To date, several algorithms have been proposed to search in restricted classes of policies, thereby circumventing the complexity of policy spaces. When there is no expert knowledge available to determine a suitable class, a popular approach consists of searching for the best policy represented by a *finite state controller* of some limited size. Policy iteration (PI) [44, 45], gradient ascent (GA) [88, 1], branch and bound (B&B) [89] and stochastic local search (SLS) [20] can be used to perform this search; however, the size of the controllers constructed by PI tends to grow exponentially, GA may get trapped in local optima, and the running times of B&B and SLS tend to scale poorly. Chapter 3 describes a new anytime algorithm called *bounded policy iteration* (BPI) that combines some of the advantages of GA (efficiency) and PI (less vulnerability to local optima). Beyond the contribution of a new robust scalable algorithm, useful insights are provided concerning some of the limitations of PI and GA. In particular, an analysis of the local optima that may trap GA is carried out. More precisely, sufficient *tangency* conditions characterizing those local optima are described. We also explain how to extend PI to stochastic controllers (previously limited to deterministic controllers), which allows one to find smaller controllers, yet with higher value, than PI with deterministic controllers.

Chapter 4 describes a new *value-directed compression* (VDC) technique to circumvent the complexity of large state spaces. The idea is to compress POMDP dynamics to a more compact representation by retaining only the state information necessary to the decision process. Since an optimal policy can be found as long as the utility of each policy can be accurately estimated, state information that doesn't help in the evaluation of policies is discarded. General sufficient conditions are derived to ensure a lossless compression. When considering the class of *linear* compressions, interesting mathematical properties of those sufficient conditions are highlighted and used to derive a simple *Krylov iteration* algorithm that finds the best linear lossless compression. In the event where the best lossless compression doesn't yield a small enough representation, two other algorithms are proposed to find good lossy linear compressions. Compared to other techniques that overcome the complexity of large state spaces, linear VDC has the advantage that the resulting compressed POMDPs can be solved (more or less) directly by *any* existing POMDP algorithm. Similarities and differences between several related compression/model minimization techniques are also discussed. Very briefly (see Section 4.2.3 for more details), it turns out that VDC subsumes *state aggregation* [14], computes a *stochastic bi-simulation* [38], produces compressed POMDPs that are *predictive state representations* [71], but exploits different structure than compressions by *exponential principal component analysis* [119].

Currently, state-of-the-art algorithms that do not exploit any problem structure suffer from both sources of intractability (i.e., complex policy space and large state space) and therefore can only solve toy POMDPs with roughly 1000 states and optimal policy graphs of roughly 1000 nodes. Algorithms that restrict their search to classes of compactly representable policies can tackle slightly more difficult problems, but are still plagued by the complexity of large state spaces. Similarly, algorithms that exploit problem structure to mitigate the impact of large state spaces, but do not tackle the complexity of policy space, cannot solve problems significantly larger. In order to tackle large scale POMDPs, it is necessary to simultaneously tackle both sources of intractability. Hence, the most significant contribution of this thesis is the description in Chapter 5 of three new algorithms that each combine existing ideas to overcome both sources of intractability. In particular, a compressed version of BPI is described by integrating BPI with VDC. Compressed and symbolic versions of the point-based value iteration algorithm called Perseus (by Spaan and Vlassis [142, 137, 136]) are also described by integrating Perseus with VDC and a popular symbolic representation called algebraic decision diagrams (ADDs). The scalability of those algorithms is demonstrated on synthetic network management problems and an assistive technology task of up to 33 million states with unknown optimal policies that are believed to be complex. Finally, the last contribution of this thesis is the description of a POMDP model for an automated system designed to assist persons with dementia.

1.3 Outline

The thesis is structured as follows. Chapter 2 introduces the POMDP framework and explains in detail its advantages for formalizing planning under uncertainty. Related decision-theoretic models and classical planning approaches are compared to help situate POMDPs in the literature. Then, several classic solution algorithms are reviewed to better understand the two sources of intractability that pervasively plague POMDP algorithms.

Chapter 3 focuses on algorithms that tackle the complexity of policy space. First, a literature review of existing algorithms is presented, noting that the complexity of policy spaces can often be circumvented by restricting the search for a good policy to compactly representable policies and by tailoring policies to the reachable belief region. Then, the bounded policy iteration algorithm is proposed as a robust and efficient alternative to existing algorithms for computing good finite state controllers. The robustness of BPI to local optima is verified on some problems known to possess important local optima and its scalability is compared to other algorithms on a suite of benchmark problems.

Chapter 4 focuses on algorithms that tackle the complexity arising from large state spaces. A literature review of the various types of problem structure and the algorithms that exploit them is presented. Then, a new value-directed compression technique is presented. It simultaneously exploits additive separability, conditional independence and context-specific independence to generate a compressed model solvable by most POMDP algorithms. The effectiveness of lossless and lossy compressions are verified on several factored POMDPs.

Chapter 5 combines several algorithms of Chapters 3 and 4 to simultaneously tackle both sources of intractability. A compressed version of bounded policy iteration is described by combining BPI with VDC. Also, compressed and symbolic versions of Perseus are described by combining Perseus with VDC and ADDs. The scalability of those algorithms is tested on synthetic network management problems and an assistive technology task with up to 33 million states. The assistive technology task consists of an automated system designed to help persons with dementia to wash their hands.

Finally, Chapter 6 concludes by summarizing the thesis and presenting some open problems for future work.

Chapter 2

Partially Observable Markov Decision Processes

Partially observable Markov decision processes (POMDPs) were first introduced in the control theory and operations research communities [32, 2, 134, 90, 26, 74] as a framework to model stochastic dynamical systems and to make optimal decisions. This framework was later considered by the artificial intelligence community as a principled approach to planning under uncertainty [22, 55]. Compared to other methods, POMDPs have the advantage of a well-founded theory. They can be viewed as a special (continuous) case of the well-known fully observable Markov decision process (MDP) model, which is rooted in probability theory, decision theory and utility theory.

2.1 Model Description

POMDPs provide a rich framework to model uncertainty in a planning problem. They allow action effects and state observations to be modeled probabilistically. They also allow implicit learning of the model components when they are not completely known [34].

2.1.1 Components

Formally, a POMDP is specified by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{Z}, T, Z, R, h, \gamma \rangle$. We now describe each one of those 8 components. In practice, each component must be specified by a domain expert or learned from data.

State space $\mathcal{S}$

The world is modeled by a set $\mathcal{S}$ of distinct states. The number of states may be finite, countably infinite or continuous. Unless otherwise stated, we will focus on discrete models with a finite number of states.

Action space $\mathcal{A}$

An agent living in this world seeks to influence its state by executing actions from the set $\mathcal{A}$. Again, $\mathcal{A}$ can be finite, infinite or continuous, but we will assume that it is finite unless otherwise stated. Roughly speaking, the agent's goal is to choose actions that will influence the world in such a way that desirable states are visited more frequently.

Transition function T

As opposed to classical planning models, POMDPs allow action effects with uncertainty to be modeled. From the agent's point of view, this means that the world has a certain probability of making a transition to any state in $\mathcal{S}$ as a result of an action execution. The stochastic nature of action effects is captured by the transition function T. Let $T(s, a, s') = Pr(s'|s, a)$ denote the probability that the world makes a transition to state s' when action a is executed in state s. Note that this transition function exhibits the Markov property, which says that the probability of transition to some state s_{t+1} at the next time-step $t+1$ depends only on the state s_t and the action a_t at the current time-step t. It is independent of the previous states and actions.

Observation space $\mathcal{Z}$

After executing an action, the agent makes an observation $z \in \mathcal{Z}$. Observations correspond to features of the world directly perceptible by an agent's sensors. In contrast, states correspond to all the relevant features of the world, but these may not be perceptible by the agent.

Observation function Z

Observations provide information about the current state of the world. More precisely, observations are probabilistically related to states by the observation function Z. Let

$Z(a, s', z') = \Pr(z'|a, s')$ denote the probability that the agent experiences observation z' after executing action a and making a transition to state s'.

Note that in fully observable MDPs $\mathcal{Z} = \mathcal{S}$ as the agent knows exactly the current state. In this case the observation function corresponds to an identity function (i.e., $\Pr(z' = s'|a, s') = 1$ and $\Pr(z' \neq s'|a, s') = 0$). In partially observable MDPs, which is the focus of this thesis, $\mathcal{Z}$ may differ from $\mathcal{S}$. Observations provide only partial information to the agent since the same observation may be experienced in different states.

Reward function R

The preferences of the agent are encoded in the reward function R. This function indicates how much utility $R(s, a)$ is earned by an agent when the world is in state s and it executes some action a. The reward function is a powerful tool since it allows simple goals as well as complex concurrent goals to be modeled. The key to modeling concurrent goals is the use of utility theory, which provides a common scale that allows an agent to combine multiple goals and to make rational tradeoffs with respect to those goals.

Horizon h and discount factor γ

In decision theory, the goal of an agent is to maximize the expected utility earned over some time frame. The horizon h defines this time frame by specifying the number of time-steps the agent must plan for. The horizon can be finite or infinite. A discount factor γ is also used to indicate how rewards earned at different time-steps should be weighted. In general, the more delayed a reward is, the smaller will be its weight. Therefore, γ is a constant in $[0, 1]$ indicating by how much a reward should be scaled down for every time-step delay. A reward earned k steps in the future is scaled down by γ^k. Unless otherwise indicated, this thesis assumes infinite horizon POMDPs with a discount factor strictly less than 1.

Unknown parameters

The specification of a POMDP requires all of the above components. In practice, components such as the transition, observation and reward functions are often difficult to define accurately due to incomplete prior knowledge. Therefore, one must often resort to learning techniques. One can either conduct offline experiments to learn the unknown parameters in a preprocessing step, or alternatively, one can implicitly learn the unknown parameters while planning. The latter approach is quite interesting since the problem of simultaneously learning and planning can be formulated as a larger POMDP, for which all the components are known [34]. Since states are not directly observable and therefore possibly unknown, we can augment the state space with the domain of the unknown parameters. As an agent interacts with the world, it will make some observations that help it estimate the state of the world as well as the unknown parameters. Hence, the agent can learn about unknown parameters while executing a policy.

Note also that the POMDP formulation considered in this thesis assumes *stationary* components. That is, $\mathcal{S}$, $\mathcal{Z}$, $\mathcal{A}$, T, Z and R do not change over time. On the other hand, it is possible to transform non-stationary processes into stationary ones, by including in $\mathcal{S}$, $\mathcal{Z}$, $\mathcal{A}$ all the states, observations and actions possible at any time-step. If some states or observations are not possible at some time-step, they will simply have zero probability of occurring in the transition and observation functions. Similarly, if some actions are not possible at some time-step, we simply have to encode an infinitely negative reward to prevent an optimal agent from selecting them. When T, Z and R change over time, it may be due to some unknown parameters of those functions. As mentioned above, one can then augment the state space with those unknown parameters, making the process stationary and allowing the agent to learn them while planning.

By the nature of partially observable Markov decision processes, the transition, observation and reward functions are also assumed to be *Markovian*, that is, they only depend on the current state and action (not the past states and actions). When a non-Markovian process has dependencies on the past states and actions, it is also possible to transform it into a larger Markovian process by augmenting the state space to include all relevant past states and actions.

Without loss of generality, the focus of this thesis will be on planning algorithms for fully specified, stationary and Markovian processes. When some components are unknown, non-stationary or non-Markovian, we will assume that the larger POMDP resulting from the inclusion of all necessary parameters in the state space is used.

2.1.2 Policies

Given a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{Z}, T, Z, R, h, \gamma \rangle$ specifying a POMDP, what action should an agent execute at each time-step to earn as much reward as possible over time? Let's define Π to be the set of all *policies* π (action strategies) that an agent can execute. Roughly

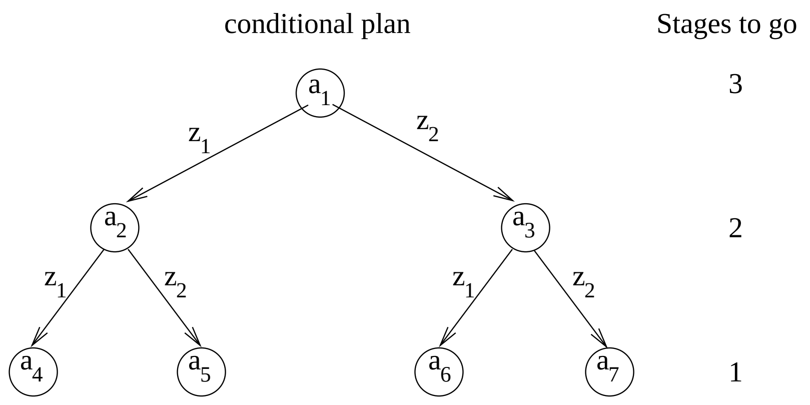


Figure 2.1: Tree representation of a three-step conditional plan.

speaking, a policy is some strategy that dictates which action a to execute (at each time-step) based on some information previously gathered. The relevant information available to the agent consists of some belief b_0 about the initial state of the world and the history (sequence) of actions and observations experienced up to the current time-step t ($hist_t = \langle a_0, z_1, a_1, z_2, \ldots, a_{t-1}, z_t \rangle$). Since the agent may not have complete knowledge of the initial state of the world, we use b_0 to denote a probability distribution over all possible states that corresponds to its belief about the initial state. Hence, a policy π is a mapping from initial beliefs and histories to actions.

Representations

For a given initial belief state b_0, a finite policy π can be represented by a tree corresponding to a conditional plan β. Figure 2.1 shows such a tree for a three-step conditional plan. Intuitively, a conditional plan is defined as a mapping from histories to actions. The execution of a conditional plan consists of the traversal of its corresponding tree from the root to a leaf by interleaving action execution and observation gathering. For instance, in Figure 2.1, action a_1 is first executed. Suppose observation z_2 is received, then action a_3 is executed. If observation z_1 is received next, then action a_6 is executed.

We can define recursively a k-step conditional plan β_k in terms of $(k-1)$-step conditional plans β_{k-1}. The idea is to define $\beta_k = \langle a, \sigma_k \rangle$ as a tuple consisting of an action a and an observation strategy σ_k such that $\sigma_k : \mathcal{Z} \rightarrow \Gamma_{k-1}$ is a mapping from observations to conditional plans of length $k-1$ (Γ_{k-1} is the set of all $(k-1)$-step conditional plans). Figure 2.2 shows the recursive definition of the conditional plan in Figure 2.1.

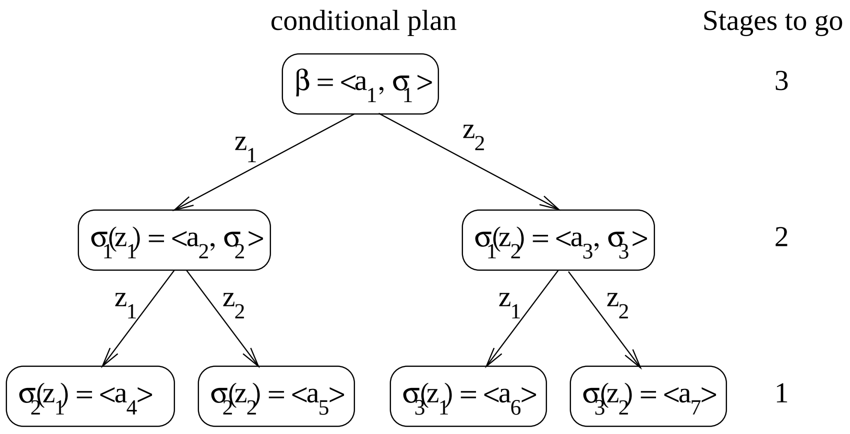


Figure 2.2: Recursive definition of a conditional plan.

Note that one-step conditional plans correspond to an action only (i.e., $\Gamma_1 = \mathcal{A}$) since an observation strategy is unnecessary.

Unfortunately, as the number of steps increases, the number of histories grows exponentially and it is infeasible to represent mappings over all such histories. Furthermore, infinite-horizon problems require mappings over arbitrarily long histories, which limits the use of conditional plans to problems with a short horizon. Note, however, that it is possible to have mappings over infinite *cyclic* histories. Such mappings can be represented by a *finite state controller* [44], which is essentially a set of cyclic conditional plans. In Figure 2.3, each node is the root of a cyclic conditional plan that is executed in the same way as the finite conditional plans previously described. For example, suppose that execution starts in the top node of the controller in Figure 2.3, then action a_1 is executed. Then, if observation z_2 is made, the middle node is reached and action a_2 is executed. If observation z_1 is received next, then the bottom left node is reached and action a_1 is executed. Execution continues by following edges labeled with the observations made and executing the actions of the nodes traversed.

Alternatively, it is possible to summarize histories by a sufficient statistic that encodes all the relevant information from previous actions and observations for planning purposes. Recall that the transition, reward and observation functions exhibit the Markov property, which means that the outcome of future states, rewards and observations depend only on the current state and action. If the agent knew the current state of the world, then it would have all the desired information to make an optimal action choice. Thus, histories of past actions and observations are only relevant to the extent that they provide infor-

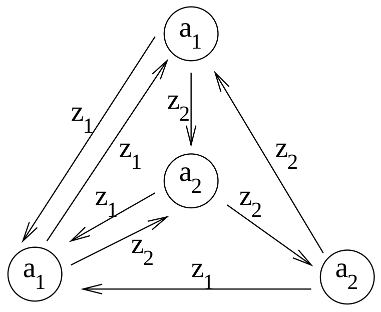


Figure 2.3: Finite state controller for a simple POMDP with two actions and two observations.

mation about the current state of the world. Let b_t be the beliefs of the agent about the state of the world at time-step t, which we represent by a probability distribution over the state space $\mathcal{S}$. Using Bayes theorem, one can compute the current belief state b_t from the previous belief state b_{t-1}, the previous action a_{t-1} and the current observation z_t:

$$\begin{aligned} b_t(s') &= C \sum_{s \in \mathcal{S}} b_{t-1}(s) \Pr(s'|s, a_{t-1}) \Pr(z_t|a_{t-1}, s') && (2.1) \\ &= C \sum_{s \in \mathcal{S}} b_{t-1}(s) T(s, a_t, s') Z(a_{t-1}, s', z_t) \; . && (2.2) \end{aligned}$$

Here, $b(s)$ denotes the probability of s according to the distribution of belief state b and C denotes a normalizing constant. Hence, belief state b_t is a sufficient statistic that captures the relevant information contained in the history of past actions and observations.

To summarize, we can represent a policy π as a mapping from initial belief states b_0 and histories $hist_t$ to actions a_t, or as a mapping from belief states b_t to actions a_t. The former is problematic because of the exponentially growing number of histories with respect to the horizon and the latter is problematic because the belief space is an $|\mathcal{S} - 1|$-dimensional continuous space. Fortunately, a key result by Smallwood and Sondik [132, 134] allows us to circumvent the continuous nature of belief space. In the following section, we first introduce value functions, and then discuss Sondik's solution.

Evaluation

Given the set of all policies Π, we need a mechanism to evaluate and compare policies. Roughly speaking, the goal of an agent is to maximize the amount of reward earned over time. This loosely defined criterion can be formalized in many ways [69]: one may wish to maximize *total* (accumulated) or *average* reward, *expected* or *worst-case* reward, *discounted* or *undiscounted* reward. Unless otherwise stated, this thesis assumes an *expected total discounted* reward criterion, since it is by far the most popular in the literature. Mathematically, we define the value $V^{\pi}(b_0)$ of executing some policy π starting at belief state b_0 to be the expected sum of the discounted rewards earned at each time-step:

$$V^{\pi}(b_0) = \sum_{t=0}^{h} \gamma^t \sum_{s \in \mathcal{S}} b_t(s) R(s, \pi(b_t)) \ . \tag{2.3}$$

Here, $b_t(s)$ denotes the probability of s according to belief state b_t and $\pi(b_t)$ denotes the action prescribed by policy π at belief state b_t.

Using value functions V, we are now in a position to order policies. A decision theoretic agent prefers π to π' when $V^{\pi}(b) \geq V^{\pi'}(b)$ for all belief states b. This preference ordering is a partial order because there are pairs of policies for which neither policy has a value function greater than the other one for all belief states. On the other hand, there always exists an optimal policy π^* such that its value function V^{π^*} dominates all other policies ($V^{\pi^*}(b) \geq V^{\pi}(b) \ \forall \pi, b$). This fact follows from Bellman's equation, which will be introduced in Section 2.2.

As with policies, representing a value function can be problematic because its domain is an $(|\mathcal{S}| - 1)$-dimensional continuous space corresponding to the belief space. Fortunately, a key result by Smallwood and Sondik [132, 134] shows that optimal value functions for finite-horizon POMDPs are piecewise-linear and convex (PWLC). The idea is that at any point in time during the execution of a policy, the actions prescribed for the remaining steps form a conditional plan. The value function of a conditional plan is constant for any world state. Since belief states represent probability distributions over the set of world states, the value function of a conditional plan at any belief state b is simply the weighted average (according to b) of the value at each world state. Thus, the value function $V^{\beta}(b)$ of a conditional plan β is linear with respect to b. This means that $V^{\beta}(b)$ can be represented by a vector α_{β} of size $|\mathcal{S}|$ such that $V^{\beta}(b) = \sum_{s \in \mathcal{S}} b(s) \alpha_{\beta}(s)$.[1]

For a finite horizon h, an optimal policy π^h consists of the best conditional plans for each initial belief state. More precisely, the best conditional plan β^* for some belief state b is the one that yields the highest value: $\beta^* = \operatorname{argmax}_{\beta} V^{\beta}(b)$. Although there

[1]The vector α_{β} actually lies in an $|\mathcal{S}| - 1$ dimensional space.

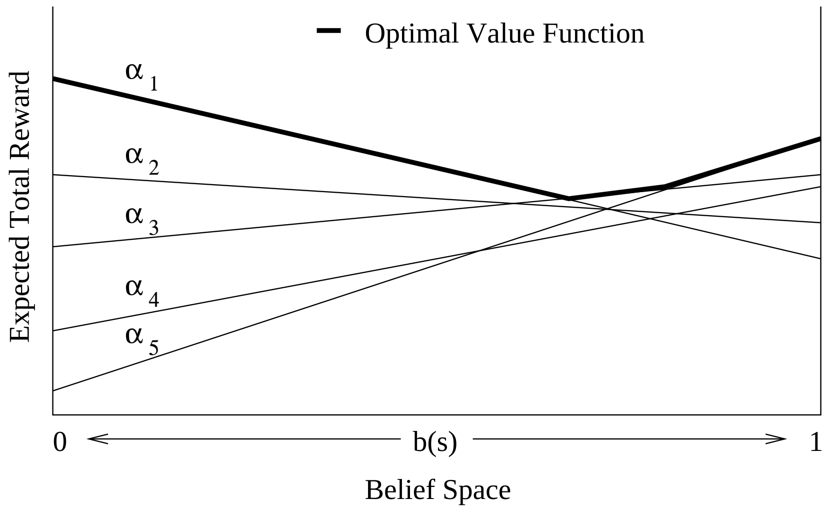


Figure 2.4: Geometric View of Value Function.

are uncountably many belief states, the set of h-step conditional plans Γ_h is finite and therefore an h-step optimal value function can be represented by a finite collection of α-vectors. For infinite horizon problems, the optimal value function may require an infinite number of α-vectors.

Figure 2.4 shows an optimal value function for a simple two-state POMDP. The horizontal axis represents belief space and the vertical axis indicates the expected total reward. Assuming the two world states are s and $\bar{s}$, then a belief state is completely determined by the probability of s. Therefore, the horizontal axis represents a continuum of belief states determined by the probability $b(s)$. Each line in the graph is an α-vector which corresponds to the value function of a conditional plan β. The upper surface of those α-vectors is a piecewise-linear and convex (PWLC) function corresponding to the optimal value function $V^*(b) = \max_\beta V^\beta(b)$.

2.1.3 Related Models

Before discussing solution algorithms that compute optimal policies for POMDPs, it is good to take a step back in order to have a broader view. Several other models are closely related to POMDPs. They usually differ by a few assumptions regarding the information available or the type of planning problems to which they cater. Although those related models are not our focus, outlining their differences with respect to POMDPs helps to characterize and situate POMDPs in this broader context. Furthermore, as we will see later, several solution algorithms for POMDPs are borrowed from or inspired by the solution algorithms for those related models.

Classical planning

Classical planning has traditionally adopted stringent certainty assumptions about the planner's knowledge of the world. More precisely, action effects are assumed to be deterministic and the current state of the world is assumed fully observable.[2] Furthermore, the goal of the planning agent is usually to reach some state within some set of desirable states as opposed to accumulating rewards for POMDPs. Also, instead of using a horizon to define the length of a plan, termination of the plan is assumed when a goal state is reached.

Classical planning problems can be modeled using POMDPs since deterministic transition functions are a special case of stochastic transition functions and goal states can be modeled by a reward function that assigns zero reward to those desired states and negative rewards to all other states. One can also simulate plan termination in an infinite-horizon POMDP by designing "absorbing" goal states. A state is absorbing when there is no possible transition out of this state.

Several extensions to classical planning have been proposed to allow uncertainty in action effects and partial observability of the state space. One broad class of extensions has lead to probabilistic *conformant* planners (BURIDAN [64, 65], UDTPOP [103], MAXPLAN [78], CPplan [51, 52], etc.), which model uncertainty in the initial state and action effects with probabilities, but do not make use of any observations. The resulting plan is a sequence of actions (as opposed to a "tree" of actions conditioned on observations for POMDPs) that will reach a goal state with probability greater than some predetermined threshold. A conformant plan succeeds (with high probability) regardless of the uncertainty, but may be of significantly lower quality since it ignores information gathered through observations.

A second class of extensions has lead to probabilistic *contingent* planners (C-BURIDAN [33], DTPOP [103], MAHINUR [96, 97], PGRAPHPLAN/TGRAPHPLAN [9], C-MAXPLAN [79], ZANDER [79, 77], etc.), which condition the choice of actions on observations. Both POMDPs and probabilistic contingent planning allow the encoding of uncertainty in states, actions effects and observations using probabilities. The main

[2]Although only the initial state of the world needs to be observable.

difference is the use of goal states (contingent planning) versus rewards (POMDPs); however, this is just a formulation difference since goal-oriented planning problems can always be transformed into equivalent reward-oriented planning problems and vice-versa. The goal-to-reward conversion was discussed earlier and the reward-to-goal conversion was established in [29]. Intuitively, by appropriately rescaling rewards between 0 and 1 and creating absorbing goal and sink states, the problem of maximizing total discounted rewards can always be transformed into an equivalent problem of maximizing the probability of goal achievement. It follows that the frameworks of probabilistic contingent planning and POMDPs are equally expressive [77].

In another line of work, situation calculus [86, 117] (which is a first-order language to model dynamical systems and to do classical planning) has also been extended to handle uncertainty. Decision-theoretic [108] and probabilistic [4] extensions have been proposed to allow stochastic actions as well as partial observability. These frameworks match the expressiveness of POMDPs for uncertainty while using a more powerful first order logic. POMDPs traditionally assume a propositional logic; however, several relational [39, 37] and first-order [16, 15] extentions have been proposed for fully observable MDPs. The remainder of this thesis will focus on propositional POMDPs.

Influence diagrams

Finite horizon POMDPs can also be viewed as special cases of *influence diagrams* [50]. An influence diagram is a directed acyclic graph used in decision analysis to model the influences between different components of a decision-theoretic problem. Figure 2.5 illustrates a finite horizon POMDP modeled as an influence diagram. The graph is composed of three types of nodes: chance nodes, decision nodes and utility nodes. Chance nodes are random variables (e.g., state variables and observation variables), decision nodes are variables set by a decision maker (i.e., action variables) and utility nodes are real variables (i.e., reward variables). There are also two types of arcs: probabilistic arcs and informational arcs. Arcs pointing into a chance node or a utility node indicate a probabilistic dependency between a child and its parents, whereas arcs pointing into a decision node indicate the information available to a decision maker (i.e., which nodes are observable) when deciding how to set the value of a decision node.

One can quantify the probabilistic dependencies of a chance node or a utility node with respect to its parents by defining a function that maps the values of parent nodes

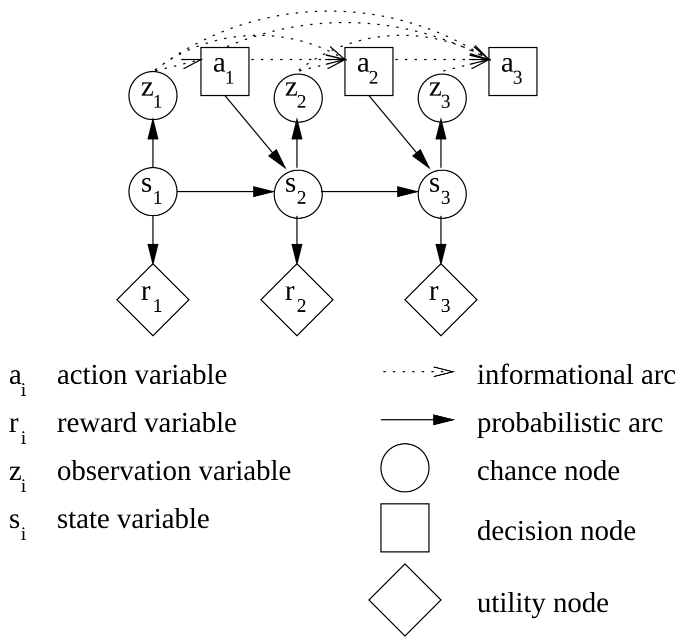


Figure 2.5: POMDP represented as an influence diagram.

to probability distributions over the values of the chance/utility node. In the POMDP example, these mappings are specified by the transition, observation and reward functions. Solving an influence diagram essentially consists of quantifying informational arcs by mapping the values of observed nodes to values of decision nodes. For the POMDP example, this is equivalent to finding a policy since decision nodes consist of actions and observed nodes consist of previous actions and observations. Thus, most influence diagram solution techniques (e.g., value-preserving reductions [125, 126], dynamic programming [139], Bayesian network conversion [127], etc.) can be used directly to find optimal POMDP policies represented as mappings from histories to actions.

In contrast with POMDPs, influence diagrams do not assume stationary state, action and observation spaces nor stationary transition, observation and reward functions. In other words, influence diagrams can model sequential decision making problems with state, action, observation spaces and transition, observation, reward functions that may be different at each time-step. Note however that this doesn't make influence diagrams more general than POMDPs since we can always turn a non-stationary process into a stationary one by expanding the non-stationary components with enough parameters until they become stationary. However, in practice, dealing with non-stationary components directly may (or may not) be advantageous. Note also that influence diagrams are typically used only for finite horizon processes since it is not possible to represent explicitely different state, action, observation spaces and transition, observation, reward functions for infinitely many steps.

2.2 Classic Solution Algorithms

Over the years, many algorithms have been proposed to find optimal POMDP policies. In the 1960s, the Operations Research community developed the POMDP framework which was first formalized by Drake [32]. Then, in the 1970s, the piecewise-linear and convex properties of optimal value functions were discovered by Smallwood and Sondik [132, 134]. This discovery enabled the formulation of several dynamic programming (DP) algorithms. This section reviews some of the classic DP-based algorithms with an emphasis on the sources of intractability that prevent them from scaling well. For a more extensive coverage of classic algorithms, the reader is referred to the surveys by Monahan [90], White [143] and Lovejoy [74].

2.2.1 Sondik's One-Pass Algorithm and Monahan's Algorithm

We first describe Sondik's One-Pass algorithm [132, 134] and Monahan's algorithm [90] due to their simplicity. These algorithms essentially compute the optimal value function at each time-step, starting from the horizon (assuming it is finite) and going backward in time. This is achieved by a dynamic programming procedure that computes the optimal value function V^{k+1} (at $k+1$ stages-to-go) from the optimal value function V^k (at k stages-to-go).

Dynamic programming backup

Intuitively, V^k tells us the highest expected total return that an agent can accumulate over the remaining k stages. Thus, if we want to find the highest expected total return for the remaining $k+1$ stages (i.e., V^{k+1}), we can decompose the problem into finding the highest combined return of the expected reward earned at $k+1$ stages-to-go with the expected total return for the remaining k stages. This idea is formalized by Bellman's equation:

$$V^{k+1}(b) = \max_a [r_a(b) + \gamma \sum_{z \in Z} \Pr(z|a,b) V^k(b_z^a)] \ . \tag{2.4}$$

Here, $r_a(b) = \sum_s b(s)R(a,s)$ is the expected reward earned for belief state b when action a is executed. Similarly, $\Pr(z|a,b) = \sum_s b(s)\Pr(z|a,s)$ corresponds to the probability of experiencing z when action a is executed in belief state b. Also, b_z^a is the updated belief state (according to Equation 2.1) after action a is executed and observation z is experienced. In theory, we can use Bellman's equation to compute V^{k+1} from V^k; however, because of the continuous nature of the belief space, computing $V^{k+1}(b)$ for each b is not feasible. Recall Sondik's observation that V^{k+1} and V^k are piecewise-linear and convex and therefore represented by some set of α-vectors, $\aleph^{k+1}$ and $\aleph^k$ respectively. Each vector α_β in $\aleph^k$ corresponds to some conditional plan β over k stages. Thus, by using the recursive definition of conditional plans (Section 2.1.2), we can obtain the set Γ_{k+1} of all conditional plans over $k+1$ stages from the set Γ_k of all conditional plans over k stages:

$$\Gamma_{k+1} = \{\langle a, \sigma \rangle \mid a \in \mathcal{A},\ \sigma : \mathcal{Z} \rightarrow \Gamma_k\} \ .$$

Similarly, we can compute the set $\aleph^{k+1}$ of all α-vectors as follows:

$$\aleph^{k+1} = \{r_a + \gamma \sum_{z \in Z} T^{a,z} \alpha_{\sigma(z)} \mid a \in \mathcal{A},\ \alpha_{\sigma(z)} \in \aleph^k\} \ .$$

Here, r_a is the $|\mathcal{S}| \times 1$ column vector that corresponds to the reward function for action a (i.e., $r_a(i) = R(a, s_i)$). Similarly, $T^{a,z}$ is the $|\mathcal{S}| \times |\mathcal{S}|$ transition matrix corresponding to the probability of reaching s_j and observing z when a is executed in s_i (i.e., $T^{a,z}(i,j) = \Pr(s_j|s_i,a)\Pr(z|s_j,a) = \Pr(s_j, z|s_i, a)$).

Tables 2.1 and 2.2 describe value iteration with Sondik's One-Pass algorithm to iteratively compute $\aleph^k$ at each time-step.[3] The optimal value function V^k is implicitly obtained from $\aleph^k$ by selecting the best α-vector for each belief state:

$$V^k(b) = \max_{\alpha \in \aleph^k} b \cdot \alpha \ .$$

[3]In Table 2.1, the function *DPbackup* can be instantiated by Sondik's *OnePassDPbackup* (Table 2.2) as well as other algorithms that we will see shortly, such as *MonahanDPbackup* (Table 2.4) and *IncrementalPruningDPbackup* (Table 2.5).

```
PROCEDURE ValueIteration($horizon$)
    $\aleph^0 \leftarrow \{$ zero vector $\}$
    for $k \leftarrow 1$ to $horizon$ do
        $\aleph^k \leftarrow$ DPbackup($\aleph^{k-1}$)
    return $\aleph^k$
END PROCEDURE
```

Table 2.1: Value iteration algorithm.

```
PROCEDURE OnePassDPbackup($\aleph^{k-1}$)
    $\aleph^k \leftarrow \{r_a + \gamma \sum_{z \in \mathcal{Z}} T^{a,z} \alpha_{\sigma(z)} \mid a \in \mathcal{A},\ \alpha_{\sigma(z)} \in \aleph^{k-1}\}$
    return $\aleph^k$
END PROCEDURE
```

Table 2.2: Sondik's One-Pass DP backup.

Finite-horizon optimal policies

An optimal policy π^k is also obtained by selecting at each stage the action a of the conditional plan $\beta = \langle a, \sigma \rangle$ that maximizes expected total return for the current belief state b:

$$\pi^k(b) = a \text{ such that } b \cdot \alpha_{\langle a,\sigma \rangle} \geq b \cdot \alpha \text{ for all } \alpha \in \aleph^k \ . \tag{2.5}$$

This implicit policy is essentially a mapping from belief states to actions. At each time-step, the agent would first update its belief state according to Equation 2.1 (to reflect the previous action executed and the observation made) and select the next action to execute according to Equation 2.5.

Alternatively, one could select the maximizing α_β-vector for the initial belief state and execute the corresponding conditional plan β directly. At each time-step, it suffices to follow the edge labeled with the observation made to reach the node with the next action to execute.

$$\begin{array}{rll}
\min\limits_{\{\delta,\, b\}} & \delta & \\
\text{s.t.} & b \cdot (\alpha' - \alpha) \leq \delta & \forall \alpha' \in \aleph - \{\alpha\} \\
& \sum_s b(s) = 1 & \\
& b(s) \geq 0 & \forall s \in \mathcal{S}
\end{array}$$

Table 2.3: LP-dominance test: vector α is dominated by $\aleph - \{\alpha\}$ when $\delta \geq 0$.

Pruning

Unfortunately, Sondik's One-Pass algorithm is intractable because of the exponentially growing number of conditional plans. A quick analysis of the algorithm reveals that

$$|\Gamma^{k+1}| = |\mathcal{A}||\Gamma^k|^{|\mathcal{Z}|} .$$

Thus, Γ^k grows exponentially with the observation space and doubly exponentially with the horizon. On the other hand, not all conditional plans in Γ_k are useful when determining the optimal value function V^k. Only those whose α-vector is maximal at some belief state are really necessary. Thus, Monahan proposed to prune $\aleph$ to its minimal representation by removing *dominated* α-vectors. For example, in Figure 2.4, α_2 and α_4 are dominated vectors since they are below the upper surface corresponding to the optimal value function. Consequently, one can safely remove α_2 and α_4 from $\aleph$ without changing V.

Dominated α-vectors can be detected by *linear programming* (LP) and sometimes by a simpler *pointwise dominance* test. A vector α is pointwise dominated by α' when the expected total reward of α is less than or equal to α' for every state (i.e., $\forall \alpha(s) \leq \alpha'(s) s \in \mathcal{S}$) and strictly less for some state (i.e., $\exists s \in \mathcal{S}\ \alpha(s) < \alpha'(s)$). In Figure 2.4, α_4 is pointwise dominated by α_3. Note however, that α_2 is not pointwise dominated by any single vector; it is dominated by a combination of vectors of the upper surface. This is detectable by an LP-dominance test (Table 2.3), which verifies that for all belief states the expected total return of α is lower than that of some other vector α'.

Table 2.4 describes Monahan's version of a DP backup, which consists of the one-pass DP backup with a pruning step to reduce $\aleph^k$ to its minimal representation. More specifically, *prune*($\aleph^k$) removes all vectors in $\aleph^k$ that are dominated by other vectors according to the pointwise dominance test or the LP-dominance test. Pruning affects the running time by reducing the base in the exponential complexity. In practice, this can

```
PROCEDURE MonahanDPbackup($\aleph^{k-1}$)
    $\aleph^k \leftarrow \{r_a + \gamma \sum_{z \in \mathcal{Z}} T^{a,z} \alpha_z | a \in \mathcal{A}, (\forall z \in \mathcal{Z}) \alpha_z \in \aleph^{k-1}\}$
    $\aleph^k \leftarrow prune(\aleph^k)$
    return $\aleph^k$
END PROCEDURE
```

Table 2.4: Monahan's DP backup.

translate in substantial time savings; however, the algorithm remains exponential in the worst case and intractable for many problems.

Infinite-horizon ϵ-optimal policies

For infinite-horizon problems, an ϵ-optimal policy (policy whose value function differs from the optimal infinite-horizon policy π^* by at most ϵ for every belief state) can also be found by Sondik's and Monahan's algorithms. The idea is to find the k-step optimal value function V^k and to execute at every time-step the action a corresponding to the maximizing vector $\alpha_{\langle a,\sigma \rangle}$ in $\aleph^k$. Let $\hat{\pi}^k$ and $\hat{V}^k$ be the policy and value function corresponding to this strategy. Here, $\hat{\pi}^k$ differs from π^k since at every time-step t $\hat{\pi}^k$ chooses the maximizing α-vector in $\aleph^k$ instead of $\aleph^{k-t}$. As a consequence, $\hat{\pi}^k$ is only represented as a mapping from belief states to actions. The corresponding implicit conditional plan is not available.

We can estimate the quality of $\hat{\pi}^k$ by measuring the distance between $\hat{V}^k$ and the optimal infinite-horizon value function V^*. If the Bellman residual, that is, the L_∞ difference between V^{k-1} and V^{k-2}, is bounded by some δ (i.e., $\sup_b |V^{k-1}(b) - V^{k-2}(b)| \leq \delta$), then one can show that $\hat{V}^k$ and V^* differ by at most $2\delta\gamma/(1-\gamma)$ for every belief state [116]. If δ is picked to be $\epsilon(1-\gamma)/2\gamma$, then $\sup|\hat{V}^k(b) - V^*(b)| \leq \epsilon$ and we say that $\hat{\pi}^k$ is ϵ-optimal. In practice, δ can be made arbitrarily small since the Bellman residual decreases with each DP backup by a factor of at least γ.

2.2.2 Linear Support and Witness Algorithms

In Sondik and Monahan's algorithms, the exponential number of α-vectors generated to compute V^k at each time-step is the main source of intractability. On the other hand, the pruning step in Monahan's algorithm often shrinks $\aleph^k$ to a small fraction of its original

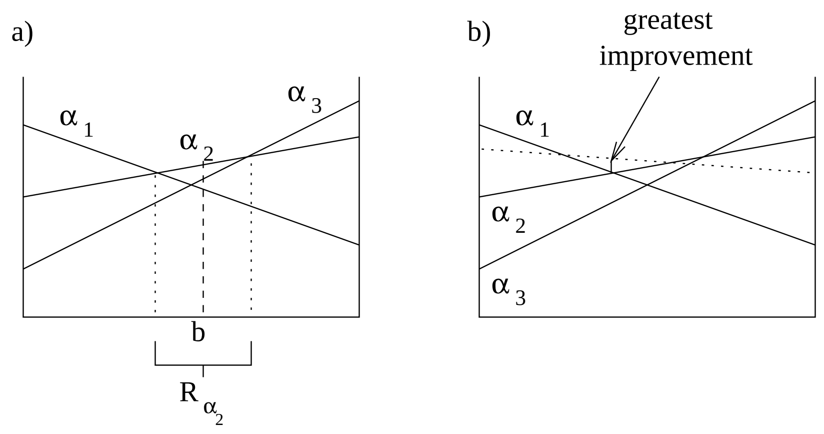


Figure 2.6: Example of a support region.

size. This observation suggests that if we were able to generate only the necessary vectors, many problems may have a tractable solution. In this section, we describe two algorithms that generate only the non-dominated vectors (necessary vectors): *linear support* by Cheng [26] and the *Witness algorithm* by Kaelbling, Littman and Cassandra [55].

Cheng's Linear support algorithm

The name "linear support" comes from the idea that non-dominated vectors provide *support* to the value function. A non-dominated vector α is optimal for some non-empty belief space region R_α called the *support region* of α (i.e., $\forall b \in R, \forall \alpha' \neq \alpha,\ b \cdot \alpha > b \cdot \alpha'$). A belief state b in the support region R_α is called a "witness" of α because it can testify that α is non-dominated since α is maximal at b. In Figure 2.6a, b is a witness of α_2 in its support region R_{α_2}.

Given any belief state b, it is easy to find the maximal vector α that provides support to V^k at b. It suffices to do a one-step lookahead to find the conditional plan $\beta = \langle a, \sigma \rangle$ with the best observation strategy $\sigma_a(z) = \text{argmax}_{\beta' \in \Gamma^{k-1}}\ b_z^a \cdot \alpha_{\beta'}$ and the best action $a = \text{argmax}_{a \in \mathcal{A}}\ b \cdot (r_a + \alpha_{\sigma_a(z)})$. Thus, finding the set of support vectors can be achieved by finding a set of corresponding witnesses.

It turns out that given a subset of support vectors, one can search for witness belief states of the missing support vectors among the vertices of the support regions of the current support vectors. To illustrate, if we add the dotted α-vector to $\aleph^k$ in Figure 2.6b, then the value-function will necessarily improve in at least one belief state that corresponds to a vertex and furthermore, the belief state with greatest value improvement is also a vertex. In this case, it is the intersection of α_1 and α_2.

Cheng's DP procedure incrementally generates the set of support vectors by searching for the vertex that witnesses the missing support vector providing the greatest improvement to the value function. Although the algorithm generates only the necessary support vectors, it may take an exponential amount of time to find a vertex that witnesses a missing support vector since a support region may have an exponential number of vertices and consequently the number of vertices in the current approximation can be exponential [70].

The Witness algorithm

Designing an algorithm that finds the minimal set $\aleph^k$ for V^k and that runs in time polynomial with respect to $|\mathcal{S}|$, $|\mathcal{A}|$, $|\mathcal{Z}|$, $|\aleph^k|$ and $|\aleph^{k-1}|$ is impossible unless NP = RP [70]. As an alternative, the Witness algorithm was designed to compute V^k indirectly by finding the Q functions of each action in polynomial time. The function Q_a^k represents the expected total return of executing action a at k stages to go followed by an optimal policy for the remaining $k-1$ stages:

$$\begin{aligned} Q_a^k(b) &= r_a(b) + \sum_{z \in \mathcal{Z}} \Pr(z|a,b) V^{k-1}(b_z^a) \;, \\ V^k(b) &= \max_a Q_a^k(b) \;. \end{aligned}$$

The Q_a^k functions are also piecewise-linear and convex, and therefore can be represented by sets $\aleph_a^k$ of α-vectors. The Witness algorithm essentially constructs for each action a the minimal sets $\aleph_a^k$ and merges them to obtain $\aleph^k$. The main achievement of this algorithm is a subroutine to construct each $\aleph_a^k$ in time polynomial with respect to $|\mathcal{S}|$, $|\mathcal{A}|$, $|\mathcal{Z}|$, $|\aleph_a^k|$ and $|\aleph^{k-1}|$. We can define a class of *polynomially action-output bounded* POMDPs (i.e., POMDPs such that for each action a the set $\aleph_a^k$ is polynomial with respect to $|\mathcal{S}|$, $|\mathcal{A}|$, $|\mathcal{Z}|$ and $|\aleph^{k-1}|$), for which the Witness algorithm has a DP procedure that runs in polynomial time. Note, however, that the number of α-vectors may still grow exponentially with the horizon.

Given a subset of the conditional plans that support Q_a^k, the Witness algorithm efficiently finds a belief state that witnesses a missing support conditional plan by perturbing the current support conditional plans. The idea is that we can verify that a conditional plan $\beta = \langle a, \sigma \rangle$ is not optimal at some witness belief state (in β's support region) if there exists a better conditional plan $\langle a, \sigma' \rangle$ obtained by slightly perturbing the observation strategy so that σ' differs from σ at only one observation z. Since there are only $|\mathcal{Z}||\aleph^{k-1}|$ such perturbations for a given conditional plan, the search for witnesses remains polynomially bounded. As with Cheng's algorithm, once a witness is found, the support conditional plan that maximizes the expected total return for that witness is easily computed by a one-step lookahead.

2.2.3 Incremental Pruning

In order to avoid generating dominated α-vectors, the linear-support and Witness algorithms introduced special search procedures to determine belief states that witness the support vectors. A drawback of those procedures is their conceptual complexity, which makes them less attractive to implement when compared to the simple DP backup of Sondik and Monahan's algorithms. Zhang and Liu [146] proposed a new algorithm called incremental pruning that achieves both simplicity and computational efficiency. This algorithm performs a DP-backup in 3 steps and at each step it prunes dominated vectors. Bellman's equation (2.4) can be decomposed in the following 3 steps:

$$\begin{aligned} Q^k_{a,z}(b) &= \frac{r_a(b)}{|\mathcal{Z}|} + \gamma \Pr(z|b,a) V^{k-1}(b^a_z) \ , && (2.6) \\ Q^k_a(b) &= \sum_{z \in \mathcal{Z}} Q^k_{a,z}(b) \ , && (2.7) \\ V^k(b) &= \max_a Q^k_a(b) \ . && (2.8) \end{aligned}$$

Equation 2.6 computes $Q^k_{a,z}$, which is the portion of the Q^k_a function that pertains to making observation z. More precisely, $Q^k_{a,z}$ is the expected total return of executing action a and observing z, weighted by $\Pr(z|a,b)$ and assuming an optimal policy for the remaining $k-1$ stages. Equation 2.7 computes the Q^k_a function as in the Witness algorithm and Equation 2.8 computes the optimal value function V^k. Each of these functions is piecewise-linear and convex and therefore can be represented by a set of α-vectors. The above three equations can be rewritten in terms of sets of α-vectors as follows:

```
PROCEDURE IncrementalPruningDPbackup($\aleph^{k-1}$)
    $\aleph^k \leftarrow \emptyset$
    for each $a \in \mathcal{A}$ do
        $\aleph^k_a \leftarrow \emptyset$
        for each $z \in \mathcal{Z}$ do
            $\aleph^k_{a,z} \leftarrow prune(\{\frac{r_a}{|\mathcal{Z}|} + \gamma T^{a,z}\alpha | \alpha \in \aleph^{k-1}\})$
            $\aleph^k_a \leftarrow prune(\aleph^k_a \oplus \aleph^k_{a,z})$
        $\aleph^k \leftarrow prune(\aleph^k \cup \aleph^k_a)$
    return $\aleph^k$
END PROCEDURE
```

Table 2.5: Incremental pruning DP backup.

$$\aleph^k_{a,z} = \{\frac{r_a}{|\mathcal{Z}|} + \gamma T^{a,z}\alpha | \alpha \in \aleph^{k-1}\} \; , \tag{2.9}$$

$$\aleph^k_a = \bigoplus_{z \in \mathcal{Z}} \aleph^k_{a,z} \; , \tag{2.10}$$

$$\aleph^k = \bigcup_{a \in \mathcal{A}} \aleph^k_a \; . \tag{2.11}$$

The symbol $\oplus$ means pairwise addition (i.e., $\aleph \oplus \aleph' = \{\alpha + \alpha' | \alpha \in \aleph \;\wedge\; \alpha' \in \aleph'\}$). The incremental pruning algorithm for DP backups is described in Table 2.5 and can be used in the value iteration algorithm of Table 2.1. Cassandra, Littman and Zhang [23] showed that incremental pruning has the same computational complexity as the Witness algorithm and usually outperforms it by a constant factor in practice.

2.2.4 Hansen's Policy Iteration

All the algorithms presented so far are *value iteration* algorithms since they iteratively compute the optimal value function at each time-step. Sondik [134, 135], Hansen [44, 45] and Meuleau et al. [89, 88] also proposed *policy iteration* algorithms that conduct an iterative search directly within the space of policies. In this section, we describe Hansen's policy iteration because of its elegance and simplicity.

Hansen's algorithm uses finite state controllers to represent policies. Recall that a finite state controller is a cyclic directed graph with vertices labeled by actions and edges labeled by observations (Figure 2.3). Vertices do not (necessarily) correspond to world states or belief states; they correspond to (possibly infinite) conditional plans. The vertex corresponding to conditional plan $\beta = \langle a, \sigma \rangle$ is labeled with action a and its outward edges correspond to the observation strategy σ. At run time, the agent executes the action labeling the current active vertex. It then makes an observation z, which is used to update the active vertex by following the outward edge labeled with z. This edge points to the next active vertex, which corresponds to the conditional plan $\sigma(z)$. The initial active vertex is determined by selecting the conditional plan β_0 that maximizes the expected total return for the initial belief state b_0 (i.e., $\beta_0 = \text{argmax}_{\beta \in \Gamma}\ b_0 \cdot \alpha_\beta$). Since the initial conditional plan is picked to be maximal with respect to the initial belief state, the value function V^π of a finite state controller π corresponds to the upper surface of the set of α-vectors $\aleph^\pi$ associated with the conditional plans of each vertex of π.

Hansen's algorithm can be used to solve discounted, infinite-horizon POMDPs. It computes a sequence of finite state controllers $\langle \pi_0, \pi_1, \ldots, \pi_n \rangle$ such that in the limit (when $n \to \infty$), π_n converges to the optimal policy π^*. That is, the value function of the controllers in the sequence increases monotonically and in the limit, converges to V^*. Given a controller π_i, the next controller π_{i+1} is obtained greedily in two steps: *policy evaluation* (compute V^{π_i} of π_i) and *policy improvement* (perform a DP backup and extract π_{i+1}).

Policy evaluation computes the set $\aleph^\pi$ of α-vectors for some controller π. This is done by solving a system of $|\Gamma^\pi||\mathcal{S}|$ equations of the following form:

$$V^{\langle a,\sigma \rangle}(s) = r_a(s) + \gamma \sum_{s' \in \mathcal{S}} \Pr(s'|s,a) \sum_{z' \in \mathcal{Z}} \Pr(z'|a,s') V^{\sigma(z')}(s') \quad \forall s \in \mathcal{S},\ \forall \langle a, \sigma \rangle \in \Gamma^\pi \ . \tag{2.12}$$

In each equation, $V^\beta(s)$ is the expected total reward for executing the conditional plan β when starting in state s. Alternatively, $V^\beta(s)$ is the s component of the vector $\alpha_\beta \in \aleph^\pi$ that corresponds to conditional plan $\beta \in \Gamma^\pi$.

Policy improvement is the key step in Hansen's algorithm. It takes the set of α-vectors $\aleph^{\pi_i}$ computed by policy evaluation and performs a DP backup as in incremental pruning. Let $\aleph^{\pi_i}_+$ be the set of α-vectors resulting from the backup. For each α-vector in $\aleph^{\pi_i}_+$ it creates a new node with outward edges corresponding to the action and observation strategy used to compute the α-vector. Then, it deletes the old nodes that are now pointwise-dominated by a new node. The inward edges of a deleted node are re-routed to its pointwise dominating new node since it offers better value for the entire belief space.

Note that LP-dominated old nodes are not deleted since their inward edges cannot be re-routed to a single dominating node.

In practice, some POMDPs have optimal policies corresponding to *infinite* state controllers. As with value iteration algorithms, one can find an ϵ-optimal finite state controller by ensuring that Bellman residual is small enough. The argument is identical to the one for value iteration since both use DP backups. Empirically, it has been observed that Hansen's algorithm requires fewer iterations than value iteration to converge to an ϵ-optimal policy. On the other hand, each iteration of Hansen's algorithm consists of a policy evaluation step and a DP backup, whereas an iteration of value iteration only performs a DP backup. Since the running time of a policy evaluation step tends to be negligible compared to a DP backup, Hansen's algorithm usually runs faster overall. Although it runs faster, its worst case computational complexity remains intractable because the number of vertices (corresponding to α-vectors) may grow exponentially.

2.3 Performance Issues

We now have reviewed some of the key classic algorithms to solve POMDPs. They have given us a general idea for the solution techniques as well as some of the performance issues. In this section, we outline the two main sources of intractability that prevent classic algorithms from scaling up effectively.

2.3.1 Policy Search

Solving POMDPs is a notoriously hard problem. In fact, finding an optimal policy has been shown to be PSPACE-complete for finite-horizon problems [99] and verifying the existence of a policy with a value function greater than some threshold for a given initial belief state is undecidable [76]. Furthermore, finding ϵ-optimal policies for any ϵ is NP-hard [75]. Yet, POMDPs remain a very attractive framework given that they can be used to naturally model numerous real-world problems. In practice, the hardness of finding optimal POMDP policies arises from two sources of intractability that plague classic algorithms:

- Complex value function and policy representations (also known as the curse of history [105]),

- Large state space (also known as the curse of dimensionality [105].

We have already observed the first problem when representing value functions as sets of α-vectors. The size of $\aleph^k$ may grow exponentially with the observation space and doubly exponentially with the horizon (even when dominated α-vectors are pruned or not generated). As for the second problem, most real-world POMDPs tend to have large state spaces. In many domains, the state space is defined by a set of variables such that each state corresponds to a joint instantiation of those variables. Hence, the number of states (and the dimensionality of α-vectors) is exponential in the number of state variables.

2.3.2 Policy Execution

Once a policy π is found, it remains to be executed. If it is represented as a mapping from histories to actions, such as a conditional plan or a finite state automaton, then at each time-step, the agent simply has to follow the edge labeled with the current observation to reach the next node and execute its corresponding action.

However, many existing algorithms produce policies represented as mappings from belief states to actions. In that case, policy execution is not as straightforward since one needs to monitor (update) the belief state (using Equation 2.1) at each time-step in order to retrieve the next action to execute. Since a belief state is a probability distribution over the state space, belief state monitoring may be intractable for large state spaces. Furthermore, although the size of the state space may be reasonable for an algorithm to find a good policy *off-line*, it may be too large when executing the policy *online*. The time constraints to execute a policy are typically much more stringent than those for finding the policy. For instance, a car maker may take several months to design and compile a POMDP auto-pilot; however, once installed in a car, the auto-pilot will have to react in real-time to challenging traffic conditions.

2.3.3 Structure

Although classic POMDP algorithms do not scale in practice, real-world problems often exhibit a significant amount of structure that can be exploited to mitigate the above sources of intractability. The fact that variants of the POMDP policy search problem are NP-hard, PSPACE-complete or undecidable simply means that in the *worst case* solutions are impractical to find. Subclasses of POMDPs may be much easier to solve. In fact, as mentioned in Section 2.2.2, the Witness algorithm can perform DP backups in polynomial time for the class of polynomially action-output bounded POMDPs. Unfortunately, most real-world POMDPs are not known to be part of this class.

Nevertheless, real-world POMDPs tend to exhibit a significant amount of structure and the key will be to design scalable algorithms that exploit it. To that effect, Chapter 3 reviews some of the techniques proposed by the research community to deal with the complexity of value function and policy representations, and then describes a new algorithm called *bounded policy iteration* (BPI). Similarly, Chapter 4 summarizes recent algorithms and the structure they exploit to handle exponentially large state spaces, and then describes a new *value-directed compression* (VDC) technique to mitigate the curse of dimensionality. In order to tackle large-scale POMDPs, both sources of intractability must be simultaneously contained, so Chapter 5 describes how to combine BPI and VDC as well as the point-based value iteration algorithm called Perseus [142, 137, 136] with VDC and ADDs.

Chapter 3

Compact Value Function and Policy Representations

Between the two sources of intractability identified in Section 2.3, the complexity of policy and value function representations tends to be the most important one. POMDPs with as few as two states can easily have optimal value functions requiring exponentially many α-vectors. This chapter focuses on techniques for dealing with the complexity of policy and value function spaces. In particular, Section 3.1 reviews a wide range of techniques proposed in the literature to compactly represent and efficiently manipulate policies and value functions. Based on the insights gained from previous work, Section 3.2 describes a novel algorithm called *bounded policy iteration* (BPI). Finally, Section 3.3 demonstrates BPI's ability to overcome the complexity of policy space on a suite of benchmark problems.

3.1 Literature Review

Recall that optimal value functions for discrete POMDPs are piecewise linear and convex, and therefore representable by a set of α-vectors, each corresponding to the value of a conditional plan. Unfortunately, the number of α-vectors (and conditional plans) tends to grow exponentially with the observation space and doubly exponentially with the horizon. This section reviews some approximation techniques to compactly represent value functions and policies.

3.1.1 Parsimonious Sets of α-vectors

An early approach to keep value functions tractable was to force Q-functions to be linear. Recall from Section 2.2.2 that function Q_a^k represents the expected total return for executing action a at k stages-to-go followed by an optimal policy for the remaining $k-1$ stages. The optimal value function at k stages-to-go is simply $V^k(b) = \max_a Q_a^k(b)$ and $\aleph^k = \bigcup_{a \in \mathcal{A}} \aleph_a^k$ (where $\aleph_a^k$ is the set of α-vectors that represents Q_a^k). Making the Q functions linear means restricting $\aleph_a^k$ to one α-vector, and consequently, restricting $\aleph^k$ to $|\mathcal{A}|$ α-vectors. Chrisman [27] and McCallum [83] used this approach while learning POMDPs, since it allowed them to easily adapt classic reinforcement learning algorithms that learn linear Q-functions. Later on, Littman et al. [67] experimented with several algorithms (including those by Chrisman and McCallum) that find linear approximations to Q-functions. The approximations are often good for small problems, but they quickly degrade for large problems due to their limited degrees of freedom.

Several of the classic solution algorithms can be modified to output reduced or bounded sets of α-vectors. In particular, Cheng [26] proposed an approximate version of his linear support algorithm that generates only the most important α-vectors at each DP backup. Recall from Section 2.2.2 that the DP procedure of the linear support algorithm incrementally refines the next value function V^k by iteratively adding to $\aleph^k$ the α-vector that provides the greatest improvement to the current estimate of V^k. One can either bound the size of $\aleph^k$ by stopping the DP procedure after a predetermined number of α-vectors are constructed, or one can bound the approximation error by stopping the DP procedure when the next α-vector to be constructed provides an improvement to the value function that is smaller than some error threshold. The Witness algorithm [55] can be used in a similar fashion to incrementally construct a parsimonious set of α-vectors. Feng and Hansen [36] also proposed a scheme to prune α-vectors that marginally contribute to a value function in the incremental pruning DP procedure. When carrying out an LP-dominance test (Table 2.3), the idea is to prune a vector even when it is not dominated as long as its contribution is smaller than some error threshold.

Zhang and Zhang [147] also improved the running time of incremental pruning by interleaving DP backups with more efficient *point-based* backups. A point-based backup is an approximate DP backup that generates a subset of the support vectors by quickly selecting a few witness belief states and computing in a one-step lookahead the optimal conditional plan (and corresponding α-vector) for each witness belief state. The point-based version of incremental pruning provides substantial time savings in practice; however, the number of α-vectors still grows exponentially. Realizing that only a small portion of the belief space tends to be reachable from the initial belief state, Pineau et al. [106] proposed a *point-based value iteration* (PBVI) algorithm that performs point-based backups only for witness belief states that are reachable. The algorithm alternates between point-based backups and forward search to generate new reachable belief points. This technique tends to perform very well in practice since for many POMDPs the optimal value function of the reachable belief region can often be well approximated by a small set of α-vectors. Spaan and Vlassis [142, 137, 136] proposed a randomized version of PBVI called Perseus that performs *partial* point-based backups by stopping early the generation of support vectors once the value of all witness points has improved. Smith and Simmons [133] also combined heuristic search with point-based value iteration to obtain a new algorithm called HSVI that simultaneously computes lower bounds (by point-based backups) and upper bounds (by heuristic search) of the optimal value function.

In general, one may be interested in finding the *best* policy or value function of a given size. To that effect, Parr and Russell [100] devised the SPOVA (smooth partially observable value approximation) algorithm. It is a gradient descent algorithm that directly minimizes Bellman residual by approximating the value function with a smooth combination of a predetermined number of α-vectors. Since gradient descent works only for continuous functions, the value function $V^*(b) = \max_{\alpha \in \aleph^*} b \cdot \alpha$ is made continuous by smoothing the max operator by some L_k-norm (i.e., $V^*(b) = \sqrt[k]{\sum_{\alpha \in \aleph^*} (b \cdot \alpha)^k}$). In a similar spirit, Meuleau et al. [88] as well as Aderbeen and Baxter [1] proposed gradient descent algorithms to find the best stochastic finite state controllers of a given size. Several other gradient-based algorithms [94, 93, 56, 5] have been proposed to search within restricted classes of policies determined based on prior expert knowledge. In general, gradient-based algorithms are simple and efficient, but they are not guaranteed to find the best policy since they may get trapped into a local optimum. To circumvent local optima, Meuleau et al. [89] and Braziunas et al. [20] respectively proposed branch and bound as well as stochastic local search algorithms for finite state controllers of a given size. Unfortunately, these techniques do not scale very well and tend to become impractical for controllers with more than 30 nodes.

3.1.2 Grid-based Representations

Part of the difficulty in accurately representing value functions is their continuous domain: the belief space. A popular approach to get around the continuous nature of the belief space is to select a finite grid (or more generally a finite set) of belief states for which we store the corresponding values. The values of all other belief states are interpolated from the values of the grid points. In general, grid-based algorithms [32, 57, 35, 73, 18, 47, 140, 48, 148] vary depending on the regularity of the grid, the resolution of the grid and the interpolation technique.

Lovejoy [73] proposed a *fixed-resolution regular* grid that selects grid points equally spaced in the belief simplex. A very efficient interpolation technique based on a triangulation concept assigns to non-grid belief states the convex combination of the values of nearby grid belief states. Unfortunately, as the resolution increases, the number of grid points grows exponentially with the size of the state space (which may already be exponential with respect to the number of state variables). In contrast, Hauskrecht [47, 48] and Brafman [18] proposed *variable-resolution non-regular* grids, which allow one to increase resolution in areas of poor accuracy by adding new grid points that are not necessarily equally spaced. This tremendously reduces the number of grid points while achieving similar accuracy; however, because grid points are unevenly spaced, interpolation techniques are much more computationally intensive. Recently, Zhou and Hansen [148] proposed a *variable-resolution regular* grid that allows both fast interpolation and increased resolution in only the necessary areas. Despite these refinements, experimental results [148] demonstrate that grid-based methods do not scale well for large state spaces since the size of the grid tends to grow exponentially with the number of states.

3.1.3 Neural Networks

Value functions can also be approximated by neural networks. Bertsekas and Tsitsiklis [8] proposed a general technique called *neuro-dynamic programming* to learn MDP value functions with large state spaces as well as continuous state spaces (such as belief spaces). The idea is to train a neural network by dynamic programming to approximate Q-functions. The neural network takes as inputs a belief state and an action, and it outputs an estimate of the corresponding Q-value. Unfortunately, the exact Q-value is not available, so Bertsekas and Tsitsiklis bootstrap the learning process by using a variant of Bellman's equation for Q-functions:

$$Q_a^{k+1}(b) = r_a(b) + \gamma \sum_{z \in Z} \Pr(z|a,b) \,.\, \max_{a'} Q_{a'}^k(b_z^a) \tag{3.1}$$

The idea is to use the right hand side of the above equation as the exact Q-value and to use the neural network to estimate the Q-values at b_z^a. The neural network can be trained either by sampling the belief space uniformly or by generating sequences of belief states obtained by executing the policy associated with the current value function. The latter has the advantage of focusing training efforts on regions of the belief space that are more likely to be visited.

Lin and Mitchell [66] also proposed a feed-forward neural network to learn Q-values from finite histories (instead of belief states) and two recurrent neural networks to learn simultaneously Q-values and sufficient statistics of the belief space from only the most recent observation. Those neural networks are also trained by using the right hand side of Equation 3.1 as a surrogate for the exact Q-values.

In general, there are no convergence guarantees when neural network approximations of the Q-functions are trained by using the right hand side of Equation 3.1 as a surrogate for the exact Q-values [8]. On the other hand, Baird and Moore [7] showed that gradient-based algorithms that minimize the *residual* of Bellman's equation are guaranteed to converge to a local optimum. The idea is to use a neural network (or some other function approximation) to approximate the Q's in *both*, the left hand side and the right hand side of Equation 3.1, and to minimize the difference between the two sides of Equation 3.1 by gradient descent. Although convergence to a local optimum is guaranteed, the quality of this local optimum varies significantly due to the lack of convexity of the optimization and the often arbitrary architecture of neural networks.

3.1.4 Bounded Histories

Another way of compactly approximating value functions is to define them as mappings from finite histories (instead of belief states) to expected total return. Recall from Section 2.1.2 that histories and belief states are equivalent in terms of the information they provide about the current state of the world. If we bound histories to some length, then only a finite though exponential (in length) number of histories are possible and the value function has a simple vector representation over the discrete space of bounded histories. If we were going to rewrite the transition, observation and reward functions in terms of bounded histories (instead of states), then a POMDP would be converted into an NMDP (non-Markovian decision process). Since bounded histories are observable, the POMDP appears to have been converted to an MDP; however, bounded histories do not (necessarily) possess sufficient information to predict future states so the Markov property is lost, rendering the decision process non-Markovian (in the worst case).

Nevertheless, a fair amount of work in the POMDP reinforcement learning literature has focused on such approximations. In particular, work on *memoryless* policies has been very popular [68, 130, 131, 53, 72, 6]. A memoryless policy selects actions based only on the last observation. The belief space is essentially reduced to the observation space (histories of length 1). This reduction allows standard reinforcement learning algorithms such as Q-learning and TD-λ to learn a good memoryless policy by ignoring the fact that the Markov property doesn't hold. Furthermore, memoryless policies can be represented by a table mapping each observation to some action, which circumvents the need for belief state monitoring.

Unfortunately, memoryless policies can be far from optimal due to the loss of information caused by restricting belief states to the last observation. Several researchers have explored methods for learning *finite memory* policies based on bounded histories [104, 84, 85, 66]. Those techniques offer a range of accuracy/complexity tradeoff since accuracy gains can be had (as we increase the maximum length of histories) at the cost of an exponentially growing number of possible histories.

3.2 Bounded Policy Iteration

Although the optimal policies of most POMDPs are usually composed of an exponential, if not an infinite, number of conditional plans, real-world POMDPs often have policies that are very simple and yet very good. Furthermore, when interested only in a policy for a given initial belief state, tailoring the policy to the belief region reachable from that initial belief state can be an effective way of further reducing the complexity of policy and value function representations. If the reachable belief region is a small subset of the entire belief space, then tailored policies can be much simpler to find and represent than full policies. As a result, the most successful approaches to date are those that exploit domain expert knowledge to search for good tailored policies within a restricted class of compactly representable policies (e.g., *PEGASUS* for helicopter control [92, 94] and *covariant policy search* for the game of Tetris [5]).

When prior knowledge is unavailable or difficult to elicit, bounded finite state controllers (controllers with a bounded number of nodes) provide a natural class of compact policies. Furthermore, the execution of policies represented by finite state controllers can be done in real-time since there is no need for belief state monitoring. Existing algorithms for finite state controllers (FSCs) include classic policy iteration (PI) [44, 45], gradient ascent (GA) [88, 1], branch and bound (B&B) [89] and stochastic local search (SLS) [20]. As explained earlier in Section 2.2.4, PI tends to generate FSCs that grow exponentially in size and therefore is not suitable for bounded FSCs. GA, B&B and SLS can all be used to search for a good controller of a given size; however, existing implementations of B&B and SLS scale poorly and GA may get trapped in a local optimum.

While locally optimal solutions are often acceptable, for many planning problems with a combinatorial flavor, GA can easily get trapped by simple policies that are far from optimal. Consider a system engaged in preference elicitation, charged with discovering an optimal query policy to determine relevant aspects of a user's utility function. Often no single question yields information of much value, while a sequence of queries does. If each question has a cost, a system that locally optimizes the policy by GA may determine that the best course of action is to ask no question (i.e., minimize cost given no information gain). When an optimal policy consists of a sequence of actions such that any small perturbation results in a bad policy, there is little hope of finding this sequence using methods that greedily perform local perturbations such as those employed by GA.

In general, we would like the best of both worlds: scalability and convergence to a global optimum. While finding the best deterministic controller of a given size is NP-hard [68, 88], one can hope for a tractable algorithm that at least avoids obvious local optima. This section describes a new anytime algorithm, *bounded policy iteration (BPI)*, that improves a policy much like classic PI while keeping the size of the controller fixed. Whenever the algorithm gets stuck in a local optimum, the controller is allowed to slightly grow by introducing one (or a few) node(s) to escape the local optimum.

3.2.1 Policy Iteration for Stochastic Controllers

Recall that classic policy iteration (PI) [44, 45] incrementally improves a controller by alternating between two steps, policy improvement and policy evaluation, until convergence to an optimal policy. Policy evaluation solves Equation 2.12 for a given policy. Policy improvement adds nodes to the controller by dynamic programming (DP) and removes other nodes. A DP backup applied to the value function (V in Figure 3.1a) of

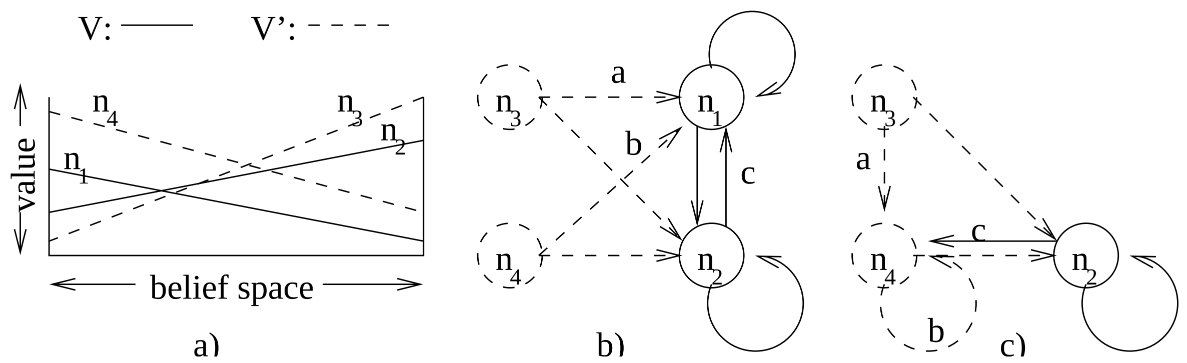


Figure 3.1: a) Value function V and the backed up value function V' obtained by DP; b) original controller (n_1 and n_2) with nodes added (n_3 and n_4) by DP; c) new controller once pointwise dominated node n_1 is removed and its inward arcs a, b, c are redirected to n_4.

the current controller yields a new, improved value function (V' in Figure 3.1a). Each α-vector composing V' corresponds to a new node added to the controller (in Figure 3.1b, nodes n_3 and n_4 are new nodes added to the original controller composed of n_1 and n_2). After the new nodes created by DP have been added, old nodes that are now *pointwise dominated* are removed (e.g., n_1 is pointwise dominated by n_4 in Figure 3.1a and therefore deleted in Figure 3.1c). The inward edges of a pointwise dominated node are re-directed to the dominating node since it offers better value (e.g., inward arcs of n_1 labeled by a, b, and c in Figure 3.1b are redirected to n_4 in Figure 3.1c). The controller resulting from this policy improvement step is guaranteed to offer higher value at all belief states. Unfortunately, up to $|\mathcal{A}||\mathcal{N}|^{|\mathcal{Z}|}$ new nodes may be added with each DP backup, so the size of the controller quickly becomes intractable in many POMDPs.

Note that policy iteration only prunes nodes that are pointwise dominated, rather than all dominated nodes (i.e., pointwise and LP-dominated). This is because the algorithm is designed to produce controllers with deterministic observation strategies. A pointwise-dominated node can be safely pruned since its inward arcs are redirected to the dominating node (which has value at least as high as the dominated node at *each state*). In contrast, a node jointly dominated by several nodes (e.g., n_2 in Figure 3.1(b) is jointly dominated by n_3 and n_4) cannot be pruned without its inward arcs being redirected to different nodes depending on the current belief state.

This problem can be circumvented by allowing stochastic observation strategies.

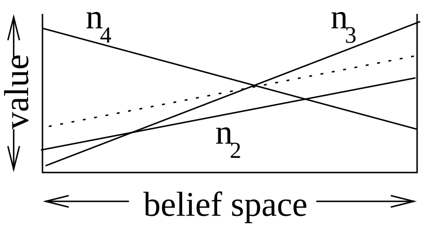


Figure 3.2: Example of a convex combination (dotted line) of n_3 and n_4 that dominates n_2.

Hence, we consider *stochastic finite state controllers*, which were first explored by Platzman, as reported by Littman in his PhD thesis [69]. Recall that an observation strategy (Section 2.2.4) is a mapping indicating the next node n' that will be visited after receiving observation z in node n. We revise the notion of observation strategy by allowing a distribution over successor nodes n' for each n, z-pair (specifically, $\sigma(n, z, n') = \Pr(n'|n, z)$). Intuitively, stochastic observation strategies can be viewed as selecting *mixtures* or *convex combinations* of successor nodes. If a stochastic strategy is chosen carefully, the corresponding convex combination of dominating nodes may pointwise dominate the node we would like to prune. In Figure 3.2, n_2 is dominated by n_3 and n_4 together (but neither of them alone). Convex combinations of n_3 and n_4 correspond to all lines that pass through the intersection of n_3 and n_4. The dotted line illustrates one convex combination of n_3 and n_4 that pointwise dominates n_2. Consequently, n_2 can be safely removed and its inward arcs re-directed to reflect this convex combination by setting the observation probabilities accordingly. In general, when a node is jointly dominated by a group of nodes, we show in Theorem 1 that there always exists a pointwise-dominating convex combination of this group.

Theorem 1 *The value function α_n of a node n is jointly dominated by the value functions $\alpha_{n_1}, \ldots, \alpha_{n_k}$ of nodes $n_1, \ldots, n_k$ if and only if there is a convex combination $\sum_i c_i \alpha_{n_i}$ that dominates α_n.*

Proof: α_n is dominated by $\alpha_{n_1}, \ldots, \alpha_{n_k}$ when the objective of the LP-dominance test described in Table 2.3 is positive. The LP in Table 2.3 finds the belief state b that minimizes the difference between $b \cdot \alpha_n$ and the max of $b \cdot \alpha_{n_1}, \ldots, b \cdot \alpha_{n_k}$. It turns out that the dual LP (Table 3.1) finds the most dominating convex combination parallel to α_n. Since the dual has positive objective value when the primal does, the theorem

$$\begin{array}{ll} \max\limits_{\{\delta,\, c_i\}} & \delta \\ \text{s.t.} & \alpha_n(s) + \delta \leq \sum_i c_i \alpha_{n_i}(s), \quad \forall s \in \mathcal{S} \\ & \sum_i c_i = 1 \\ & c_i \geq 0, \quad \forall i \end{array}$$

Table 3.1: Dual LP: convex combination $\sum_i c_i\alpha_{n_i}$ dominates α_n when $\delta \geq 0$.

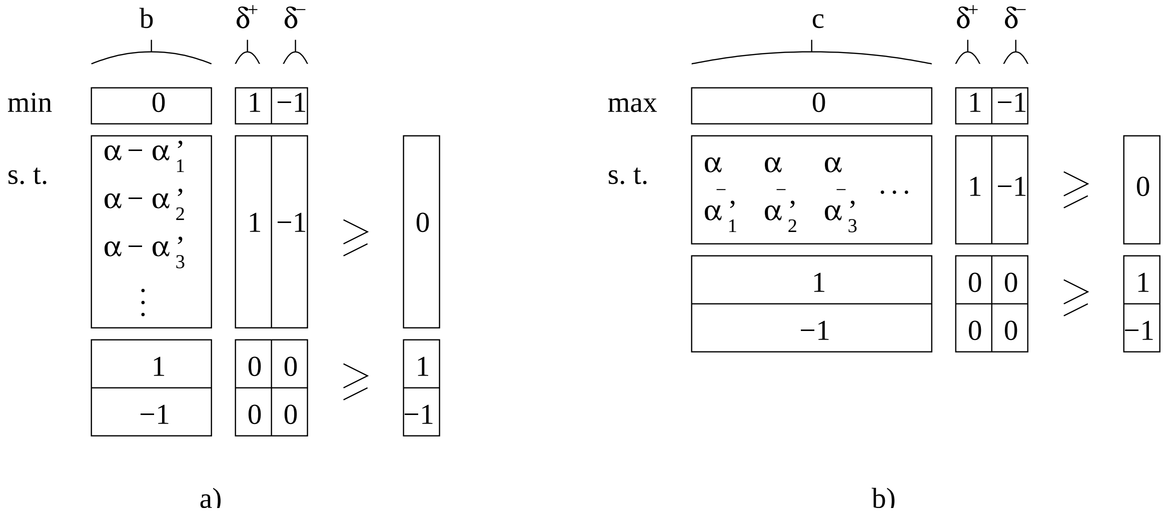


Table 3.2: a) Canonical block matrix form of the LP in Table 2.3. b) Canonical block matrix form of the LP in Table 3.1.

follows.

Recall from the duality theory [28] that the primal LP $\min_x cx$ subject to $Ax \geq b, x \geq 0$ is equivalent to (i.e., has the same objective as) the dual LP $\max_y b^T y$ subject to $A^T y \leq c^T, y \geq 0$. To help visualize the duality of the LPs in Tables 2.3 and 3.1, they have been respectively rewritten in Table 3.2a and 3.2b in a canonical block matrix format. The LPs in Table 3.2 are obtained from the LPs in Tables 2.3 and 3.1 by "splitting" the variable δ in a positive component δ^+ and a negative component δ^- and by expressing the equalities $\sum_s b(s) = 1$ and $\sum_i c_i = 1$ as pairs of inequalities. ◀

As argued in the proof of Theorem 1, the LP in Table 3.1 gives us an algorithm to find the most dominating convex combination parallel to a dominated node. In summary, by considering stochastic controllers, we can extend PI to prune all dominated nodes (point-wise or LP-dominated) in the policy improvement step. This provides two advantages:

controllers can be made smaller while improving their decision quality.

3.2.2 Policy Iteration for Bounded Controllers

Pruning all dominated nodes certainly helps to keep the controller small, but it may still grow substantially with each DP backup. Several techniques described in Section 3.1 can be used to perform *partial* DP backups that effectively bound the number of nodes. In particular, we can further reduce the number of nodes by pruning those that only marginally improve the value function [36]. Alternatively, we can generate only a subset of the nodes using Cheng's algorithm [26], the Witness algorithm [55] or point-based backups [147]. When the new nodes generated by a partial DP backup jointly dominate all the old nodes, then the old nodes can be pruned and their inward arcs re-directed to dominating convex combinations of the new nodes. The resulting controller is guaranteed to have value at least as great for the entire belief space since all the old nodes were replaced by *dominating* convex combinations of the new nodes. When there is an old node that isn't dominated by any convex combination of the new nodes, then that old node can still be pruned and its inward arcs re-directed so some good but non-dominating convex combination of the new nodes. As a result, the new controller may have lesser value at some belief states. In general, partial DP backups using any of the techniques described in Section 3.1 cannot guarantee monotonic improvement of the controller since the new nodes do not necessarily jointly dominate all the old nodes. We now propose a new algorithm called *bounded policy iteration (BPI)* that guarantees monotonic value improvement at all belief states while keeping the number of nodes fixed.

BPI considers one node at a time and tries to improve it while keeping all other nodes fixed. Improvement is achieved by replacing each node by a good convex combination of the nodes normally created by a DP backup, but without actually performing the DP backup. Since the backed up value function must dominate the controller's current value function, then by Theorem 1 there must exist a convex combination of the new nodes that pointwise dominates each old node of the controller. We can directly compute such convex combinations with the LP in Table 3.3. This LP has $|\mathcal{A}||\mathcal{N}|^{|\mathcal{Z}|}$ variables $c_{a,n_1,n_2,\ldots,n_{|\mathcal{Z}|}}$ corresponding to the probabilities of the convex combination as well as the δ variable measuring the value improvement. More precisely, $c_{a,n_1,n_2,\ldots,n_{|\mathcal{Z}|}}$ denotes the probability of the strategy where action a is chosen, node n_1 is reached when the first observation is made, node n_2 is reached when the second observation is made, ..., node

$$\begin{array}{ll} \max\limits_{\{\delta,\, c_{a,n_1,n_2,\ldots,n_{|\mathcal{Z}|}}\}} & \delta \\ \text{s.t.} & \alpha_n(s) + \delta \leq \sum_{a,n_1,n_2,\ldots,n_{|\mathcal{Z}|}} c_{a,n_1,n_2,\ldots,n_{|\mathcal{Z}|}} [R(s,a) + \\ & \qquad \gamma \sum_{s',z} \Pr(s'|s,a) \Pr(z|s',a) \alpha_{n_z}(s')],\ \forall s \in \mathcal{S} \\ & \sum_{a,n_1,n_2,\ldots,n_{|\mathcal{Z}|}} c_{a,n_1,n_2,\ldots,n_{|\mathcal{Z}|}} = 1 \\ & c_{a,n_1,n_2,\ldots,n_{|\mathcal{Z}|}} \geq 0,\ \forall a, n_1, n_2, \ldots, n_{|\mathcal{Z}|} \end{array}$$

Table 3.3: Naive LP to find a convex combination of new nodes that dominate n.

$n_{|\mathcal{Z}|}$ is reached when the last observation is made. Since there are $|\mathcal{A}||\mathcal{N}|^{|\mathcal{Z}|}$ possible action and observation strategies, that's why there are as many convex combination variables.

We can significantly reduce the number of variables by pushing the convex combination variables as far as possible into the DP backup, resulting in the LP shown in Table 3.4. The key here is to realize that we can aggregate many variables. For instance, when computing the expected immediate reward, we can aggregate together all the strategies that pick the same action, since the immediate expected reward doesn't depend on the nodes reached after each observation. Let $c_a = \sum_{n_1,n_2,\ldots,n_{|\mathcal{Z}|}} c_{a,n_1,n_2,\ldots,n_{|\mathcal{Z}|}}$ denote the aggregate probability of the observation strategies that pick action a, then

$$\sum_{n_1,n_2,\ldots,n_{|\mathcal{Z}|}} c_{a,n_1,n_2,\ldots,n_{|\mathcal{Z}|}} R(s,a) = c_a R(s,a)\ .$$

Similarly, when computing the expected future value of reaching a node n_z after executing action a and making observation z, we can aggregate together all the strategies that execute the same action and that pick the same node for observation z. Let $c_{a,n_z} = \sum_{n_1,\ldots,n_{z-1},n_{z+1},\ldots,n_{|\mathcal{Z}|}} c_{a,n_1,n_2,\ldots,n_{|\mathcal{Z}|}}$ denote the aggregate probability of the observation strategies that execute a and reach node n_z when observing z, then

$$\sum_{n_1,n_2,\ldots,\ldots,n_{|\mathcal{Z}|}} c_{a,n_1,n_2,\ldots,n_{|\mathcal{Z}|}} \Pr(s'|s,a) \Pr(z|s',a) \alpha_{n_z}(s') = \sum_{n_z} c_{a,n_z} \Pr(s'|s,a) \Pr(z|s',a) \alpha_{n_z}(s')\ .$$

By using the aggregate probability variables c_a and c_{a,n_z}, the LP in Table 3.4 is much more efficient since it has only $|\mathcal{A}||\mathcal{Z}||\mathcal{N}| + |\mathcal{A}| + 1$ variables.[1] Furthermore, once the

[1]Actually, we don't need the c_a variables since they can be derived from the c_{a,n_z} variables by summing out n_z, so the number of variables can be reduced to $|\mathcal{A}||\mathcal{Z}||\mathcal{N}| + 1$.

$$\begin{aligned}
\max_{\{\delta,\, c_a,\, c_{a,n_z}\}} \quad & \delta \\
\text{s.t.} \quad & \alpha_n(s) + \delta \leq \textstyle\sum_a [c_a R(s,a) + \\
& \qquad \gamma \textstyle\sum_{s',z} \Pr(s'|s,a) \Pr(z|s',a) \sum_{n_z} c_{a,n_z} \alpha_{n_z}(s')], \forall s \\
& \textstyle\sum_a c_a = 1 \\
& \textstyle\sum_{n_z} c_{a,n_z} = c_a, \ \forall a \\
& c_a \geq 0, \ \forall a \\
& c_{a,n_z} \geq 0, \ \forall a, z
\end{aligned}$$

Table 3.4: Efficient LP to find a convex combination of new nodes that dominate n.

LP in Table 3.4 is solved the values of c_a and c_{a,n_z} can be used to specify an improved *stochastic* action and observation strategy: pick action a with probability c_a and for each observation z, pick node n_z with probability c_{a,n_z}.

To summarize, BPI alternates between policy evaluation and policy improvement as in classic PI, but the policy improvement step simply tries to improve each node by solving the LP in Table 3.4 (without performing any DP backup). The c_a and c_{a,n_z} variables are used to set the probabilistic action and observation strategies of the new improved node. The value of the improved node is at least δ higher than the original node for the entire belief space. Note that the policy improvement step of BPI is much more efficient than the one for classical PI since linear programming has polynomial time complexity whereas a DP backup takes exponential time and space in the worst case. Overall, BPI runs in polynomial time and space complexity since the size of the controller remains constant, the policy improvement step solves linear programs and the policy evaluation step solves linear equations. In contrast, classical PI tends to grow exponentially large controllers over time.

3.2.3 Local Optima

BPI is a simple, efficient alternative to classic PI that monotonically improves an FSC while keeping its size constant. The value of the controller will increase until a local optimum is reached, that is when none of the existing nodes can be be improved with the LP in Table 3.4. We now characterize BPI's local optima and propose a method to escape them.

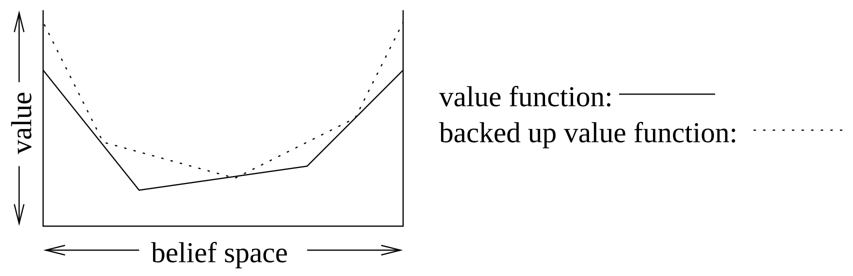


Figure 3.3: BPI local optimum: current value function is tangent to the backed up value function.

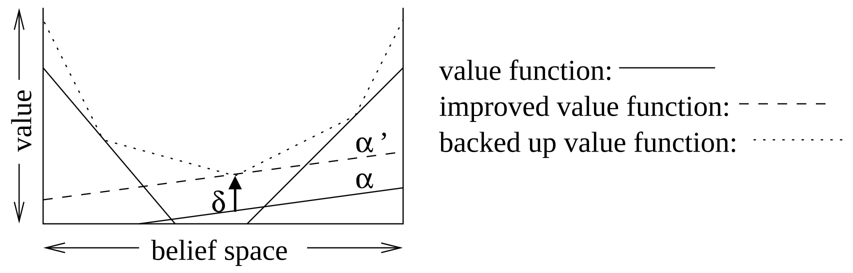


Figure 3.4: BPI can improve vector α by an amount δ to obtain vector α'.

Characterization

Theorem 2 below gives a necessary and sufficient condition characterizing BPI's local optima. Intuitively, a controller is a local optimum when each α-vector touches from below, or is *tangent to*, the controller's backed up value function (see Figure 3.3). Recall that the backed up value function V^{k+1} is the result of a DP backup (Equation 2.4) applied to the current value function V^k. Since DP backups monotonically improve the value of controllers, we know that V^{k+1} is at least as high as V^k. Furthermore, BPI improves a node by computing a best convex combination of backed up α-vectors, which is equivalent to "pushing" upward its α-vector until it is tangent to the backed up value function. Hence, a controller is a local optimum when each α-vector is already tangent to the backed up value function, leaving no room for improvement. Note that some α-vectors may be equal to linear portions of the backed up value function, but each α-vector is only guaranteed to touch the backed up value function in one belief state.

Theorem 2 *BPI has converged to a local optimum if and only if each node's α-vector is tangent to the backed up value function.*

Proof: Let's prove by contradiction that in BPI's local optima, each node's α-vector is tangent to the backed up value function. Suppose this is not the case, then there must be a gap between some α-vector and the backed up value function (e.g., vector α in Figure 3.4 is not tangent to the backed up value function). When such a gap exists, then the LP in Table 3.4 can be used to find a better convex combination that improves the value by the same amount as the gap (e.g., vector α' is found in Figure 3.4). Hence, BPI is able to improve the controller, which is not a local optimum.

Conversely, when each node's α-vector is tangent to the backed up value function (e.g., Figure 3.3), one can also show that BPI has reached a local optimum. Since the space of convex combinations of backed up α-vectors consists of hyperplanes that are below or tangent to the backed up value function, BPI can only improve a controller by finding new vectors that are below or tangent to the backed up value function. Furthermore, the LP in Table 3.4 improves a vector uniformly by pushing it upward by an equal amount for the entire belief space. Hence, as soon as a vector touches the backed up value function, BPI cannot improve it further. ◀

Interestingly, tangency is also a necessary (but not sufficient) condition for GA's local optima (see Corollary 1 below). Intuitively, GA searches for the direction of steepest monotonic improvement. It reaches a local minimum only when there is no direction providing monotonic improvement. If there is a gap between some α-vector and the backed up value function, then GA will find a direction of monotonic improvement. Hence it is necessary that all α-vectors be tangent to the backed up value function for GA to be stuck in a local optimum.

Corollary 1 *If GA has converged to a local optimum, then the α-vector of each node reachable from the initial belief state is tangent to the backed up value function.*

Proof: GA seeks to monotonically improve a controller in the direction of steepest ascent. The LP of Table 3.4 also seeks a monotonically improving direction. Thus, if BPI can improve a controller by finding a direction of improvement using the LP of Table 3.4, then GA will also find it or will find a steeper one. Conversely, when a controller is a local optimum for GA, then there is no monotonic improvement possible in any direction. Since BPI can only improve a controller by following a direction of monotonic improvement, GA's local optima are a subset of BPI's local optima. Thus, tangency is a necessary, but not sufficient, condition of GA's local optima. ◀

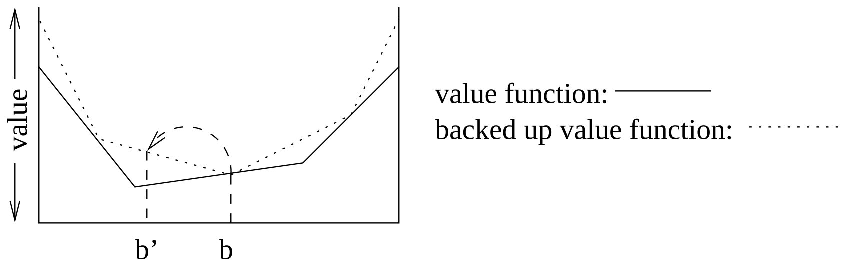


Figure 3.5: Example to escape a local optimum.

In the proof of Corollary 1, we argued that GA's local optima are a subset of BPI's local optima. This suggests that BPI is inferior to GA since it can be trapped by more local optima than GA. However, we will describe in the next section a simple technique that allows BPI to easily escape from many local optima.

Escape Technique

The tangency condition characterizing local optima can be used to design an effective escape method for BPI. It essentially tells us that tangent points are "bottlenecks" for further policy improvement. If we could improve the value at the belief states where tangency occurs, then we could break out of the local optimum. A simple method for doing so consists of a one-step lookahead search from the tangent belief states. In Figure 3.5, suppose that belief state b' can be reached in one step from tangent belief state b, and that the backed up value function improves the current value of b'. If we add a node to the controller that maximizes the value of b', its improved value can subsequently be backed up to the tangent belief state b, breaking out of the local optimum.

The escape technique is summarized as follows: perform a one-step lookahead search from each tangent belief state; when a reachable belief state can be improved, add a new node to the controller that maximizes that belief state's value. Interestingly, when no reachable belief state can be improved, the policy must be optimal at the tangent belief states.

Theorem 3 *If the backed up value function does not improve the value of any belief state reachable in one step from any tangent belief state, then the policy is optimal at the tangent belief states.*

$$\begin{array}{ll}
\min\limits_{\{\delta,\, b,\, Q_{a,z}\}} & \delta \\
\text{s.t.} & b \cdot \alpha + \delta \leq b \cdot \sum_z Q_{a,z},\ \forall a \\
 & b \cdot Q_{a,z} \geq b \cdot [r_a/|\mathcal{Z}| + \gamma T^{a,z} \alpha'],\ \forall a, z, \alpha' \\
 & \sum_s b(s) = 1 \\
 & b(s) \geq 0,\ \forall s
\end{array}$$

Table 3.5: Efficient LP to find a tangent witness point. This is the dual of the LP in Table 3.4.

Proof: By definition, belief states for which the backed up value function provides no improvement are tangent belief states. Hence, when all belief states reachable in one step are themselves tangent belief states, then the set of tangent belief states is closed under every policy. Since there is no possibility of improvement, the current policy must be optimal at the tangent belief states. ◄

Although Theorem 3 guarantees an optimal solution only at the tangent belief states, in practice, they rarely form a proper subset of the belief space. Note also that the escape algorithm assumes knowledge of the tangent belief states. Fortunately, the solution to the dual of the LP in Table 3.4 is a tangent belief state. Table 3.5 describes this dual LP. Intuitively, the dual finds the belief state b for which α has the smallest gap δ with the backed up value function. This belief point will be the first one to "touch" the backed up value function when improving α. Hence, it is necessarily a tangent point. Since most commercial LP solvers return both the solution of the primal and the dual, a tangent belief state is readily available for each node when solving the LP in Table 3.4. In other words, it is sufficient to solve only the LP in Table 3.4 (there is no need to solve the dual in Table 3.5).

3.2.4 Improvement Bias

In practice, we often wish to find a policy suitable for a given initial belief state. Since the reachable belief states often form a small subset of the entire belief space, it is generally possible to construct much smaller policies tailored to the set of reachable belief states. We now describe a simple way to bias BPI's efforts toward the set of reachable belief states.

Recall that the LP in Table 3.4 optimizes the parameters of a node to uniformly improve its value at all belief states. We can modify that LP so that improvement is weighted by the discounted *occupancy distribution* induced by the current policy. The occupancy distribution $o(s)$ of a policy is a probability distribution that indicates the relative frequency with which each state is visited. We can compute the occupancy distribution $o(s)$ by adding together all the reachable belief states (weighted by their probability of being reached and the discount factor γ) and re-normalizing:

$$o(s) = k \sum_{t=0}^{\infty} \gamma^t \sum_i Pr(b_i|t) b_i(s) \ .$$

Here, $Pr(b_i|t)$ is the probability that belief state b_i is reached at time step t. In that sense, the occupancy distribution constitutes a suitable aggregation of the reachable belief states that can be used by BPI to bias the improvement. In Table 3.6, we obtain a new LP by weighting the improvement $\delta_{n,s}$ by $o(n, s)$. Here, $o(n, s)$ is the occupancy distribution parameterized by a node n since we are improving one node at a time. Intuitively, it represents the relative (discounted) frequency of the states visited when the controller is in node n. In general, $o(s) = \sum_n o(n, s)$. We can compute an *unnormalized* version of $o(n, s)$ with a system of linear equations (similar to the system to evaluate a policy) by summing all discounted future belief states:

$$o(n', s') = b_0(n', s') + \gamma \sum_{s,a,z,n} o(n, s) \Pr(a|n) \Pr(z|a, s) \Pr(n'|n, a, z) \quad \forall s', n' \ . \tag{3.2}$$

Note that $b_0(n, s)$ is the probability that the world is in state s and the controller is in node n at time step 0. We can use the unnormalized $o(n, s)$ in Table 3.6 since normalizing doesn't affect the objective. Hence, the LP in Table 3.6 maximizes the expected improvement of a node's occupancy distribution while making sure that the value doesn't decrease for any belief state. This LP has the effect of greedily improving the current policy. Such a myopic strategy may not yield the best node improvement; however, it tends to perform well and when the reachable belief region is small, it often finds controllers with fewer nodes (yet similar value) than those found with the uniform improvement strategy of the LP in Table 3.4.

In practice, the expected improvement of a node's occupancy distribution is often limited by the constraints that prevent any decrease in value at any belief state (e.g., first set of constraints in Table 3.6). Although these constraints are necessary to ensure monotonic improvement, relaxing them a bit by allowing a slight decrease in value at some

$$\begin{array}{ll}
\max\limits_{\{\delta_{n,s},\, c_a,\, c_{a,n_z}\}} & \sum_{s,n} o(n,s)\delta_{n,s} \\
\text{s.t.} & \alpha_n(s) + \delta_{n,s} \leq \sum_a [c_a R(s,a) + \\
 & \qquad \gamma \sum_{s',z} \Pr(s'|s,a) \Pr(z|s',a) \sum_{n_z} c_{a,n_z} \alpha_{n_z}(s')], \forall s \\
 & \sum_a c_a = 1 \\
 & \sum_{n_z} c_{a,n_z} = c_a, \ \forall a \\
 & c_a \geq 0, \ \forall a \\
 & c_{a,n_z} \geq 0, \ \forall a, z
\end{array}$$

Table 3.6: LP to maximize the expected improvement of the occupancy distribution of a node.

$$\begin{array}{ll}
\max\limits_{\{\delta_{n,s},\, c_a,\, c_{a,n_z}\}} & \sum_{s,n} o(n,s)\delta_{n,s} \\
\text{s.t.} & \alpha_n(s) + \delta_{n,s} \leq \epsilon + \sum_a [c_a R(s,a) + \\
 & \qquad \gamma \sum_{s',z} \Pr(s'|s,a) \Pr(z|s',a) \sum_{n_z} c_{a,n_z} \alpha_{n_z}(s')], \forall s \\
 & \sum_a c_a = 1 \\
 & \sum_{n_z} c_{a,n_z} = c_a \ \forall a \\
 & c_a \geq 0 \ \forall a \\
 & c_{a,n_z} \geq 0 \ \forall a, z
\end{array}$$

Table 3.7: LP to maximize the expected improvement of the occupancy distribution of a node while allowing a slight decrease in value of at most ϵ at some belief states.

belief states can often further boost the occupancy improvement. Since improvement at the occupancy distribution is more important than elsewhere, such a strategy tends to further improve BPI's policy search. Table 3.7 describes a modified version of the LP that allows value to decrease by some small ϵ. Setting ϵ to some small fraction of the reward span such as $\epsilon = (\max_{a,s} R(s,a) - \min_{a,s} R(s,a))/(400(1-\gamma))$ tends to give good results.

When a node of the controller is never reached from the initial belief state, its occupancy distribution is zero. In that case, the objectives of the LPs in Tables 3.6 and 3.7 are trivially zero. So we must resort to the uniform improvement strategy of Table 3.4 or replace the occupancy distribution by some other distribution in Tables 3.6 and 3.7. In particular, we can replace the occupancy distribution by the belief state used at the time of creation of the node. Recall from Section 3.2.3 that the escape technique adds

```
PROCEDURE $BPI(controller, timebound)$
    repeat (controller growth loop)
        repeat (policy iteration loop)
            policy evaluation:
                solve linear system (Equation 2.12)
            policy improvement:
                improve each node by solving the LP in Table 3.4, 3.6 or 3.7
        until no improvement
        escape local optimum:
            find belief states reachable from tangent points in a one-step lookahead
            create $k$ new nodes that improve the value at some reachable belief states
    until no time left or no new node created by escape algorithm
END PROCEDURE
```

Table 3.8: Bounded Policy Iteration algorithm.

new nodes that maximize improvement at some belief states reachable from the tangent points. Hence, belief states reachable from tangent points provide an alternative set of belief states to the belief states reachable from the initial belief state.

3.2.5 Summary

Let's summarize BPI. Table 3.8 provides a concise description of the algorithm. BPI takes as input a controller, which can be initialized randomly or based on prior expert knowledge. It then alternates between policy evaluation and policy improvement until a local optimum is reached. At that point, new nodes are added to the controller. These nodes improve the value at some belief states reachable from the tangent points of the local optimum. BPI runs until convergence to a policy that is optimal at the tangent points or until time is up. Since the three steps (policy evaluation, policy improvement and escape step) run in polynomial time, BPI's running time is polynomial with respect to the number of actions $|\mathcal{A}|$, the number of observations $|\mathcal{Z}|$, the number of states $|\mathcal{S}|$ and the number of nodes $|\mathcal{N}|$ in the final controller.

Since real-world POMDPs often have some policies that are both very good and very small, BPI can often quickly find those policies by searching through classes of increasingly large controllers. Furthermore, when an initial belief state is available, it can often find a smaller policy tailored to the reachable belief region. As a result, BPI can effectively circumvent the complexity of policy and value function representations. BPI distinguishes itself by its slow growth of the controller size (cf. PI), its ability to avoid local optima (cf. GA), its efficiency (cf. B&B and SLS) and the fact that belief state monitoring is not required during policy execution (cf. PBVI, Perseus). In the following section, we compare BPI to existing algorithms on a suite of benchmark problems.

3.3 Experiments

This section presents some experiments to demonstrate BPI's ability to overcome the complexity of policy and value function representations. A first set of experiments (Section 3.3.1) verifies BPI's robustness to local optima. A second set of experiments demonstrates BPI's scalability on a suite of benchmark problems. Compared to existing algorithms, BPI is a robust, scalable algorithm.

3.3.1 Local Optima Robustness

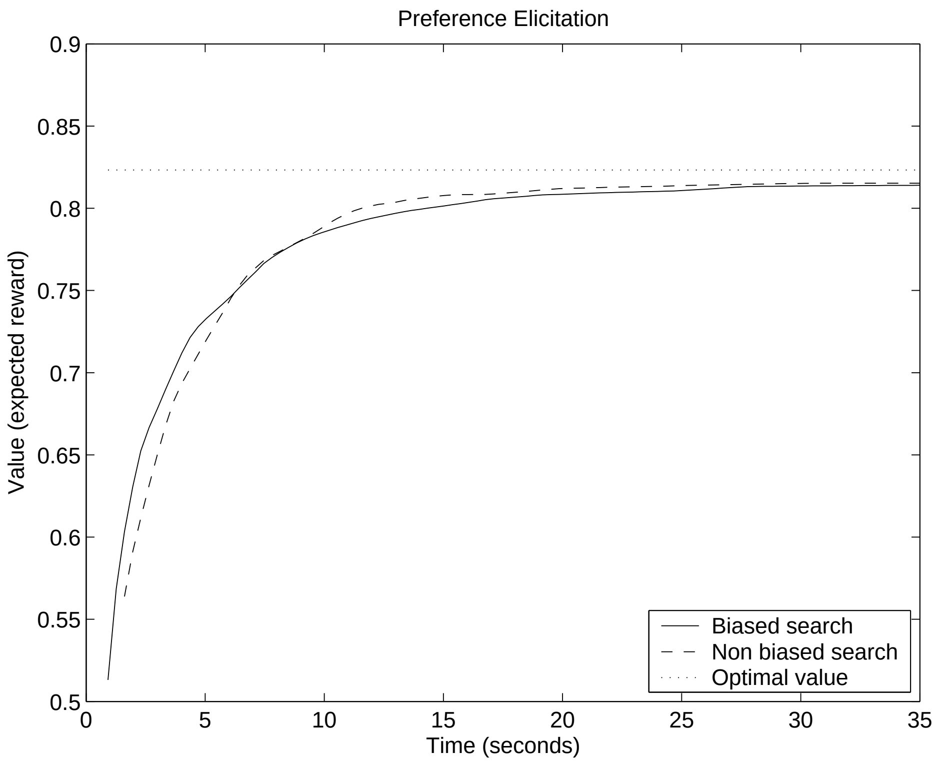
As argued earlier, in several domains it is common to find safe but far from optimal policies that form local optima. When the basin of attraction of those local optima is important, myopic techniques such as GA are ineffective because they easily get trapped. We now test BPI on a preference elicitation problem proposed by Boutilier [10] and a modified version of the heaven and hell problem proposed by Braziunas and Boutilier [20]. Both problems are known to have local optima into which GA will almost always fall.

In the preference elicitation problem, the task of the decision maker is to make a recommendation to a user. However, since the decision maker doesn't know the user's preferences, it can ask queries to gain some information before making a recommendation. The query process followed by the recommendation can be modeled by a POMDP [10] where the actions correspond to the possible queries and recommendations, the observations correspond to the user answers and the states correspond to the possible user preferences. The rewards consists of the cost associated with each query and the value (measured in terms of the user preferences) of the recommendation. The preference elicitation problem we consider is taken from earlier work by Braziunas and Boutilier [19, 20] (see Appendix A for more details). It has the following interesting property: the cost of a query is higher than the value of the information gained by a single query. As a result, myopic algorithms tend to get stuck in a local optimum that consists of making a blind recommendation without making any queries.

Figure 3.6 shows the expected rewards of the controllers found by BPI. Although BPI is deterministic, the expected rewards are averaged over 20 runs starting with different random initial controllers. The solid and dashed lines respectively correspond to the results achieved by BPI with and without biasing the search to the reachable belief region. There is no significant difference between the two because this problem is fairly small; however, we will see shortly that the biased search is much more effective on larger/more difficult problems. Braziunas and Boutilier [19, 20] report that the optimal value for this problem is 0.8233. Classic policy ietartion (PI), which is not subject to local optima, finds a 334-node controller that achieves this optimal value in 32 seconds. BPI finds near optimal (value of 0.81) controllers of 35 nodes in 35 seconds. Hence BPI is able to find a much smaller controller of similar quality than PI in a similar amount of time. In contrast, GA is not able to find policies with value higher than 0.6552 and belief-based stochastic local search (BBSLS) finds near optimal 22-node controllers in roughly 150 seconds [19, 20].

The second problem considered is Heaven and Hell [20]. In this problem, two nearly identical mazes offer a positive reward (heaven) and a negative reward (hell) in opposite locations. The agent starts in one of the two mazes, but doesn't know which. By visiting a priest and paying a small fee, it can find out which maze it is in and then reach heaven without any risk. Myopic algorithms tend to avoid the priest, gaining no knowledge about the world, and consequently avoiding reward-bearing states due to the risk of falling in hell (see Appendix A for more details).

Braziunas and Boutilier [20] report that the optimal policy has value 8.641. While PI should be able to find this optimal policy, the best controller found after 84,600 seconds (24 hours) of computation had 15,003 nodes and a value of 4.06 only. As for GA and BBSLS, the former gets stuck in a local optimum of value 0 and the latter finds 20-node controllers with average value of 7.65 in 5100 seconds. In contrast, BPI (see Figure 3.7) finds 60-node controllers of value 7.7 (averaged over 20 trials) in 35 seconds. Note how biasing BPI's search to reachable belief states (dotted line) produces smaller and yet better controllers than a search that ignores the initial belief state (dashed line).



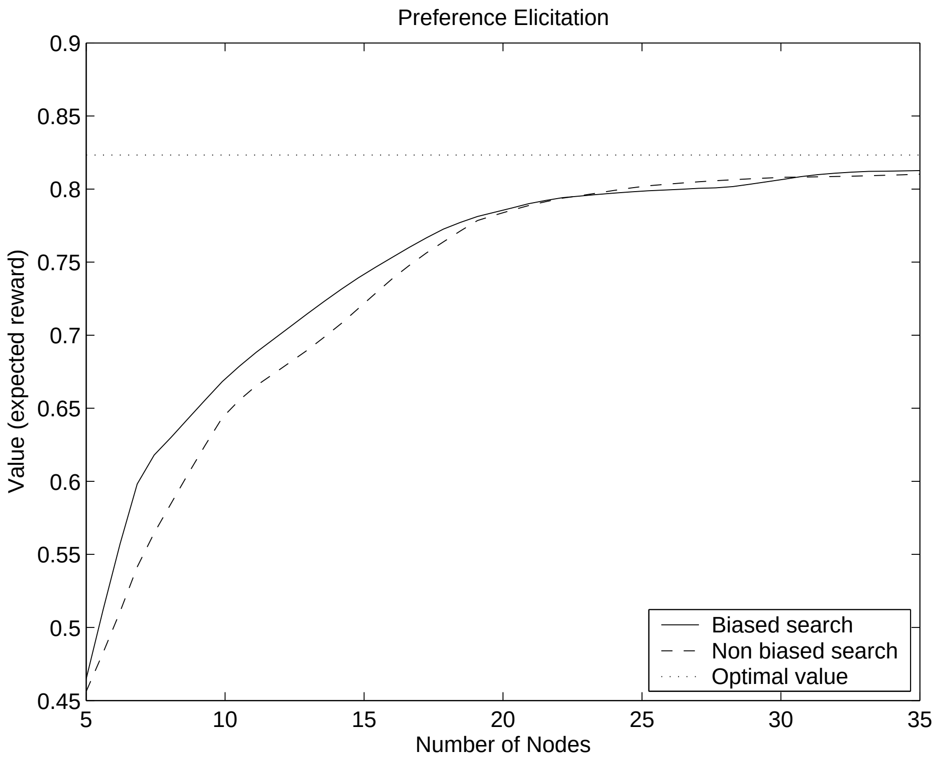
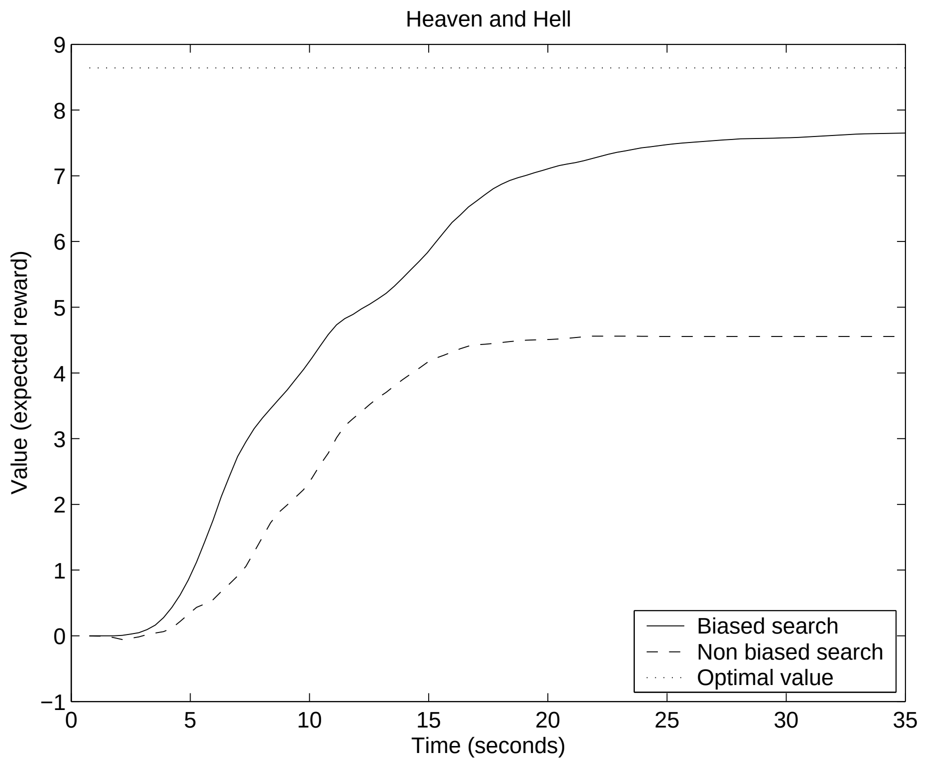


Figure 3.6: Preference elicitation: expected reward (averaged over 20 trials) earned by the controllers found by BPI with respect to time and number of nodes.



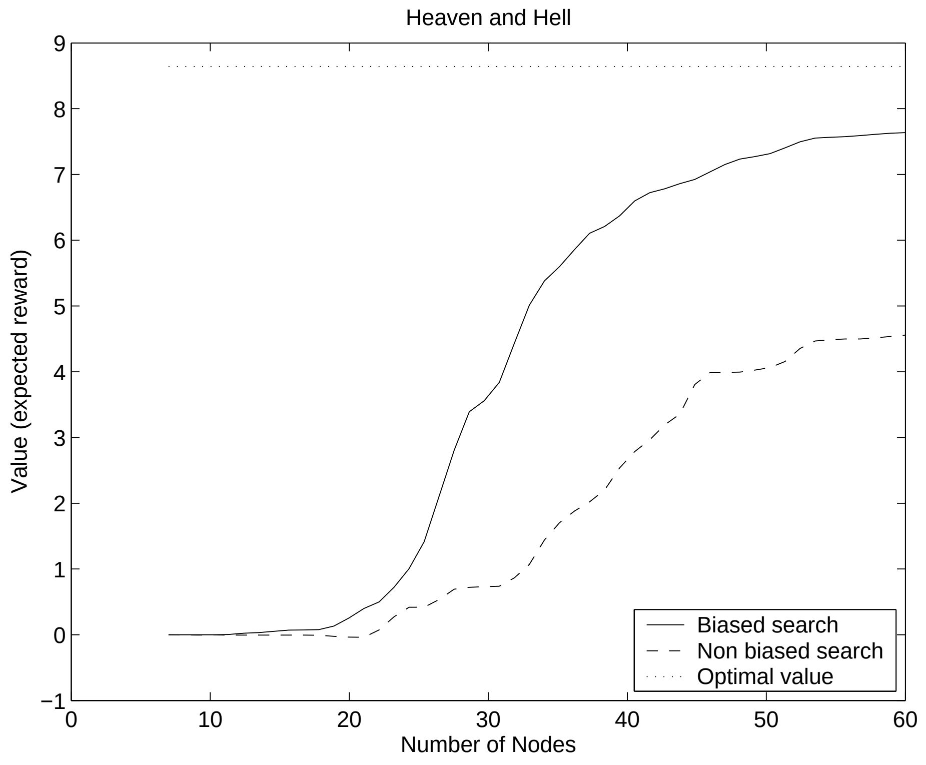


Figure 3.7: Heaven and hell: expected reward (averaged over 20 trials) earned by the controllers found by BPI with respect to time and number of nodes.

Problems	$\|\mathcal{S}\|$	$\|\mathcal{O}\|$	$\|\mathcal{A}\|$
tiger-grid	33	17	5
hallway	57	21	5
hallway2	89	17	5
tag-avoid	870	30	5

Table 3.9: Problem properties.

3.3.2 Scalability

The experiments of the previous section clearly demonstrate BPI's ability to avoid some important local optima. In this section we compare BPI to state-of-the-art algorithms on 4 additional benchmark problems: tiger-grid, hallway, hallway2 and tag-avoid. Although these problems are not known to have important suboptimal local optima, they are too difficult for the classic POMDP algorithms and thus have been used extensively to test the scalability of recent POMDP algorithms. All four problems are maze problems where a robot must perform a navigation task. The tiger-grid, hallway, hallway2 problems were introduced by Littman et al. [67] (available at `http://www.pomdp.org/pomdp/examples/index.shtml`) and the tasks consist of reaching a designated goal as quickly as possible. The tag-avoid problem was introduced by Pineau et al. [106] (available at `http://www-2.cs.cmu.edu/~jpineau/`) and the task consists of tagging an opponent robot that is trying to escape. The size of the state, action and observation spaces of each problem are given in Table 3.9. The hardness of those problems depends on the complexity of the optimal policy/value function. In the worst case, the number of nodes/α-vector in the optimal solution is $|\mathcal{A}|^{|\mathcal{Z}|^h}$ (where h is the horizon). The optimal solution of those problems is currently unknown; however, as we will see (Table 3.10, small yet very good solutions can be found for each of those problems. In general, it is very difficult to quantify a priori the policy complexity of a problem since the upper bound $|\mathcal{A}|^{|\mathcal{Z}|^h}$ often doesn't reflect the relatively small size of some near-optimal policies. A posteriori, we can more easily quantify policy complexity by looking at the size and quality (value) of the policy graphs found. Finding a reliable a priori measure of policy complexity is an open problem.

Figures 3.8, 3.9, 3.10 and 3.11 report the expected reward (averaged over 20 trials) earned by the controllers produced by BPI with respect to time and number of nodes. The results show that BPI can find relatively small controllers in a short period of time.

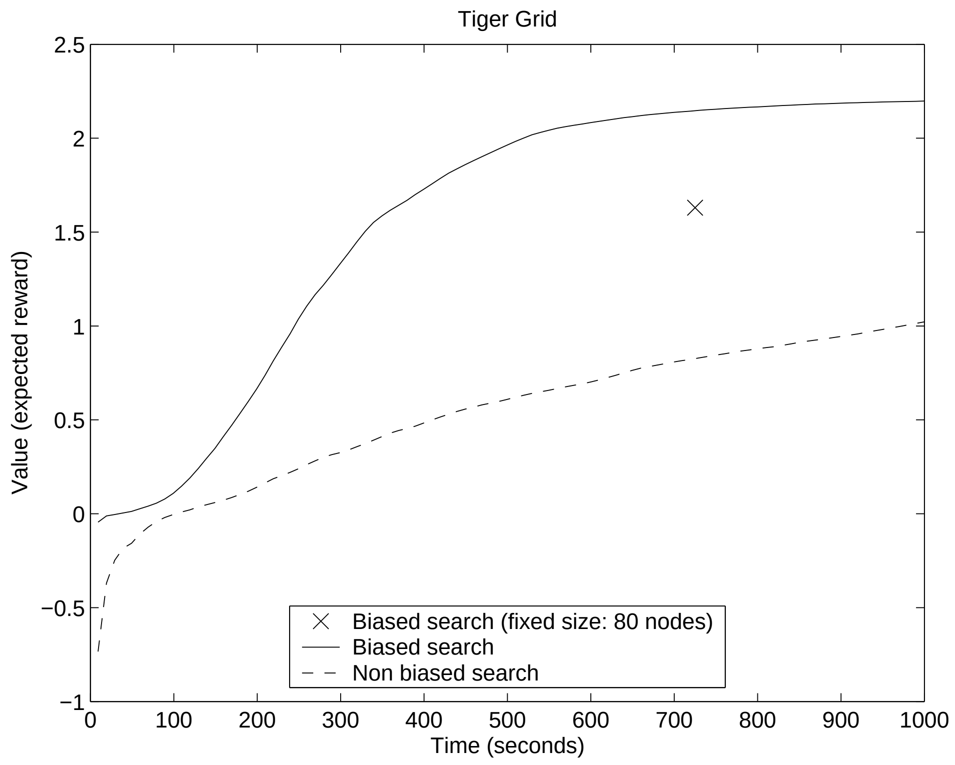
Note that biasing the search to the reachable belief region significantly helps BPI.

In Figures 3.8, 3.9, 3.10 and 3.11, we also report with an "x" the results obtained when fixing a priori the number of nodes (i.e., preventing BPI from growing the controller with the escape heuristic). For all four problems, when fixing the number of nodes, the local optimum controller that BPI converges to takes more time to find and has lower value than the one obtained by incrementally growing a controller. The smaller running time achieved by incrementally growing the controller is explained by the fact that early iterations can be done quickly when the controller is still small. The higher expected rewards achieved by growing the controller are explained by the fact that the escape heuristic greedily adds nodes that improve the controller in the reachable belief region. In contrast, when fixing the size of the controller a priori, the random initialization of the controller generates nodes that may not be relevant to the reachable belief region.

Table 3.10 summarizes BPI's results and compares them to state-of-the-art algorithms. PBVI refers to Pineau et al.'s original point-based value iteration [106]. Perseus refers to Spaan and Vlassis' randomized version of PBVI [142, 137, 136] that performs partial point-based backups. PBUA is another point-based algorithm proposed by Poon [109]. BBSLS is the belief-based stochastic local search algorithm proposed by Braziunas and Boutilier [20]. PI refers to Hansen's classic policy iteration algorithm [44, 45] with incremental pruning for the improvement step. Similarly, appPI refers to an approximate version of Hansen's policy iteration algorithm, where the number of α-vectors is bounded to 150 at each step of incremental pruning. Following Feng and Hansen's proposal [36], this is done by retaining the 150 α-vectors that contribute the most (i.e., α-vectors with the smallest objective when solving the LP-dominance test in Table 2.3) to the value function at each step and pruning all remaining α-vectors.

In order to make sure that the results for BPI are comparable to the ones previously reported for those algorithms, the rewards in Table 3.10 are accumulated over an infinite horizon for tag-avoid and tiger-grid, and until a goal state is reached for hallway and hallway2. In contrast, the results reported in Figures 3.8, 3.9, 3.10 and 3.11 assume an infinite horizon. Note also that time comparisons in Table 3.10 may not be adequate since the running times for Perseus, PBVI, PBUA and BBSLS are taken from the papers describing each algorithm and therefore important differences in hardware and software architecture may exist.[2] The size of the solution reported is the number of nodes in the

[2]The results for BPI, PI and appPI were computed using Matlab 6.5 and CPLEX 7.1 on a 3 GHz linux machine.



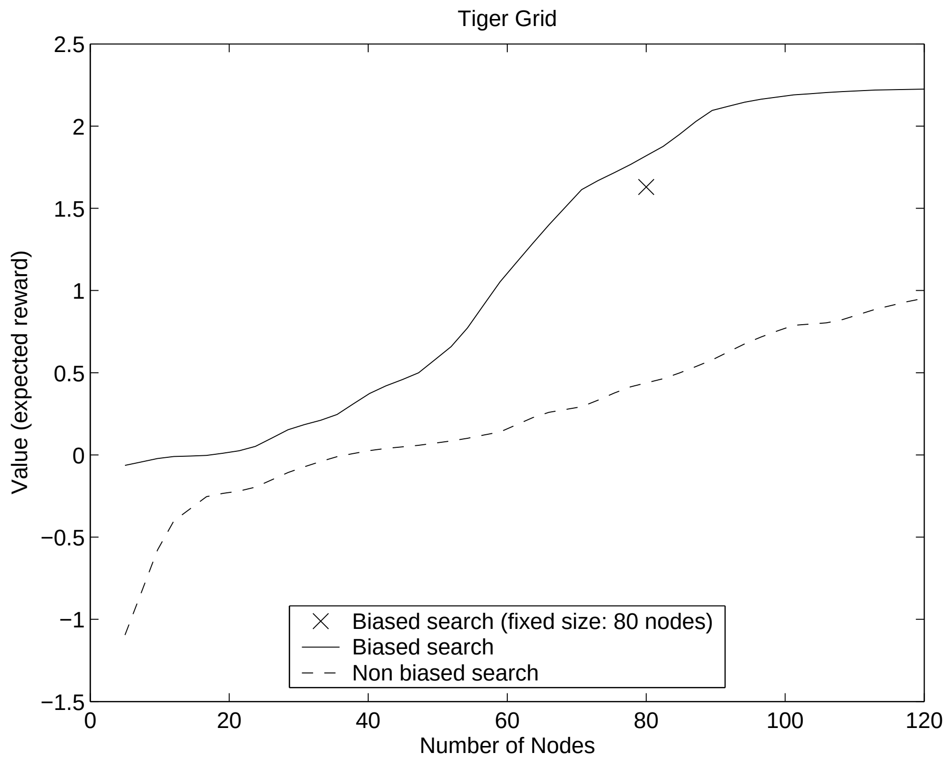
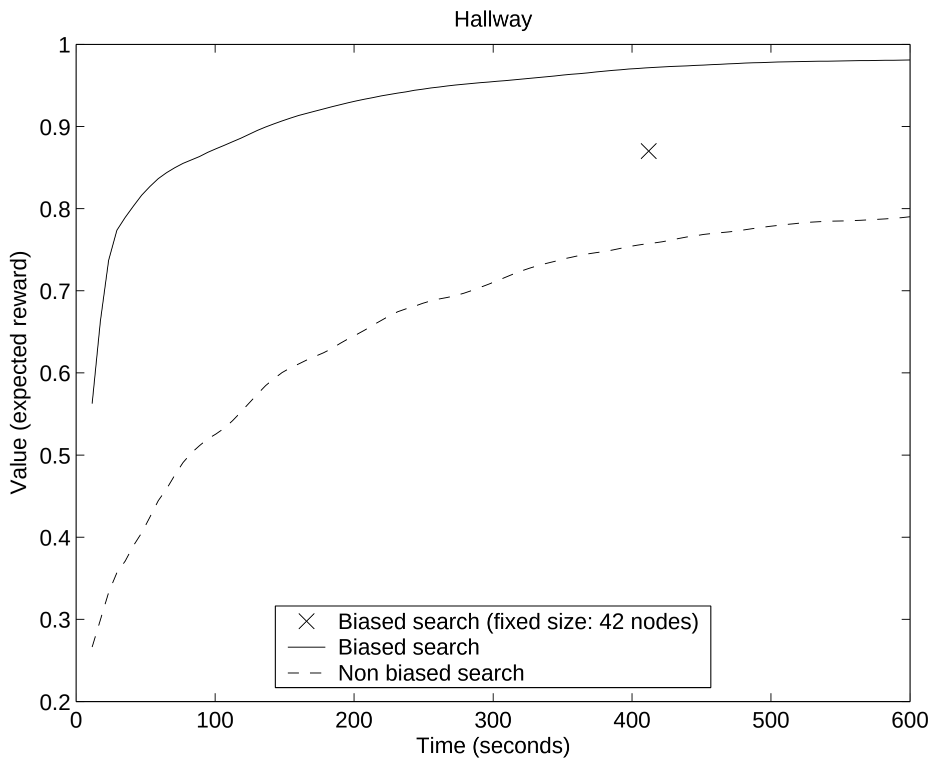


Figure 3.8: Tiger-grid: expected reward (averaged over 20 trials) earned by the controllers found by BPI with respect to time and number of nodes.



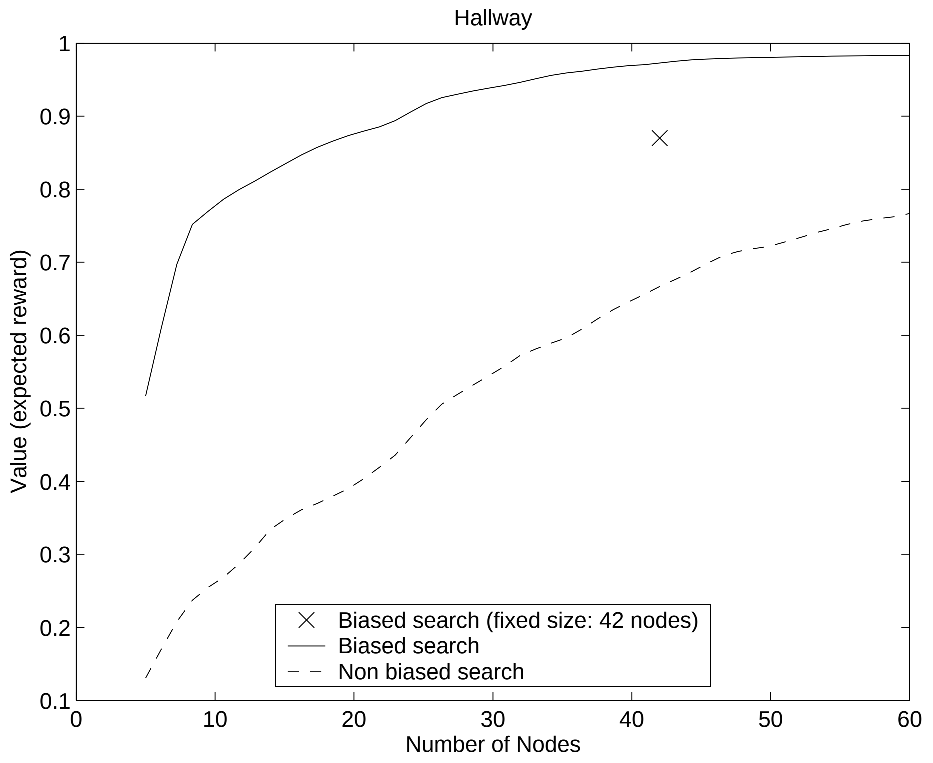
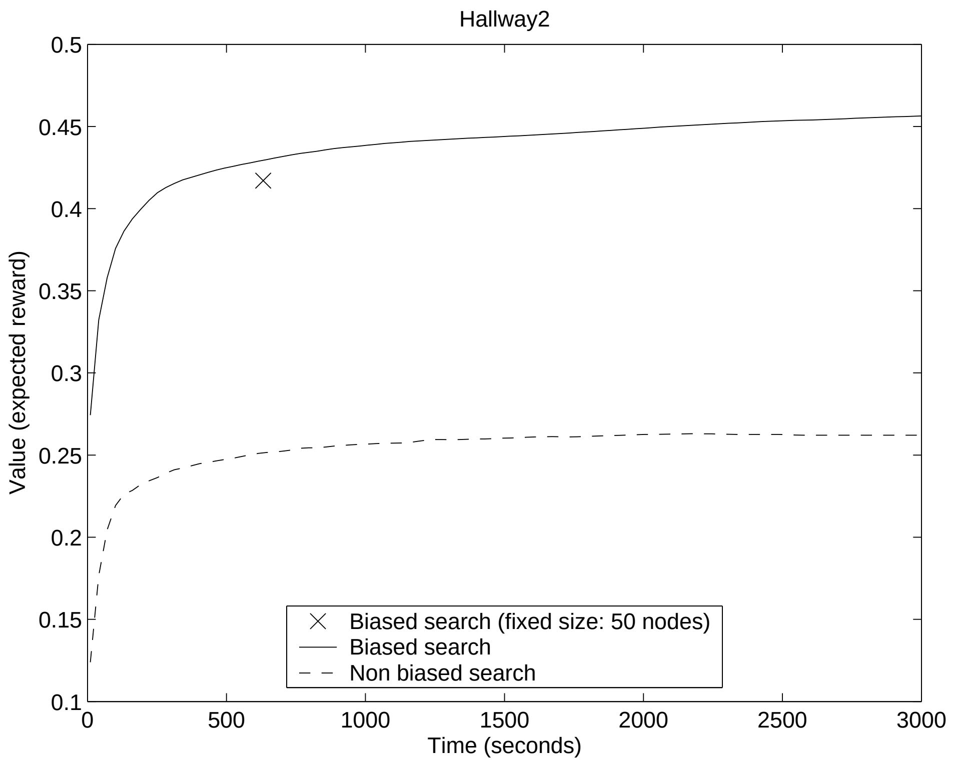


Figure 3.9: Hallway: expected reward (averaged over 20 trials) earned by the controllers found by BPI with respect to time and number of nodes.



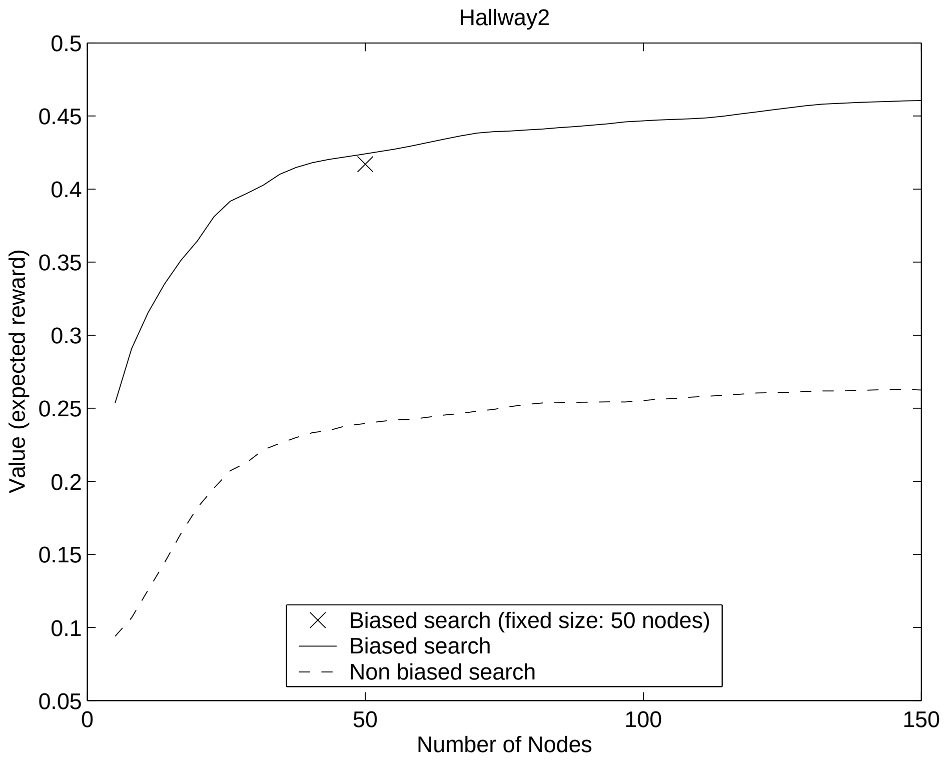
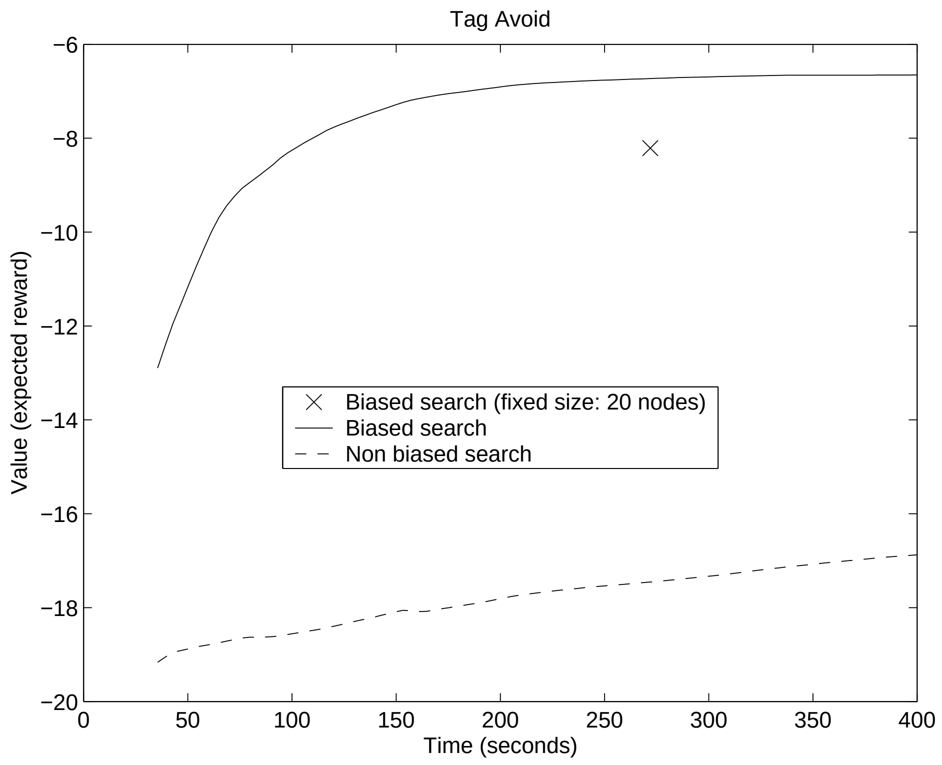


Figure 3.10: Hallway2: expected reward (averaged over 20 trials) earned by the controllers found by BPI with respect to time and number of nodes.



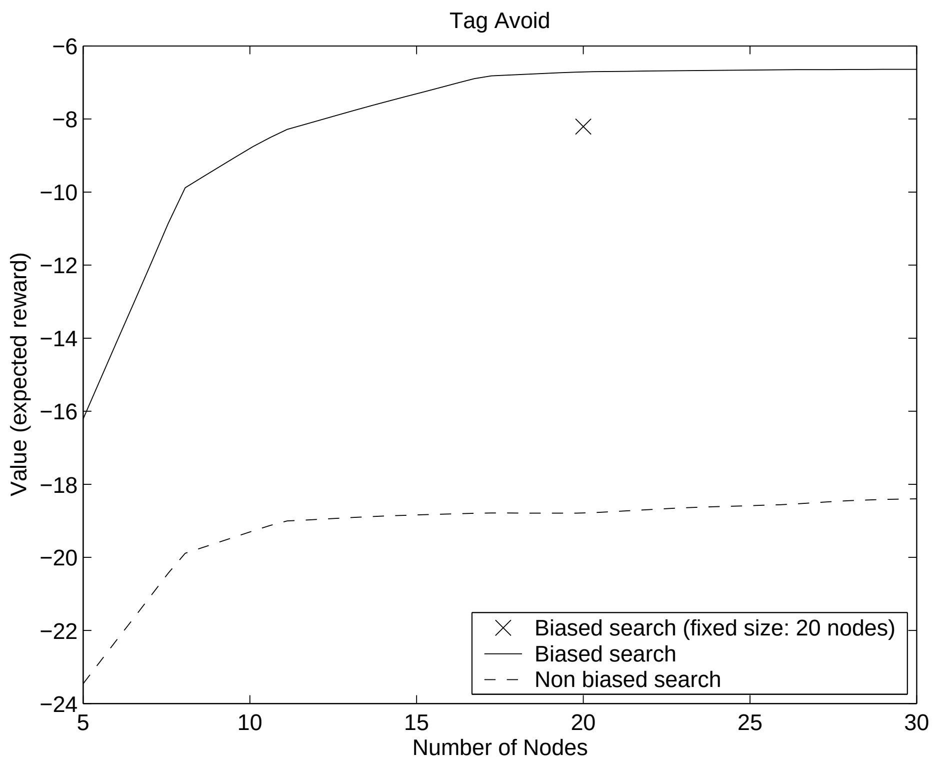


Figure 3.11: Tag-avoid: expected reward (averaged over 20 trials) earned by the controllers found by BPI with respect to time and number of nodes.

controller for BPI, BBSLS, PI and appPI, and the number of α-vectors for PBVI, Perseus and PBUA. Since there is one α-vector per node in a controller, these quantities are comparable. Note also that PI and appPI were stopped after 24 hours (86,400 seconds) so the results reported are for the controllers found in the last completed iteration. For tag-avoid and hallway2, the size of the controllers found by PI is artificially low because it only completed two iterations. In the third iteration, the number of nodes blew up significantly, preventing PI from completing the third iteration within 24 hours.

As mentioned earlier, the size of the components of a POMDP are often poor indicators of its *inherent* policy complexity. Unlike state space complexity, which can be easily quantified by the number of states, policy complexity is much harder to quantify a priori. While the number of actions and observations can be used to bound the size of the optimal policy graph (e.g., $|\mathcal{A}|^{|\mathcal{Z}|^h}$), there often exist relatively small policies of good quality. In fact, all problems in Table 3.10 possess small yet very good policies. Furthermore, even though tag-avoid has a larger number of observations than the other problems, smaller policies of high quality can be found for it. As a result, it can be solved in a similar amount of time to the other problems. Indeed, it is difficult to quantify a priori policy complexity.

In Table 3.10, the running time of each algorithm (except PI) is polynomial with respect to the size of the policies or value functions they construct. Hence, the best algorithms tend to be those that quickly find small policies of high quality. BPI tends to find very small policies compared to the other algorithms and therefore is competitive in practice. By its ability to avoid local optima and to find relatively small policies of high quality, BPI is a robust scalable algorithm that can effectively overcome policy and value function complexity. Perseus is also quite efficient on some problems. We will discuss Perseus and BPI in more details in Chapter 5 when we combine them with techniques to deal with large state spaces.

Problems	Algorithms	Expected Reward	Solution Size	Time (seconds)
tiger-grid	BPI	2.22	120	1000
	Perseus	2.34	134	104
	PBVI	2.25	470	3448
	PBUA	2.30	660	12116
	BBSLS	n.a.	n.a.	n.a.
	PI	0	6448	86400
	appPI	0	150	86400
hallway	BPI	0.51	43	185
	Perseus	0.51	55	35
	PBVI	0.53	86	288
	PBUA	0.53	300	450
	BBSLS	n.a.	n.a.	n.a.
	PI	0.14	286	86400
	appPI	0.14	150	86400
hallway2	BPI	0.32	60	790
	Perseus	0.35	56	10
	PBVI	0.34	95	360
	PBUA	0.35	1840	27898
	BBSLS	n.a.	n.a.	n.a.
	PI	0.02	5	86400
	appPI	0.05	150	86400
tag-avoid	BPI	-6.65	17	250
	Perseus	-6.17	280	1670
	PBVI	-9.18	13340	180880
	PBUA	n.a.	n.a.	n.a.
	BBSLS	-8.20	30	100000
	PI	-15.71	3	86400
	appPI	-19.08	150	86400

Table 3.10: Results on benchmark problems.

Chapter 4

Compact State Space and Belief Space Representations

In Chapter 3, we saw that policy and value function complexity can often be overcome for POMDPs that possess small policies of high quality by using algorithms such as BPI that concentrate their search on compactly representable policies. Such techniques allow us to make an important step forward; however, a second source of intractability remains: the complexity of state spaces. Real-world POMDPs tend to have very large state spaces, which make impractical the classic solution algorithms of Chapter 2 as well as the more advanced algorithms of Chapter 3. Fortunately, real-world POMDPs tend to exhibit a significant amount of structure, so this chapter focuses on techniques that can exploit structure (when available) to mitigate the complexity of state and belief spaces. In particular, Section 4.1 reviews several types of structure previously identified in the literature and a wide range of algorithms proposed to exploit these. In Section 4.2, we describe a new *value-directed compression* technique that can reduce the dimensionality of POMDP components while preserving the information necessary to compute optimal policies. Finally, Section 4.3 illustrates the effectiveness of the compression technique on some test problems.

4.1 Literature Review

In a POMDP, the transition, observation and reward functions all use the state space to define their domain. For problems with very large state spaces, several techniques allow us to represent these functions in a compact way. In particular, we review in Section 4.1.1

some *factored* representations of those functions. Despite the possibility of specifying very large POMDPs in a compact form, it is not straightforward to design solution algorithms that also work directly with a compact representation without ultimately enumerating all or most states. In Sections 4.1.2-4.1.6, we survey some solution techniques that attempt to overcome the state space complexity by exploiting the structure of factored POMDPs.

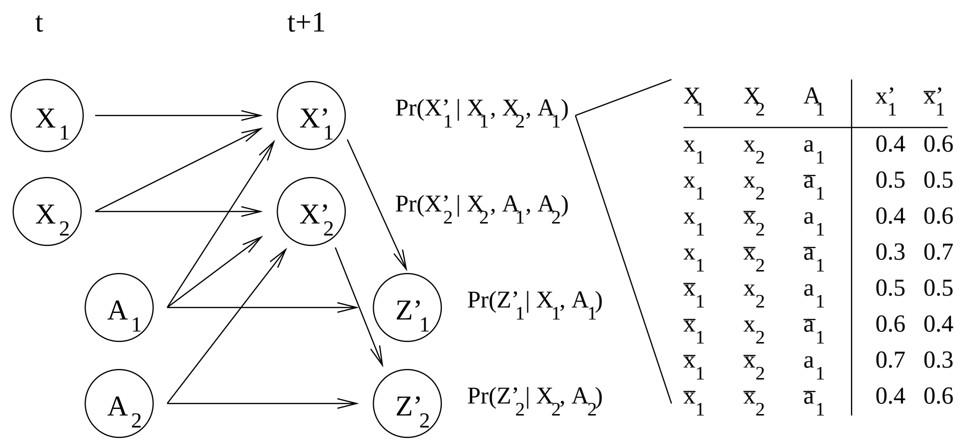
4.1.1 Factored POMDPs

In a POMDP, the relevant features of the world are summarized by a state $s \in \mathcal{S}$. Suppose each relevant feature is represented by some variable X_i with domain $\mathcal{D}_i$. Then, each state is a different joint instantiation of the variables $X_1, X_2, \ldots, X_n$ (assuming n variables). The size of the state space is exponential in n since $\mathcal{S} = \mathcal{D}_1 \times \mathcal{D}_2 \times \ldots \times \mathcal{D}_n$. Similarly, one can often decompose the action space $\mathcal{A}$ and the observation space $\mathcal{Z}$ such that each action a corresponds to a joint instantiation of the action variables and each observation z corresponds to a joint instantiation of the observation variables. In many domains, it is possible to define the transition, observation and reward functions in terms of the state variables, action variables and observation variables (instead of the states, actions and observations), allowing compact *factored* representations.

Conditional Independence

The transition and observation functions of a POMDP can typically be compactly represented as a dynamic Bayesian network (DBN) [30], which is a graphical representation for stochastic processes that exploits *conditional independence*. Conditional independence refers to the fact that some variables are probabilistically independent of each other when the values of other variables are held fixed [102].

The conditional probabilities $\Pr(s'|s,a)$ and $\Pr(z|s,a)$ can be expressed using an acyclic directed graph such as the one in Figure 4.1. In this graph, the nodes are state, action and observation variables and the edges represent probabilistic dependencies. The nodes are arranged in slices corresponding to two successive time-steps and each state variable occurs in each slice. Each node X_i' in the second slice has a conditional probability table (CPT) that specifies the conditional probability distribution $\Pr(X_i'|parents(X_i'))$ of X_i' with respect to its parent variables. Parents are always in the current or prior slice and the graph must be acyclic. Using Bayes theorem, the transition function, which corresponds to $\Pr(X_1', X_2'|X_1, X_2, A_1, A_2)$, can be factored into a product of smaller



X_1	X_2	A_1	x'_1	$\bar{x}'_1$
x_1	x_2	a_1	0.4	0.6
x_1	x_2	$\bar{a}_1$	0.5	0.5
x_1	$\bar{x}_2$	a_1	0.4	0.6
x_1	$\bar{x}_2$	$\bar{a}_1$	0.3	0.7
$\bar{x}_1$	x_2	a_1	0.5	0.5
$\bar{x}_1$	x_2	$\bar{a}_1$	0.6	0.4
$\bar{x}_1$	$\bar{x}_2$	a_1	0.7	0.3
$\bar{x}_1$	$\bar{x}_2$	$\bar{a}_1$	0.4	0.6

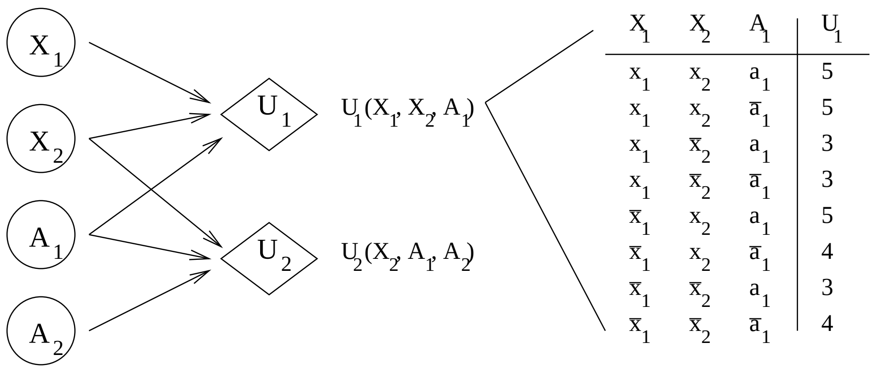
Figure 4.1: Dynamic Bayesian network.

conditional distributions. In the example of Figure 4.1, $\Pr(X'_1, X'_2|X_1, X_2, A_1, A_2) = \Pr(X'_1|X_1, X_2, A_1)\Pr(X'_2|X_2, A_1, A_2)$. Similarly, the observation function, which corresponds to $\Pr(Z'_1, Z'_2|X'_1, X'_2, A_1, A_2)$, can also be factored into a product of smaller conditional distributions (e.g., $\Pr(Z'_1, Z'_2|X'_1, X'_2, A_1, A_2) = \Pr(Z'_1|X'_1, A_1)\Pr(Z'_2|X'_2, A_2)$). Hence, the transition and observation functions are compactly encoded by the CPTs of the DBN. The size of each CPT is exponential in the number of parent variables; however, in practice, variables tend to have only a few parents.

Additive separability

The reward function of a POMDP can often be compactly represented by exploiting *additive separability* [139]. Additive separability refers to the fact that utility functions often decompose into sums of smaller utility functions [59, 3]. For instance, the reward function in Figure 4.2 is the sum of two small reward functions each depending on a small subset of action and state variables.

The conditional probability tables of transition and observation functions may also decompose into sums of smaller distributions. In particular, this happens when a distribution is a *mixture* (e.g., convex combination) of other distributions. For example, the CPT $\Pr(X'_1|X_1, X_2, A_1)$ in Figure 4.1 could be represented more compactly by a mixture of smaller distributions $\Pr(X'_1|X_1, A_1)$ and $\Pr(X'_1|X_2, A_1)$ (see Figure 4.3). Mixture distributions often arise in practice when several distributions are possible and we are not sure which one is right or as tractable approximations to complex distributions.



X_1	X_2	A_1	U_1
x_1	x_2	a_1	5
x_1	x_2	$\bar{a}_1$	5
x_1	$\bar{x}_2$	a_1	3
x_1	$\bar{x}_2$	$\bar{a}_1$	3
$\bar{x}_1$	x_2	a_1	5
$\bar{x}_1$	x_2	$\bar{a}_1$	4
$\bar{x}_1$	$\bar{x}_2$	a_1	3
$\bar{x}_1$	$\bar{x}_2$	$\bar{a}_1$	4

Figure 4.2: Additive rewards: $R(X_1, X_2, A_1, A_2) = U_1(X_1, X_2, A_1) + U_2(X_2, A_1, A_2)$.

$$\Pr(X'_1 \mid X_1, X_2, A_1) \quad = \quad 0.5 * \Pr(X'_1 \mid X_1, A_1) \quad + \quad 0.5 * \Pr(X'_1 \mid X_2, A_1)$$

X_1	X_2	A_1	x'_1	$\bar{x}'_1$
x_1	x_2	a_1	0.4	0.6
x_1	x_2	$\bar{a}_1$	0.5	0.5
x_1	$\bar{x}_2$	a_1	0.4	0.6
x_1	$\bar{x}_2$	$\bar{a}_1$	0.3	0.7
$\bar{x}_1$	x_2	a_1	0.5	0.5
$\bar{x}_1$	x_2	$\bar{a}_1$	0.6	0.4
$\bar{x}_1$	$\bar{x}_2$	a_1	0.7	0.3
$\bar{x}_1$	$\bar{x}_2$	$\bar{a}_1$	0.4	0.6

X_1	A_1	x'_1	$\bar{x}'_1$
x_1	a_1	0.6	0.4
x_1	$\bar{a}_1$	0.2	0.8
$\bar{x}_1$	a_1	0.8	0.2
$\bar{x}_1$	$\bar{a}_1$	0.4	0.6

X_2	A_1	x'_1	$\bar{x}'_1$
x_1	a_1	0.2	0.8
x_2	$\bar{a}_1$	0.8	0.2
$\bar{x}_2$	a_1	0.6	0.4
$\bar{x}_2$	$\bar{a}_1$	0.4	0.6

Figure 4.3: Mixture distribution.

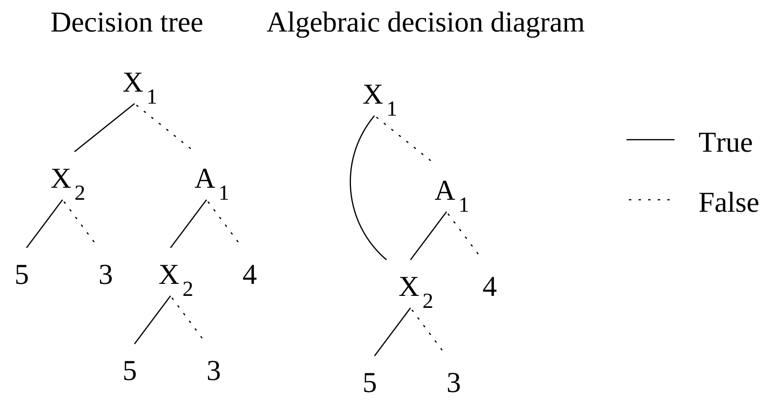


Figure 4.4: Decision tree and algebraic decision diagram to compactly represent the utility table of U_1 in Figure 4.2.

Context-Specific Independence

The probability tables in Figure 4.1 and the utility tables in Figure 4.2 can be further compressed by exploiting *context-specific independence* [12]. Intuitively, context-specific independence refers to the fact that some utility or random variables are independent of each other in some contexts (i.e., for some specific values of other random variables). In general, context-specific independence allows one to compactly represent probability and utility tables using Horn rules [107] decision trees (DTs) [12, 11, 14] or algebraic decision diagrams (ADDs) [49, 46]. For instance, Figure 4.4 illustrates the DT and ADD representations of the utility table for U_1 in Figure 4.2. A DT is a tree that stores in its leaves the values of a function defined over some variables. The value of the utility function U_1 for some truth assignment to X_1, X_2 and A_1 is found by following the corresponding branch. For example, when X_1 and X_2 are true, the value of U_1 is 5. Note that U_1 is independent of A_1 when X_1 and X_2 are true since the value of U_1 is 5 whether A_1 is true or false. Hence, by exploiting context-specific independence, the decision tree in Figure 4.4 can encode with only 5 leaves the 8 values of the utility table in Figure 4.2. An algebraic decision diagram (ADD) is essentially a decision tree (DT) with branches that are allowed to merge together. This allows further space reduction over DTs with minimal overhead to keep track of which subtrees are shared. In general DTs and ADDs can compactly represent a function defined over some set of variables by aggregating together values that are identical.

4.1.2 State Aggregation

Given a POMDP with transition, observation and reward functions compactly represented by exploiting conditional independence, additive separability and context-specific independence, we would like to devise a solution algorithm that can also work with compact representations. Classic solution algorithms all have in common a DP procedure to compute a set $\aleph^k$ of α-vectors from the previous set $\aleph^{k-1}$ of α-vectors. An α-vector is a linear function defined over the state space, so Boutilier and Poole [14] proposed a method to aggregate states with the same expected total return by compactly representing α-vectors by DTs. Assuming the reward, transition and observation functions are represented by DTs, then the DP procedure used by classic solution algorithms can be adapted to efficiently compute the expected total return of all states by partitioning the state space in aggregates with identical expected total return. The method was later refined to generate larger aggregates by using ADDs [49, 46] and by aggregating states with similar (instead of identical) expected total return [138, 36].

In practice, substantial savings have been demonstrated on several synthetic problems [46, 36]. Generally speaking, the smaller are the DTs/ADDs representing transition, observation and reward functions, the more likely it is that α-vectors can be computed and represented by small DTs/ADDs.

4.1.3 Linear Change of Basis

Given that reward functions are often additively separable into smaller reward functions defined over a small subset of variables, then α-vectors, which are computed from reward functions, may also be additively separable into small linear functions defined over a small subset of variables. Recent work in fully observable MDPs [62, 63, 40, 41, 124, 118] has explored the possibility of making a linear change of basis. Basis functions can be selected based on prior expert domain knowledge or by some automated techniques [101, 113] that greedily grow the subsets of state variables used to define the basis functions. Guestrin et al. [42] also proposed to adapt this idea to POMDPs: instead of working with α-vectors defined as linear functions over the state space, the α-vectors are approximated by some linear combination of small basis functions. However, the method remains to be implemented.

Note that state aggregation techniques such as DTs and ADDs exploit context specific independence whereas approximations with linear combinations of basis functions exploit additive separability. It is possible to simultaneously exploit both types of structure by using linear combinations of DTs/ADDs where each DT/ADD is a basis function [43].

4.1.4 Predictive Representations

In the POMDP framework, a state captures all the relevant features of the world for the purpose of planning. One can compress the state space by reducing it to some representation that retains only the relevant features of the world. Since the goal of a decision-theoretic agent is to accumulate as much future reward as possible, we can shrink the state space to some sufficient statistic that allows us to accurately predict future observations (assuming rewards are observable). Chrisman [27] first proposed a reinforcement learning technique called the *perceptual distinction approach* to grow a set of states that is statistically sufficient for predicting future observations. The method was later refined by McCallum who recognized that it is not necessary to distinguish future observations if they yield the same reward. He proposed several *utile distinction memory* approaches [83, 84, 85] that grow a set of states that are sufficient for predicting only expected future rewards. Lin and Mitchell [66] also report experiments using recurrent neural networks to learn sufficient statistics of the state space for predicting either future observations or future rewards. Along the same lines, Littman et al. [71, 120, 129, 54, 128] proposed to learn from past observations a *predictive state representation* that is sufficient to predict future observations.

These techniques were proposed for reinforcement learning scenarios where the number of states, the transition function, the observation function and the reward function are completely unknown. Since the agent only knows its own actions and the observations made, it must learn the remaining POMDP components (directly or indirectly) in addition to finding a good policy. The learning part of the problem turns out to be much harder than the planning part since a prohibitively large number of sample runs are often necessary. As a result, these techniques have only been demonstrated on small POMDPs.

4.1.5 Factored Belief States

When a policy is represented by a mapping from belief states to actions, the current belief state must be quickly updated at each time step using Bayes' theorem (Equation 2.1) during the execution of the policy. For factored POMDPs, since the transition and observation functions can be factored into compact representations, it may also be possible to factor belief states into products of marginals by exploiting conditional independence. The idea is to leverage the dynamic Bayesian network representation of factored POMDPs to efficiently update and compactly represent belief states. Recall that a belief state is a joint probability distribution of the state variables. If the state variables can be partitioned into probabilistically independent subsets, then the joint distribution can be compactly represented by the product of the marginal distributions of each subset.

If at every time-step belief states can be factored into sets of marginals, then belief state monitoring algorithms can focus only on factored belief states which have a reduced dimensionality equal to the sum of the size of each marginal. The size of a marginal is exponential in the size of the subset of variables it corresponds to; however, for small subsets, this effectively reduces dimensionality. Unfortunately, as observed by Boyen and Koller [17], even in the extreme case in which the initial belief state is completely factored (all state variables are mutually independent), correlations introduced by the transition and observation functions tend to "bleed through" the DBN, rendering most (if not all) state variables correlated after some time.

Nevertheless, many of those correlations are weak in practice and this suggests an approximation scheme where we force some subsets of variables to remain independent by breaking at each time step any correlation that could creep in. Boyen and Koller [17] proposed to *project* at each time step the exact joint distribution of the belief state onto a predetermined set of marginals that partition state variables into mutually independent subsets. They also showed that this approximation technique can significantly speed up belief state monitoring while ensuring that the KL divergence between exact and approximate belief states remains bounded for rapidly mixing DBNs.

During the execution of a POMDP policy, belief state monitoring is conducted only to facilitate action selection. Thus, although the KL divergence between exact and approximate belief states remains bounded, the policy may be altered since the action selected for every approximate belief state may be different from the one prescribed by the exact belief state. Poupart and Boutilier [110] showed that the quality of the policies resulting from approximate belief state monitoring can be significantly lower than the original policy even when the KL-divergence remains fairly small and that policy quality can be unaffected when KL-divergence is large. They also devised algorithms to select sets of marginals to project on that directly minimize the impact on decision quality rather than KL-divergence [110, 111].

Factored representations of belief states have also been used by McAllester and Singh [82] in a forward search algorithm that expands the search tree of future actions and observations to select the next action that maximizes future expected total return. In a POMDP reinforcement learning scenario, Sallans [122] also used factored representations of belief states to reduce the number of parameters for learning the POMDP dynamics as well as an optimal value function.

4.1.6 Sparsity

When the state space is very large or continuous, belief states can also be compactly approximated and efficiently monitored by sampling techniques (a.k.a. particle filtering or sequential Monte Carlo) [31, 58]. The idea is to approximate a belief state by a weighted sample of states, which is updated as follows: given a weighted sample representing the current belief state, some states reachable at the next time-step are sampled according to the transition function and re-weighted according to the observation function. This new weighted sample approximates the next belief state.

Assuming the sample size is much smaller than the state space size, then dimensionality is effectively reduced. In practice, sampling techniques tend to work well for problems with sparse transition functions and/or very informative observation functions. When the transition function is sparse, then only a few states can be reached from any state, allowing accurate approximation by a small sample of states. Similarly, when the observation function is very informative, then after making an observation only a few states are likely, once again allowing accurate approximation by a small sample of states. Poupart, Ortiz and Boutilier [114] also derived bounds on decision quality associated with sampling methods for approximate belief state monitoring during policy execution. Given that approximations by factored representations exploit conditional independence and that sampling techniques exploit distributional sparsity, Ng et al. [95] combined the two approaches in a very effective technique.

Besides sparse belief states, it is also common for real-world POMDPs to have very few reachable belief states, meaning that the reachable region of the belief space is also sparse. Roy and Gordon [119] proposed to compress the belief space to a lower dimensional manifold by exponential principal component analysis. The idea is to fit a low dimensional manifold to a sample of reachable belief states. This technique has proved very effective for robot navigation tasks, since the uncertainty in the location of a robot tends to be local.

4.2 Value-Directed Compression

Although real-world POMDPs tend to have very large state spaces, we highlighted in the previous section several types of structure and a variety of techniques that can exploit that structure to mitigate the complexity of state spaces. These techniques can be quite effective in reducing the dimensionality of the state space when the structure they exploit is present; however, most of them typically exploit a single type of structure, which limits their applicability. Furthermore, none of the techniques have been integrated with those of Chapter 3, which leaves them at the mercy of the policy space complexity. As a result, even though the techniques described in the previous section are designed to handle large-scale POMDPs, they can typically solve only toy problems.

This section describes a new *value-directed compression* (VDC) technique, which offers two important advantages. First, by combining ideas from state aggregation, predictive state representations and linear change of basis, VDC generates a reduced POMDP by simultaneously exploiting conditional independence, context-specific independence, additive separability and the fact that belief states often contain superfluous information that is irrelevant to the decision process. Second, as will be described in Chapter 5, the reduced POMDP produced by VDC can be fed to any of the algorithms of Chapter 3, allowing both sources of intractability to be contained.

4.2.1 Lossless Compressions

Recall that belief states constitute a sufficient statistic summarizing all information available to the decision maker (i.e., past actions and observations). However, as long as enough information is available to determine the *value* of each policy, one can still choose the best policy. Since belief states often contain information irrelevant to the estimation of future rewards, one can often compress belief states into some lower-dimensional representation. We provide a characterization of lossless compressions and then consider algorithms to compute linear compressions.

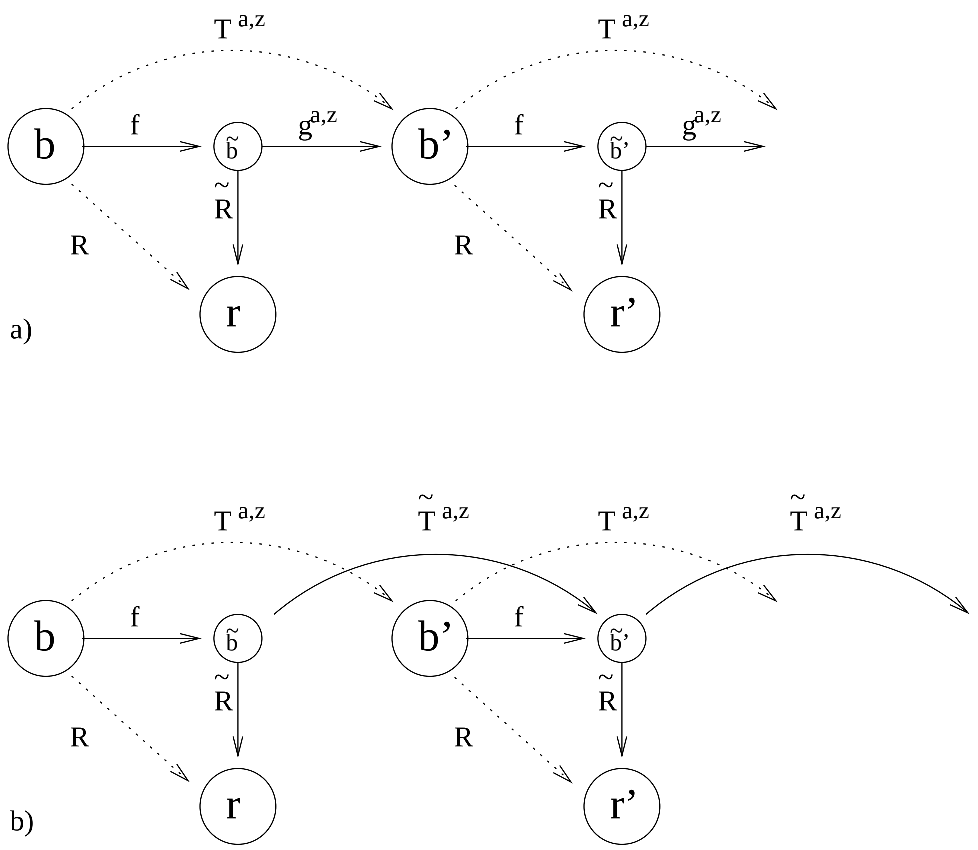


Figure 4.5: Functional flow of a POMDP (dotted arrows) and a compressed POMDP (solid arrows). a) next belief state is accurately predicted. b) next *compressed* belief state is accurately predicted.

Sufficient Conditions

Let f be a *compression function* that maps each belief state b into some lower dimensional compressed belief state $\tilde{b}$ (see Figure 4.5(a)). Here, $\tilde{b}$ can be viewed as a *bottleneck* (e.g., in the sense of the information bottleneck method [141]) that filters the information contained in b before it is used to estimate future rewards. A possible approach is to find a compression f such that $\tilde{b}$ corresponds to the smallest statistic sufficient for accurately predicting the current reward r as well as the next belief state b' (since we can accurately predict all following rewards from b'). Such a compression f exists if we can also find mappings $g^{a,z}$ and $\tilde{R}$ such that:

$$\begin{aligned} R &= \tilde{R} \circ f \;, && (4.1) \\ T^{a,z} &= g^{a,z} \circ f \;\; \forall a \in \mathcal{A}, z \in \mathcal{Z} \;. && (4.2) \end{aligned}$$

Here, $g \circ f$ denotes the composition of function g applied to the output of function f.

Since we are only interested in predicting future rewards, we don't really need to accurately estimate the next belief state b'; we could just predict the next compressed belief state $\tilde{b}'$ because it captures all information in b' relevant for estimating future rewards. Figure 4.5(b) illustrates the resulting functional flow, where $\tilde{T}^{a,z}$ represents the transition function that directly maps one compressed belief state to the next compressed belief state. The sufficient conditions of Equations 4.1 and 4.2 can then be replaced by the following weaker but still sufficient conditions:

$$\begin{aligned} R &= \tilde{R} \circ f \;, && (4.3) \\ f \circ T^{a,z} &= \tilde{T}^{a,z} \circ f \;\; \forall\, .a \in \mathcal{A}, z \in \mathcal{Z} && (4.4) \end{aligned}$$

Given an f, $\tilde{R}$ and $\tilde{T}^{a,z}$ satisfying Equations 4.3 and 4.4, we can evaluate a policy π by using the compressed POMDP dynamics as follows:

$$\tilde{V}^{\pi}(\tilde{b}) = \tilde{R}(\tilde{b}) + \gamma \sum_{z} \tilde{V}^{\pi}(\tilde{T}^{\pi(\tilde{b}),z}(\tilde{b})) \;. \qquad (4.5)$$

Since the conditions ensure accurate predictions of all future rewards, we can show that the value of each policy is the same with respect to the original POMDP and the compressed POMDP:

Theorem 4 *Let f, $\tilde{R}$ and $\tilde{T}^{a,z}$ satisfy Equations 4.3 and 4.4 then $V^{\pi}(b) = \tilde{V}^{\pi}(\tilde{b})$.*

Proof: We use a proof by induction. Base case: let $V_0^\pi(b) = R(b)$ and $\tilde{V}_0^\pi(\tilde{b}) = \tilde{R}(\tilde{b})$, then

$$\begin{aligned} V_0^\pi(b) &= R(b) \\ &= \tilde{R}(f(b)) \quad \text{(Equation 4.3)} \\ &= \tilde{R}(\tilde{b}) \quad \text{(by definition)} \\ &= \tilde{V}_0^\pi(\tilde{b}) \ . \end{aligned}$$

Induction: let $V_n^\pi(b) = \tilde{V}_n^\pi(\tilde{b})$ with n stages-to-go, then

$$\begin{aligned} V_{n+1}^\pi(b) &= R(b) + \gamma \textstyle\sum_z V_n^\pi(T^{\pi(b),z}(b)) && \text{(Bellman equation)} \\ &= R(b) + \gamma \textstyle\sum_z \tilde{V}_n^\pi(f(T^{\pi(b),z}(b))) && \text{(by induction)} \\ &= R(b) + \gamma \textstyle\sum_z \tilde{V}_n^\pi(\tilde{T}^{\pi(\tilde{b}),z}(f(b))) && \text{(Equation 4.4)} \\ &= \tilde{R}(f(b)) + \gamma \textstyle\sum_z \tilde{V}_n^\pi(\tilde{T}^{\pi(\tilde{b}),z}(f(b))) && \text{(Equation 4.3)} \\ &= \tilde{R}(\tilde{b}) + \gamma \textstyle\sum_z \tilde{V}_n^\pi(\tilde{T}^{\pi(\tilde{b}),z}(\tilde{b})) && \text{(by definition)} \\ &= \tilde{V}_{n+1}^\pi(\tilde{b}) \ . && \text{(Bellman equation)} \end{aligned}$$

◀

Theorem 4 essentially shows that all policies have identical values with respect to the original and compressed POMDPs. This means that we can run our favorite solution algorithm on the compressed POMDP to find an optimal policy for the original POMDP. When the components of the compressed POMDP are much smaller than the original POMDP, substantial time savings are possible.

Linear Compressions

We now turn to the problem of computing a compressed POMDP. In other words, we would like to find a compression mapping f and some compressed dynamics $\tilde{T}^{a,z}$ and $\tilde{R}$ that satisfy the sufficient conditions in Equations 4.3 and 4.4. Let's consider the class of *linear* compressions. As we will see, linear compressions naturally arise for problems that exhibit additive separability in the reward and/or transition functions. Furthermore, linear compressions can be easily computed.

Before explaining how to find linear lossless compressions, we briefly review several linear algebraic concepts that will be used later (see Saad [121] for more details). Let $\mathcal{X}$ be a vector subspace. We say that $\mathcal{X}$ is *invariant* with respect to matrix M if it is closed under multiplication by M (i.e., $Mx \in \mathcal{X}, \forall x \in \mathcal{X}$). A *Krylov subspace* $Kr(M, x)$ is the smallest subspace $\mathcal{X}$ that contains x and is invariant with respect to M. A basis

B for a Krylov subspace can easily be generated by repeatedly multiplying x by M (i.e., $B = \{x, Mx, M^2x, M^3x, \ldots, M^{n-1}x\}$) until a vector M^nx is found to be a linear combination of the previous vectors. If $Kr(M, x)$ is n-dimensional, one can show that $M^{n-1}x$ is the last linearly independent vector in this sequence and that all subsequent vectors are linear combinations of B.

When the compression mapping f is linear and therefore representable by some matrix F, the compressed transition and reward functions $\tilde{T}^{a,z}$ and $\tilde{R}$ must also be linear for discrete POMDPs (assuming Equations 4.3 and 4.4 are satisfied). Consequently, Equations 4.3 and 4.4 are linear and can be rewritten in matrix notation:

$$R = F\tilde{R}\,, \tag{4.6}$$

$$T^{a,z}F = F\tilde{T}^{a,z} \quad \forall a, z\ . \tag{4.7}$$

In a linear compression, F can be viewed as effecting a change of basis for the value function, with the columns of F defining a subspace in which the compressed value function lies. Furthermore, the rank of F indicates the dimensionality of the compressed state space. When Equations 4.6 and 4.7 are satisfied, the columns of F span a subspace that contains R and that is invariant with respect to each $T^{a,z}$. Intuitively, Equation 4.7 says that a sufficient statistic must be able to predict itself at the next time step, which is why the compression subspace must be invariant. Similarly, Equation 4.6 says that a sufficient statistic must also predict the current reward which is why the compression subspace must contain R. A formal proof is given in Theorem 5.

Theorem 5 *If $\tilde{T}^{a,z}$, $\tilde{R}$ and F satisfy Equations 4.6 and 4.7, then the range of F contains R and is invariant with respect to each $T^{a,z}$.*

Proof: Equation 4.6 ensures R is a linear combination of the columns of F, so it lies in the range of F. Equation 4.7 also requires that the columns of each $T^{a,z}F$ are linear combinations of the columns of F, so F is invariant with respect to each $T^{a,z}$. ◀

Thus, the best linear lossless compression corresponds to the smallest invariant subspace that contains R. This is by definition the Krylov subspace $Kr(\{T^{a,z} : a \in \mathcal{A}, z \in \mathcal{Z}\}, R)$. Using this fact, we can easily compute the best lossless linear compression by iteratively multiplying R by each $T^{a,z}$ until the Krylov basis is obtained. We then let

```
PROCEDURE $KrylovIteration(T^{a,z}, R)$
  compute $F$:
    set $F_1$ (first column of $F$) to $R$
    for each column $F_i$ of $F$ do:
      for each $\langle a, z \rangle$-pair do:
        if $T^{a,z} F_i$ is linearly independent of $F_1, \ldots, F_{i-1}$
          add $T^{a,z} F_i$ to the column set of $F$
        end if
      end for
    end for
  compute $\tilde{R}$:
    solve $R = F\tilde{R}$
  compute $\tilde{T}^{a,z}$:
    for each $\langle a, z \rangle$-pair do:
      solve $T^{a,z} F = F\tilde{T}^{a,z}$
    end for
  return $F$, $\tilde{R}$ and $\tilde{T}^{a,z}$
END PROCEDURE
```

Table 4.1: Krylov iteration for lossless compressions.

the Krylov basis form the columns of F, and compute $\tilde{R}$ and each $\tilde{T}^{a,z}$ by solving Equations 4.6 and 4.7. Table 4.1 summarizes the Krylov iteration algorithm to compute F, $\tilde{R}$ and $\tilde{T}^{a,z}$.

Although the Krylov iteration algorithm always generates the *best* linear lossless compression, an important question is how large will F be? In the worst case, when no compression is possible, F has $|\mathcal{S}|$ columns spanning the entire state space. However, the hope is to find a much smaller compression, which is likely to happen when the reward function and/or the transition dynamics are decomposable into sums of small functions. Once a good compression is found, we can solve the POMDP in the compressed state space by using $\tilde{R}$ and $\tilde{T}^{a,z}$.

Note that the Krylov iteration algorithm runs in time linear with respect to the number of states. For factored POMDPs that have exponentially many states, this is bad news. This means that compressing a POMDP by Krylov iteration is as imprac-

$$F = \begin{array}{c} s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \end{array} \begin{pmatrix} 5 & 3 & 2 \\ 1 & 2 & 1 \\ 1 & 2 & 1 \\ 8 & 0 & 0 \\ 4 & 0 & 4 \\ 0 & 0 & 8 \end{pmatrix}$$

Table 4.2: Interpretation of F.

tical as solving the POMDP directly. On the other hand, when a factored POMDP is specified compactly, say, using a DBN, we may be able to exploit the DBN structure and context-specific independence. In particular, if the transition, observation and reward functions are represented using decision trees (DTs) or algebraic decision diagrams (ADDs), then the matrix operations required by Krylov iteration can be implemented more efficiently [11, 49, 60]. The resulting basis vectors are also DTs or ADDs whose size depend on the amount of conditional and context-specific independence exhibited by the POMDP dynamics.

Interpretation

The mapping F corresponding to a lossless linear compression has an interesting interpretation. Consider the matrix F in Figure 4.2 representing a hypothetical compression mapping. Since F is a 6 by 3 matrix, it maps belief states from a 6-dimensional simplex to a 3-dimensional linear subspace. In particular, each row of F corresponds to a state and indicates the 3-dimensional point to which that state is mapped. For instance, state s_1 is mapped to $(5, 3, 2)$. When 2 rows are identical it means that the corresponding states have been aggregated since they are mapped to the same 3-dimensional point (e.g., s_2 and s_3 are aggregated in Figure 4.2). This means that linear compressions subsume state aggregation.

Beyond "pure" state aggregation, it is also possible that some states be aggregated with belief states. Suppose that s_4 corresponds to *yes*, s_5 to *maybe* and s_6 to *no*. Intuitively, the *maybe*-state is in between the *yes*-state and the *no*-state. From an information theoretic point of view, the *maybe*-state may be equivalent to 50% no, 50% yes. When that happens, the compression will simply aggregate the *maybe*-state with the uniform belief state over *yes* and *no*. Aggregations of states with belief states arise in F when a row is the convex combination of some other rows (e.g., s_5 is a convex combination of s_4 and s_6). Since belief states correspond to all possible convex combinations of the states, the convex hull of all rows forms the compressed belief space.

In general, linear compressions are more powerful than simple state and belief state aggregation. When a row is a linear combination of other rows, its corresponding state doesn't get aggregated. However, it gets projected in the same linear subspace, which reduces the dimensionality of the belief space.

As mentioned earlier, we can also interpret the columns of F as a basis spanning a linear subspace that contains the value function of every conditional plan. When F is computed by Krylov iteration, each entry corresponds to the reward earned after starting in some state and executing some sequence of actions and observations. For instance, Krylov iteration sets the first column of F to R so the entries of the first column are the immediate rewards of each state. Suppose the second column corresponds to $T^{a,z}R$ then the entries of the second column correspond to the rewards earned after executing a and observing z from each state. Similarly, suppose column n corresponds to $T^{a_1,z_1}T^{a_2,z_2}T^{a_3,z_3}R$ then the entries of that column correspond to the rewards received after executing and observing the sequence $a_1, z_1, a_2, z_2, a_3, z_3$ from each state. Note however that the entries of F may not be interpreted as future expected rewards when F is computed by other means than Krylov iteration.

4.2.2 Lossy Compressions

Suppose that due to computational constraints we can only solve POMDPs with belief spaces of k dimensions or fewer. Assuming there doesn't exist any effective lossless compression, we would like to find the best k-dimensional lossy compression. A compression is lossy when Equation 4.6 or 4.7 is not satisfied. As a result, the value estimates of each policy with respect to the compressed POMDPs may differ from those with respect to the original POMDP. However, if the error estimates are small enough, the best policy with respect to the compressed POMDP may still be near-optimal with respect to the original POMDP. We propose two approaches to find linear lossy compressions that almost satisfy Equations 4.6 and 4.7. The first approach is an optimization program that minimizes the residual error of Equations 4.6 and 4.7. The second approach consists of Krylov iteration with early stopping.

$$\begin{array}{ll} \min & c\epsilon_R + d\epsilon_T \\ \text{s.t.} & \|R - F\tilde{R}\|_\infty \leq \epsilon_R \\ & \|T^{a,z}F - F\tilde{T}^{a,z}\|_\infty \leq \epsilon_T \quad \forall a \in \mathcal{A}, z \in \mathcal{Z} \\ & \|F\|_\infty = 1 \end{array}$$

Table 4.3: Optimization program for linear lossy compressions.

Optimization program

We can formulate the problem of finding the best k-dimensional lossy compression as an optimization problem. The idea is to treat F, $\tilde{R}$ and $\tilde{T}^{a,z}$ as unknowns that we optimize to satisfy Equations 4.6 and 4.7 as much as possible. Table 4.3 outlines a simple optimization program to find lossy compressions that minimize a weighted sum of the max-norm[1] residual errors, ϵ_T and ϵ_R, of each sufficiency condition. Here, c and d are weights that allow us to vary the degree to which each sufficiency condition should be satisfied. The unknowns of the program are all the entries of $\tilde{R}$, $\tilde{T}^{a,z}$ and F as well as ϵ_T and ϵ_R. Here, $\|M\|_\infty$ indicates the maxnorm of matrix M, which is the largest absolute row sum (i.e., $\max_i \sum_j |M_{ij}|$). The constraint $\|F\|_\infty = 1$ is necessary to preserve scale, otherwise ϵ_T could be driven down to 0 simply by setting all the entries of F to 0. Since $\tilde{T}^{a,z}$ and $\tilde{R}$ multiply F, some constraints are nonlinear. However, it is possible to solve this optimization program by a series of LPs (linear programs). We alternate between solving the LP that adjusts $\tilde{R}$ and $\tilde{T}^{a,z}$ while keeping F fixed, and solving the LP that adjusts F while keeping $\tilde{R}$ and $\tilde{T}^{a,z}$ fixed. Convergence is guaranteed since the objective function decreases at each iteration; however, the resulting fixed point may not be a global optimum nor a local optimum.

The quality of the compression resulting from this optimization program depends on the weights c and d. Ideally, we would like to set c and d in a way that $c\epsilon_R + d\epsilon_T$ represents the loss in decision quality associated with compressing the state space. If we can bound the error ϵ_V of evaluating any policy using the compressed POMDP, then the difference in expected total return between the policy that is best with respect to the compressed POMDP and the truly optimal policy is at most $2\epsilon_V$. Let ϵ_V be $\max_\pi \|V^\pi - \tilde{V}^\pi \circ f\|_\infty$. Theorem 6 gives an upper bound on ϵ_V as a linear combination of the max-norm residual errors in Equations 4.6 and 4.7.

[1]Any norm could be used; however, the max-norm will be convenient to compute an upper bound on the loss in value in Theorem 6.

Theorem 6 *Let $\epsilon_V = \max_\pi \|V^\pi - \tilde{V}^\pi \circ f\|_\infty$, $\epsilon_R = \|R - \tilde{R} \circ f\|_\infty$, $\epsilon_T = \max_{a,z} \|T^{a,z} - \tilde{T}^{a,z} \circ f\|_\infty$ and $\tilde{V}^* = max_\pi \tilde{V}^\pi$. Then $\epsilon_V \leq \frac{1}{1-\gamma}\epsilon_R + \frac{\gamma|\mathcal{Z}|\ \|\tilde{V}^*\|_\infty}{1-\gamma}\epsilon_T$.*

Proof: The proof essentially consists of a sequence of substitutions of the type $\|AB\|_\infty \leq \|A\|_\infty \ \|B\|_\infty$ and $\|A+B\|_\infty \leq \|A\|_\infty + \|B\|_\infty$. In the following derivation, e represents a column vector of 1's.

$$\begin{aligned}
\epsilon_V &= \max_\pi \|V^\pi - \tilde{V}^\pi \circ f\|_\infty \\
&= \max_\alpha \|\alpha - F\tilde{\alpha}\|_\infty \\
&= \max_{\alpha_z,a,z} \|R + \gamma \sum_z T^{a,z}\alpha_z - F\tilde{R} - \gamma \sum_z F\tilde{T}^{a,z}\tilde{\alpha}_z\|_\infty \\
&\leq \|R - F\tilde{R}\|_\infty + \gamma \max_{\alpha_z,a,z} \|\sum_z (T^{a,z}\alpha_z - F\tilde{T}^{a,z}\tilde{\alpha}_z)\|_\infty \\
&\leq \epsilon_R + \gamma \max_{\alpha_z,a,z} \|\sum_z T^{a,z}(\alpha_z - F\tilde{\alpha}_z)\|_\infty \\
&\quad + \gamma \max_{\alpha_z,a} \|\sum_z (T^{a,z}F - F\tilde{T}^{a,z})\tilde{\alpha}_z\|_\infty \\
&\leq \epsilon_R + \gamma \max_{a,z} \|\sum_z T^{a,z}e\epsilon_V\|_\infty + \gamma \max_{\tilde{\alpha}_z,a} \sum_z \|(T^{a,z}F - F\tilde{T}^{a,z})\tilde{\alpha}_z\|_\infty \\
&\leq \epsilon_R + \gamma\epsilon_V + \gamma \sum_z \epsilon_T \|\tilde{V}^*\|_\infty \\
&\leq \frac{1}{1-\gamma}\epsilon_R + \frac{\gamma|\mathcal{Z}|\ \|\tilde{V}^*\|_\infty}{1-\gamma}\epsilon_T\ .
\end{aligned}$$

The above derivation assumes a discrete POMDP, which is known to have a piecewise-linear and convex value function. Hence a value function V can be substituted by a set of α-vectors each representing a linear piece of V. ◀

Even though the above error bound tends to grossly overestimate the actual loss in decision quality, we can still use it as a guide for setting c and d. Here, $\gamma|\mathcal{Z}|\ \|\tilde{V}^*\|_\infty/(1-\gamma)$ is typically much greater than $1/(1-\gamma)$ because of the factor $\|\tilde{V}^*\|_\infty$. This means that ϵ_T has a much higher impact on the loss in decision quality than ϵ_R. Intuitively, this makes sense because the error ϵ_T in predicting the next compressed belief state may compound over time, so we should set d significantly higher than c. Table 4.4 summarizes the alternating optimization procedure to compute good lossy compressions for a fixed dimensionality. Note that we can't set d directly to $\gamma|\mathcal{Z}|\ \|\tilde{V}^*\|_\infty/(1-\gamma)$ because $\|\tilde{V}^*\|_\infty$ is unknown; however, we can substitute $\|\tilde{V}^*\|_\infty$ by some estimate or simply set d to some large number.

```
PROCEDURE $AlternatingOptimization(T^{a,z}, R)$
  set $d$ to some estimate of $\gamma|\mathcal{Z}|\|\tilde{V}^*\|_\infty/(1-\gamma)$ (or some relatively large number)
  set $c$ to $1/(1-\gamma)$ (or some relatively small number)
  initialize $F$ randomly
  repeat
    fix $F$ and solve Program 4.3 for $\tilde{R}$ and $\tilde{T}^{a,z}$
    fix $\tilde{R}$ and $\tilde{T}^{a,z}$, and solve Program 4.3 for $F$
  until convergence of $\epsilon_T$ and $\epsilon_R$
  return $F$, $\tilde{R}$ and $\tilde{T}^{a,z}$
END PROCEDURE
```

Table 4.4: Alternating optimization for lossy compressions.

Truncated Krylov Iteration

Compared to Krylov iteration, the alternating optimization procedure is much more demanding computationally. Krylov iteration involves a sequence of matrix multiplications (to compute F) and solves several systems of linear equations (to compute $\tilde{R}$ and $\tilde{T}^{a,z}$). In contrast, the alternating optimization procedure solves a series of linear programs (LPs). Furthermore, empirical evidence suggests that it converges slowly, often requiring a large number of LPs to be solved. It would be nice to design an algorithm for lossy compressions that retains the simplicity of Krylov iteration.

To that effect, we propose a *truncated* Krylov iteration algorithm. The idea is to stop Krylov iteration early, retaining only the basis vectors that are "far" from being linear combinations of prior vectors. For instance, if v is a linear combination of $v_1, v_2, \ldots, v_n$, then there are coefficients $c_1, c_2, \ldots, c_n$ s.t. the error $||v - \sum_i c_i v_i||_2$ is zero. Given a threshold ϵ or some upper bound k on the desired number of columns in F, we run Krylov iteration, retaining only the vectors with an error greater than ϵ, or the k vectors with largest error. When F is computed by truncated Krylov iteration, we cannot compute $\tilde{R}$ and $\tilde{T}^{a,z}$ by solving Equations 4.6 and 4.7—due to the lossy nature of the compression, the system is over-constrained. But, we can find suitable $\tilde{R}$ and $\tilde{T}^{a,z}$ by computing a least square approximation, solving:

```
PROCEDURE $TruncatedKrylovIteration(T^{a,z}, R, k)$
  compute $F$:
    initialize $F$ to $\emptyset$
    initialize $basisSet$ to $R$
    while $basisSet$ is not empty and $F$ has fewer than $k$ columns do:
      remove vector $x$ from $basisSet$ with largest error $||x - \sum_i c_i F_i||_2$
      add $x$ to the columns of $F$
      for each $\langle a, z\rangle$-pair do:
        add $T^{a,z}x$ to $basisSet$
      end for
    end while
  compute $\tilde{R}$:
    solve $F^\top R = F^\top F \tilde{R}$
  compute $\tilde{T}^{a,z}$:
    for each $\langle a, z\rangle$-pair do:
      solve $F^\top T^{a,z} F = F^\top F \tilde{T}^{a,z}$
    end for
  return $F$, $\tilde{R}$ and $\tilde{T}^{a,z}$
END PROCEDURE
```

Table 4.5: Truncated Krylov iteration for lossy compressions.

$$F^\top R = F^\top F \tilde{R}\,, \tag{4.8}$$

$$F^\top T^{a,z} F = F^\top F \tilde{T}^{a,z} \;\; \forall a \in \mathcal{A}, z \in \mathcal{Z}\,. \tag{4.9}$$

Alternatively, we can also find suitable $\tilde{R}$ and $\tilde{T}^{a,z}$ by solving the LP in Table 4.3 since F is fixed. This is more demanding computationally than solving Equations 4.8 and 4.9, but it tells us the maxnorm residuals ϵ_R and ϵ_T that can be used to compute the upper bound ϵ_V on the loss of decision quality. In practice, the actual loss of decision quality appears to be roughly the same whether we compute $\tilde{R}$ and $\tilde{T}^{a,z}$ by minimizing the Euclidean norm (Equations 4.8 and 4.9) or the maxnorm (Table 4.3).

Table 4.5 summarizes truncated Krylov iteration for a fixed desired dimensionality k. Note again that for factored POMDPs with exponentially many states, the algorithm can be carried out efficiently by using DTs or ADDs to represent vectors and matrices. If at any point during the computation the size of some DTs/ADDs become prohibitively large, we can always shrink them by merging leaves with similar value (see prior work [138] for more details). This additional approximation introduces further inaccuracies in the compression, which means that the compression will simply be more lossy.

Intuitively, truncated Krylov iteration stores in F the future rewards that are "harder" to predict from the previous ones. Furthermore, since future rewards are considered by gradually increasing the horizon, early stopping yields lossy compressions that tend to evaluate policies more accurately over a short horizon than a long horizon. Hence, POMDPs with good myopic policies can often be solved effectively despite the lossy nature of the compressions produced by truncated Krylov iteration.

4.2.3 Summary

The value-directed compression (VDC) technique proposed addresses the intractability caused by large state spaces by compressing POMDP problems to a lower dimensional representation. The compression proposed takes advantage of the fact that optimal decision making is possible as long as each policy can be accurately evaluated. Since belief states often contain superfluous information that is irrelevant to the reward estimation of policies, it is often possible to compress belief spaces (as well as the reward functions and the transition dynamics). A set of sufficient conditions for lossless compression was presented and algorithms for linear (lossless and lossy) compressions were proposed.

An important aspect of linear VDC is its ability to simultaneously exploit several types of structure. In particular, VDC naturally exploits additive separability in reward functions and conditional probability tables. When reward functions and conditional probability tables can be decomposed into sums of smaller terms, then value functions also tend to decompose additively, allowing compact representations as linear combinations of the columns of F. Krylov iteration can also take advantage of conditional and context-specific independence. When reward functions and conditional probability tables are compactly represented DTs or ADDs then matrix computations can often be done efficiently and the columns of F can also be compactly represented as DTs and ADDs. Similarly, sparsity in the transition dynamics and the rewards can be exploited by using sparse matrices, sparse DTs or sparse ADDs, which omit zero entries. On the other hand, VDC doesn't exploit sparsity in the reachable belief region (i.e., when there are few belief states reachable). Another important aspect of linear VDC is that compressed POMDPs can be fed directly to most of the algorithms discussed in Chapter 3, allowing both sources of intractability to be mitigated simultaneously.

Note that linear VDC essentially performs a change of basis in which the value function of every possible conditional plan is compactly represented by a linear combination of the columns of F. In practice, it is rarely the case that a small basis set can span *all* the value functions of every possible conditional plan. Although one can always resort to a lossy compression, it may also be the case that a small basis set is sufficient to span *only* the value functions of the conditional plans of the optimal policy (and the intermediate policies as we compute the optimal policy). This is a promising alternative that remains to be explored.

We now highlight the similarities and differences between VDC and some related techniques.

State Aggregation

In general, linear VDC subsumes state aggregation since all possible aggregation schemes can be represented by compression mappings F that have identical rows for aggregated states. VDC can also be viewed as a generalization of Givan et al.'s [38] *model minimization* technique since VDC computes a *stochastic bi-simulation* of POMDPs. Two MDPs $M_1 = \langle \mathcal{S}_1, \mathcal{A}, T_1, R_1 \rangle$ and $M_2 = \langle \mathcal{S}_2, \mathcal{A}, T_2, R_2 \rangle$ are stochastically bi-similar when there exists an equivalence relation E between $\mathcal{S}_1$ and $\mathcal{S}_2$ such that for all $(s_1, s_2) \in E$:

$$\begin{aligned} &R_1(s_1) = R_2(s_2) \ , \\ &T_1(s_1, a, s_1') = T_2(s_2, a, s_2') \quad \forall (s_1', s_2') \in E \ . \end{aligned}$$

Since POMDPs can be viewed as belief state MDPs, then two POMDPs are stochastically bi-similar when there exists an equivalence relation E between their respective belief spaces $\mathcal{B}_1$ and $\mathcal{B}_2$ such that for all $(b_1, b_2) \in E$:

$$b_1 R_1 = b_2 R_2 \ , \tag{4.10}$$

$$\text{if } b_1 T^{a,z} = b_1' \text{ and } b_2 T^{a,z} = b_2' \text{ then } (b_1', b_2') \in E \quad \forall a \in \mathcal{A}, z \in \mathcal{Z} \ . \tag{4.11}$$

If we view the compression mapping f as an equivalence relation that maps each belief state b to its corresponding compressed belief state $\tilde{b}$ then it is easy to see that

Equations 4.3 and 4.10 are equivalent and that Equations 4.4 and 4.11 are equivalent. Hence, value directed compressions of POMDPs are stochastic bi-simulations.

Predictive state representations

VDC is also closely related to the predictive state representation (PSR) framework proposed by Littman et al. [71, 128]. The PSR approach compactly represents the information available to the decision maker by a sufficient statistic for estimating the probability of future observations. In a linear PSR, belief states are compressed using a matrix U of "core tests" corresponding to the probability of observing some sequence of actions and observations from each state. The matrices U (for PSR) and F (for VDC) differ only in the sense that U's entries are probabilities and F's entries are rewards. Littman et al. also proposed a Krylov iteration algorithm to compute U by multiplying e (vector of 1's) by each $T^{a,z}$ over and over, retaining only the linearly independent vectors to form the columns of U. Hence, linear PSR and linear VDC are equivalent except for the fact that one predicts future observations and the other predicts future rewards. When the rewards are observable, then the representations produced by both are equivalent.

Exponential Principal Component Analysis

As mentioned earlier, Roy and Gordon [119] developed a non-linear compression technique for belief states called *exponential principal component analysis*. The idea is to exploit the sparsity of the reachable belief region to compress it to some low dimensional manifold. Exponential PCA and VDC differ in the type of structure they exploit to achieve their compression. Exponential PCA exploits sparsity, where as VDC doesn't. In contrast, VDC prunes from belief states the superfluous information that is irrelevant to the decision process, whereas exponential PCA doesn't. In the future it would be interesting to combine the two techniques to simultaneously exploit both types of structure.

Note also that Exponential PCA performs a *non-linear* compression whereas we have only proposed algorithms for *linear* VDC. Although non-linear compressions are more powerful than linear ones, most of the algorithms described in Chapter 3 cannot operate on compressed POMDPs resulting from non-linear compressions.

4.3 Experiments

This section presents some experiments to test VDC's ability to reduce the dimensionality of state spaces. We only report experiments for Krylov iteration since it is several orders of magnitude faster and much simpler to implement than the alternating optimization technique. The experiments were conducted on four problems taken (or inspired) from the literature: a coffee delivery problem [14], a spoken dialog system [144] and two synthetic network management problems similar to those of Guestrin et al. [40]. All four problems are factored POMDPs.

The coffee problem (proposed by Boutilier and Poole [14]) is a toy POMDP featuring a robot that must purchase coffee at a coffee shop and deliver it to a user. The spoken-dialog system (developed by Williams [144]) consists of an automated travel agent that must elicit the departure and destination cities of a customer. The agent's task is to ask a series of questions to determine the traveling needs of the customer, which is complicated by the noisy nature of speech recognition. Since the details of those two problems are not essential to the experiments, a full description is provided only in Appendix A.

The network management problems are partially observable extensions of the fully observable ones described by Guestrin et al. [40]. We briefly describe them for the benefit of the analysis of the experiments. A system administrator must maintain a network of machines. Each machine has a 0.1 probability of failing at any stage, which increases to 0.333 when a neighboring machine is down. The administrator receives a reward of 1 per working machine and 2 per working server.[2] At each stage, she can either reboot a machine, ping a machine or do nothing. She only observes the status of a machine (with 0.95 accuracy) if she reboots or pings it. Costs are 2.5 (rebooting), 0.1 (pinging), and 0 (doing nothing). An n-machine network induces a POMDP with 2^n states, $2n+1$ actions and 2 observations. We report on experiments with networks of 7 machines organized in two configurations: *cycle* (Figure 4.6a) and *3legs* (Figure 4.6b). Edges indicate which machines are neighbors.

Note that all four problems are relatively small (see Table 4.6). In fact, they can be solved without any compression. This allows us to evaluate the loss in decision quality resulting from lossy compressions by comparing the value of the optimal policies for the original and compressed POMDPs. In Chapter 5, we will present further experiments

[2]In this problem, servers behave the same way as ordinary machines. However, the administrator receives a higher reward for working servers than for working ordinary machines.

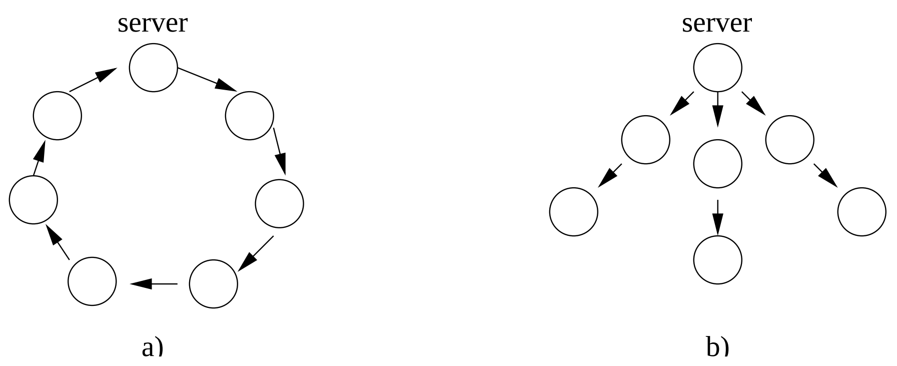


Figure 4.6: Network configurations: a) *cycle*, b) *3legs*.

Problems	$\|\mathcal{S}\|$	$\|\mathcal{O}\|$	$\|\mathcal{A}\|$
coffee	32	3	2
spoken-dialog	433	37	16
cycle7	128	2	15
3legs7	128	2	15

Table 4.6: Problem sizes.

with larger POMDPs of up to 33 million states.

In a first experiment, Krylov iteration was run on each problem to see if lossless compressions are possible. Table 4.7 compares the dimensionality of the best lossless compression to the original state space. The coffee and spoken dialog problems can be significantly compressed, but not the network problems.

In a second experiment, truncated Krylov iteration was run on each problem to further reduce the dimensionality. Figures 4.7, 4.8, 4.9 and 4.10 report the value of the policies found by running BPI on compressed versions of varying dimensionality (the number of basis functions indicate the dimensionality of the compressed space). Since the controllers found by BPI depend on the random initialization, we report, for each dimensionality tested, an interval corresponding to one standard deviation of 20 BPI runs. In each

	coffee	spoken-dialog	cycle7	3legs7
$\|\mathcal{S}\|$	32	433	128	128
lossless	7	31	128	128

Table 4.7: Dimensionality of the best lossless compression versus original state space.

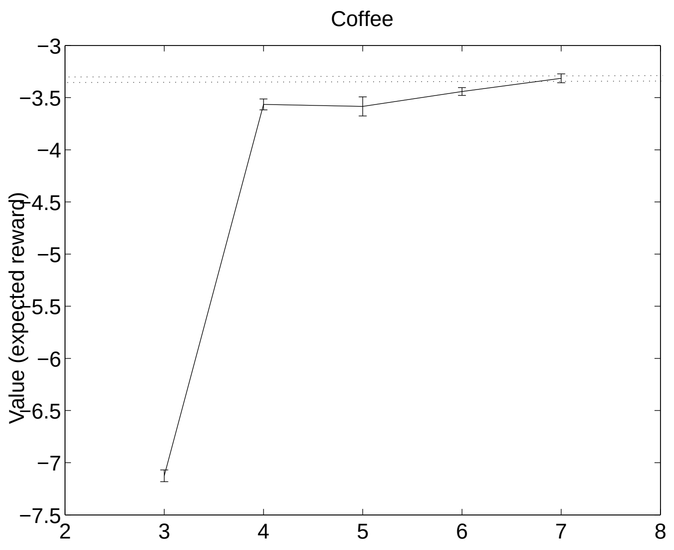


Figure 4.7: Coffee problem: value of the policies resulting from lossy compressions (solid line) versus the range of best values achieved without compression (dotted lines).

graph, the horizontal dotted lines correspond to the value (with one standard deviation) of the best controller found by BPI without any compression. Hence, the difference between the intervals of the curves represent the loss in decision quality induced by a lossy compression. In all cases, near-optimal solutions can still be found with a relatively small number of dimensions. For the coffee problem and the spoken-dialog system, only 4 basis functions out of 32 and 7 basis functions out of 433 respectively can still yield a near optimal policy. Similarly, for the network problems, 50 basis functions out of 128 are sufficient to obtain a near optimal policy. This is still remarkable given that no lossless compressions were possible for the network problems. The curves are generally noisy since increasing the number of basis functions doesn't necessarily yield better policies and numerical instability may also creep in. We believe this explains the dip in value observed when the number of basis functions approaches 128 for the *3legs* problem. It is interesting to note that all four problems are factored POMDPs exhibiting conditional independence, context-specific independence and additive separability. This explains the success of VDC.

In contrast, the four maze problems (tiger-grid, hallway, hallway2 and tag-avoid) of Chapter 3 are not factored POMDPs and do not exhibit any conditional independence, context-specific independence and additive separability. We also tried Krylov iteration on

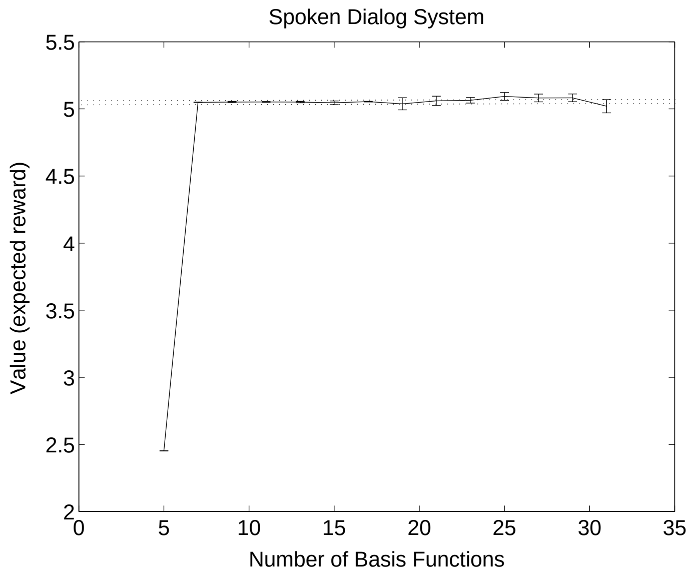


Figure 4.8: Spoken dialog system: value of the policies resulting from lossy compressions (solid line) versus the range of best values achieved without compression (dotted lines).

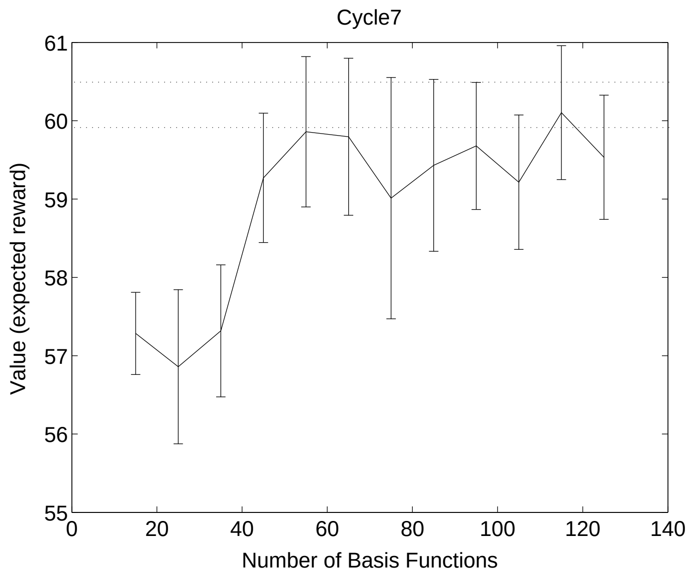


Figure 4.9: Cycle network of 7 machines: value of the policies resulting from lossy compressions (solid line) versus the range of best values achieved without compression (dotted lines).

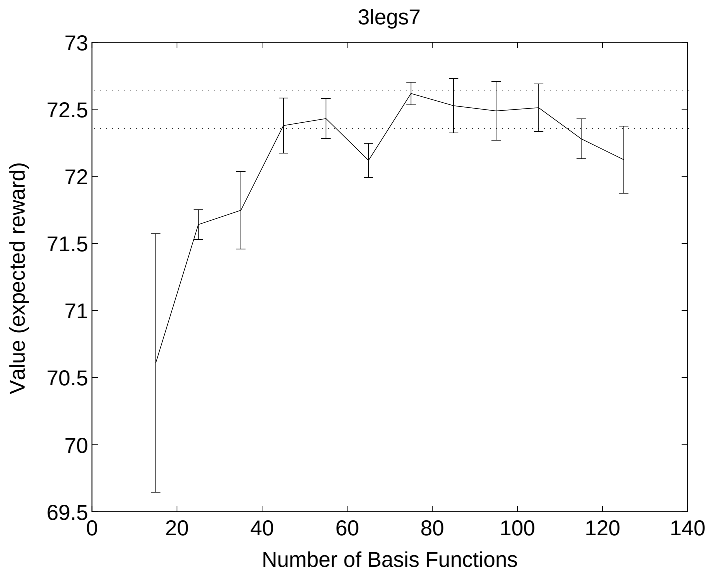


Figure 4.10: 3legs network of 7 machines: value of the policies resulting from lossy compressions (solid line) versus the range of best values achieved without compression (dotted lines).

those problems, but without any success. There were no lossless compressions and lossy compressions were too unstable numerically to allow BPI to work properly.[3] Note that the state space of these four maze problems correspond to a discretization of some continuous floor plan. Hence, instead of compressing the state space by VDC, one can simply discretize the floor plan more coarsely. Alternatively, since these are robot navigation problems with sparse reachable belief regions, we also expect exponential PCA to be quite effective.

Note also that there were no time savings when running VDC followed by BPI on the compressed problems compared to running BPI on the full problems. This is simply because the overhead due to the compression is too large for such small problems. We defer to Chapter 5 experiments on larger problems that BPI (or any other algorithm of Chapter 3) cannot handle.

[3]Numerical instability naturally arises when a vector is multiplied over and over by some matrices as in Krylov iteration. It can be significantly reduced by orthogonalizing the Krylov basis, but this wasn't good enough in our experiments.

Chapter 5

Algorithms for Large POMDPs

In Chapter 3, the complexity of policy and value function spaces was addressed by observing that there are often very good policies with value functions representable by a small number of α-vectors. Various algorithms such as point-based value iteration (PBVI) [106], Perseus [137], bounded policy iteration (BPI) [112], gradient ascent (GA) [89, 1] and stochastic local search (SLS) [20] exploit this fact to produce (often near-optimal) policies of low complexity, allowing larger POMDPs to be solved. Still, these algorithms scale to problems of only roughly 1000 states since each α-vector may have exponential dimensionality. On the other hand, it was observed in Chapter 4 that factored POMDPs often exhibit a significant amount of structure (e.g., conditional independence, context-specific independence, additive separability) and that reachable belief states are often sparse and tend to carry more information than necessary. Hence, one can often reduce vector dimensionality by using compact representations such as decision trees (DTs) [14], algebraic decision diagrams (ADDs) [46], linear combinations of small basis functions (LCBFs) [42], or by compressing the belief space into a small subspace by a value-directed compression (VDC) [112] or exponential principal component analysis (PCA) [119]. To date, these techniques have only been implemented with the classical solution algorithms. Although they can effectively overcome the complexity of large state spaces, none of them address the exponential complexity of policy and value function space. As a result, they cannot solve much larger/difficult POMDPs.

Scalable POMDP algorithms can only be realized when both sources of intractability are tackled simultaneously. This chapter describes how to combine some of the techniques of Chapters 3 and 4. More precisely, we discuss how to integrate BPI with VDC (Section 5.1) as well as Perseus with VDC and ADDs (Section 5.2). The resulting algorithms can effectively overcome both sources of intractabilities for POMDPs exhibiting suitable structure. In Section 5.3, experiments demonstrate their scalability on synthetic network management problems and an assistive technology task of up to 33 million states.

5.1 Compressed Bounded Policy Iteration

In principle, any POMDP algorithm can be used to solve the compressed POMDPs produced by VDC. If the compression is lossless and the POMDP algorithm exact, the computed policy will be optimal for the original POMDP. In practice, POMDP algorithms are usually approximate and lossless compressions are not always possible, so care must be taken to ensure numerical stability and a policy of high quality for the original POMDP. We now discuss some of the integration issues that arise when combining VDC with BPI.

5.1.1 Nonnegativity

Recall that BPI optimizes a controller by maximizing the α-vector of each node. When BPI operates on a compressed POMDP, we would like to make sure that value improvements in the compressed α-vectors translate in value increases with respect to the original POMDP. Otherwise BPI may think that it is improving a controller when it is really harming it. Since a node's compressed value $\tilde{\alpha}$ is related to its real value α by $\alpha = F\tilde{\alpha}$ (see Theorem 4), then maximizing $\tilde{\alpha}$ automatically maximizes α when F is nonnegative. Hence it is essential that the entries of F be nonnegative. Otherwise, the optimal policy of the compressed POMDP may not be optimal for the original POMDP.

Fortunately, when R is nonnegative then F is guaranteed to be nonnegative by the nature of Krylov iteration. Recall from Table 4.1 that each column of F is the product of some transition matrices $T^{a,z}$ by R. Hence F is nonnegative when R and each $T^{a,z}$ are nonnegative. By definition, the transition matrices $T^{a,z}$ are nonnegative, but the reward function R may not be. If some rewards are negative, we can add a sufficiently large constant to R to make it nonnegative without changing the decision problem. When computing lossy compressions with the optimization program in Table 4.3, we can also ensure that F is nonnegative by adding suitable constraints (i.e., $F_{i,j} \geq 0$).

Hence, we can generally ensure that improvements in the compressed α-vectors translate in value increases with respect to the original POMDP by making F nonnegative, which is the case with Krylov iteration when the reward function is also made nonnegative.

5.1.2 Normalization

Since BPI computes approximately optimal policies, we would also like to make sure that ϵ-optimal policies in the compressed space are indeed ϵ-optimal with respect to the original POMDP. If there are important differences in the optimality measures of the compressed and original spaces, then BPI may return a policy that is far from optimal for the original POMDP despite the fact that it is ϵ-optimal for the compressed POMDP.

For example, suppose F has two columns F_1 and F_2 with L_1-lengths 1 and 100, respectively. Since $\alpha = F\tilde{\alpha} = \tilde{\alpha}_1 F_1 + \tilde{\alpha}_2 F_2$, changes in $\tilde{\alpha}_2$ have a much greater impact on α than changes in $\tilde{\alpha}_1$. Such a difference in sensitivity may bias the search for a good policy to an undesirable region of the belief space, which may prevent approximate algorithms such as BPI from finding near-optimal policies.

This problem can be avoided by normalizing the columns of F to have the same L_1 norm. More precisely, each column F_i is normalized by dividing by its L_1-length (i.e., $F_i \leftarrow F_i/||F_i||_1$). This way, changes of the same magnitude in the entries of $\tilde{\alpha}$ will have a similar impact on α. This will ensure that compressed ϵ-optimal policies are also ϵ-optimal with respect to the original POMDP. Note that this normalization doesn't change the subspace spanned by the columns of F.

5.1.3 Iterative Policy Evaluation

Note also that it is "safer" to evaluate policies iteratively rather than solving the system in Equation 2.12. Recall that the value of a policy is the expected sum of future rewards, which can be computed by an infinite series of policy backups:

$$\begin{array}{ll}
V_0^{\langle a,\sigma\rangle}(s) = r^a(s) & \forall s \in \mathcal{S},\ \langle a,\sigma\rangle \in \Gamma^\pi \\
V_1^{\langle a,\sigma\rangle}(s) = r^a(s) + \gamma \sum_{s'\in\mathcal{S}} \Pr(s'|s,a) \sum_{z'\in\mathcal{Z}} \Pr(z'|a,s') V_0^{\sigma(z')}(s') & \forall s \in \mathcal{S},\ \langle a,\sigma\rangle \in \Gamma^\pi \\
V_2^{\langle a,\sigma\rangle}(s) = r^a(s) + \gamma \sum_{s'\in\mathcal{S}} \Pr(s'|s,a) \sum_{z'\in\mathcal{Z}} \Pr(z'|a,s') V_1^{\sigma(z')}(s') & \forall s \in \mathcal{S},\ \langle a,\sigma\rangle \in \Gamma^\pi \\
V_3^{\langle a,\sigma\rangle}(s) = r^a(s) + \gamma \sum_{s'\in\mathcal{S}} \Pr(s'|s,a) \sum_{z'\in\mathcal{Z}} \Pr(z'|a,s') V_2^{\sigma(z')}(s') & \forall s \in \mathcal{S},\ \langle a,\sigma\rangle \in \Gamma^\pi \\
\ldots &
\end{array}$$

Alternatively, the system in Equation 2.12 can also be used to evaluate policies since the transition matrices $T^{a,z}$ have eigenvalues of magnitude less than or equal to 1, which ensures that the solution is the same as that produced by infinitely many policy backups. In contrast, lossy compressed transition matrices $\tilde{T}^{a,z}$ may have eigenvalues of magnitude greater than one. When that is the case, the system in Equation 2.12 does not correspond to policy evaluation and therefore should not be used. Hence, it is safer to perform a series of policy backups for the policy evaluation step of BPI.

5.1.4 Constraint Generation

There are also important differences between the compressed and original belief spaces that may impact algorithms such as BPI. Recall that by definition a belief state is a probability distribution and therefore the sum of its components must be 1. Furthermore, the space of all belief states is a simplex corresponding to all nonnegative vectors whose components sum to 1. In contrast, compressed belief states are vectors of future reward estimates that do not generally sum to 1 and the compressed belief space corresponds to the convex hull of the rows of F. BPI normally exploits the simplex nature of the belief space during the policy improvement step. More precisely, the LPs in Tables 3.4, 3.6 and 3.7 have a set of constraints that ensure the value of a node doesn't decrease at any belief state. Since the belief space is a simplex, it suffices to have one constraint per state to ensure that there is no decrease in value for the entire belief space. A similar effect can be obtained with compressed belief spaces by having one constraint per vertex of the convex hull. Unfortunately, the number of vertices of the convex hull tends to be very large. In fact, in the worst case, there may be as many vertices as the number of rows of F which equals the number of states of the original model. Hence, for factored POMDPs with exponentially many states, the LPs in Tables 3.4, 3.6 and 3.7 tend to have exponentially many constraints.

Fortunately, we can get around this problem by using constraint generation techniques (similar to cutting plane and column generation techniques). These have been used successfully for similar LPs in the context of MDP [124] and minimax regret [13] optimization. The idea is to solve the LP with a small subset of constraints, find among the remaining constraints the one that is most violated by the current solution, add it to the set of constraints and repeat until there are no more violated constraints. The most violated constraint can often be found efficiently (i.e., without enumerating all constraints) by solving a *cost network* [124, 13]. Table 5.1 summarizes the constraint generation approach to incrementally build a small sufficient set of constraints. In practice, the number

```
PROCEDURE $LPconstraintGeneration(LP)$
    Let $C = \{c_1, c_2, \ldots, c_n\}$ be the entire set of constraints
    Let $G$ be the set of constraints generated
    Initialize $G = \{c_1\}$
    Repeat forever
        Solve LP subject to $G$
        If solution satisfies all constraints in $G$
            Then return solution
            Else add most violated constraint to $G$
        End if
    End repeat
END PROCEDURE
```

Table 5.1: LP constraint generation.

of constraints generated tends to be fairly small (proportional to the number of *active* constraints), so the LPs in Tables 3.4, 3.6 and 3.7 can be solved relatively efficiently even when belief space convex hulls have exponentially many vertices.

5.1.5 Summary

To summarize, a few adjustments are necessary to ensure that BPI can be run properly on compressed POMDPs produced by VDC. In particular, F should be nonnegative and its columns normalized to prevent numerical instability. Also, policy evaluation in BPI should be done by a series of DP backups since the eigenvalues of $\tilde{T}^{a,z}$ can be greater than 1. Finally, the LPs for policy improvement should be solved by constraint generation to deal with the potentially large number of vertices of belief space convex hulls. With those adjustments, POMDPs can first be compressed by VDC, generating a compression function F as well as compressed dynamics $\tilde{T}^{a,z}$ and $\tilde{R}$. The compressed model can then be fed to BPI to find a good controller.

The combination of BPI with VDC can solve relatively large POMDPs when a good compression is found by VDC and a small controller of high quality is found by BPI. Section 5.3 demonstrates the scalability of BPI with VDC on synthetic network management problems of up to 33 million states.

5.2 Fast Point-Based Value Iteration

This section describes how to combine point-based value iteration with VDC and ADDs. The version of point-based value iteration we consider is the Perseus algorithm [142, 137] because of its efficiency and simplicity. Let's briefly review the algorithm, which is summarized in Table 5.2. First, a set of reachable belief states are collected by doing a forward search from the initial belief state. This can be done in a number of ways. If we already have a "default" policy, we can execute it and record the belief states visited during n runs of k steps. Otherwise, we can execute a random policy by sampling the actions and observations at each step. Then, value iteration is performed by doing partial point-based backups at the reachable belief states. A point-based backup computes the best α-vector for each point in a set of witness belief states. A *partial* point-based backup looks at each witness point in turn and computes its best α-vector *only* if no other α-vector improves its value compared to the previous backup. The algorithm is fairly efficient since it produces a parsimonious set of α-vectors concentrated on the reachable belief region and it avoids costly optimization programs (every step consists of simple matrix multiplications).

5.2.1 Compressed Point-Based Value Iteration

As is, Perseus cannot tackle exponentially large POMDPs since the dimensionality of the α-vectors is linear in the number of states. This problem can be overcome by compressing large POMDPs with VDC and then running Perseus on the compressed version. The integration of Perseus with VDC is fairly straightforward and similar to that of BPI with VDC. As explained in the previous section, F should be nonnegative and its columns normalized to ensure numerical stability. Within Perseus itself, only one adjustment is necessary. When computing the set of reachable belief states, we normalize each belief state to make sure that its components sum to 1. More precisely, recall from Equation 2.1 that b_t is computed by multiplying b_{t-1} by the transition and observation functions for some action-observation pair and by rescaling the resulting vector so that its components sum to 1. When computing reachable belief states in the compressed space, we also compute $\tilde{b}_t$ by multiplying $\tilde{b}_{t-1}$ by the compressed dynamics $\tilde{T}^{a,z}$; however, it is unclear how to "normalize" the resulting vector. As explained in the previous section, the components of compressed belief states do not generally sum to 1, hence we should not normalize by dividing by the sum of the components.

```
PROCEDURE Perseus($T^{a,z}$, $R$)
  sample set of reachable belief states $B$:
  (e.g. execute $n$ runs of $k$ steps of a random or default policy)
    $B \leftarrow \emptyset$
    repeat $n$ times:
      $b \leftarrow$ initial belief state
      repeat $k$ times
        $B \leftarrow B \cup \{b\}$
        sample $a$ uniformly (or via a default policy)
        sample $z$ from $\Pr(z|b,a)$
        $b \leftarrow b_z^a$
      end repeat
    end repeat
  value iteration (partial point-based backup):
    $\aleph \leftarrow \{r_a | a \in \mathcal{A}\}$
    repeat
      $\aleph' \leftarrow \emptyset$
      for each $b \in B$
        if $\aleph' = \emptyset$ or $\max_{\alpha' \in \aleph'} b \cdot \alpha' < \max_{\alpha \in \aleph} b \cdot \alpha$
          (compute best new $\alpha'$ for $b$)
          for each $a \in \mathcal{A}$ do:
            $\alpha_a^* \leftarrow r_a + \gamma \sum_z T^{a,z} \operatorname{argmax}_{\alpha \in \aleph} b_z^a \cdot \alpha$
          end for
          $\alpha^* \leftarrow \operatorname{argmax}_{\alpha_a^*} b \cdot \alpha_a^*$
          $\aleph' \leftarrow \aleph' \cup \{\alpha^*\}$
        end if
      end for
      $\aleph \leftarrow \aleph'$
    until convergence
  return $\aleph$
END PROCEDURE
```

Table 5.2: Fast point-based value iteration.

Suppose that x is an unnormalized vector (i.e., $||x||_1 \neq 1$) and that once normalized it corresponds to belief state b. Let $\tilde{x}$ be the compressed version of x such that $\tilde{x} = xF$. We would like to find an operation that rescales $\tilde{x}$ into the compressed belief state $\tilde{b}$ corresponding to b (i.e., $\tilde{b} = bF$).

Here is a simple way of accomplishing this operation. Suppose that the i^{th} column of F is e (a vector of 1's), then the i^{th} entry of $\tilde{x}$ is $||x||_1$ (i.e., $\tilde{x_i} = xF_i = xe = ||x||_1$). We can then compute $\tilde{b}$ simply by dividing $\tilde{x}$ by its i^{th} entry $\tilde{x_i}$:

$$\begin{aligned}
\tilde{x}/\tilde{x}_i &= xF/\tilde{x}_i \\
&= xF/||x||_1 \\
&= bF \\
&= \tilde{b}\,.
\end{aligned}$$

It follows that we can effectively normalize compressed belief states when $F_i = e$. If F has a linear combination of columns equal to e, then we can still normalize $\tilde{x}$ by dividing it by the corresponding linear combination of its components. In Krylov iteration, we can make sure that F has a column (or linear combination of columns) equal to e by initializing the first two columns of F to e and R (instead of just R).

The adjustments described above are sufficient to integrate Perseus with VDC. So, we can easily run Perseus on a compressed POMDP by making sure F is nonnegative, has normalized columns and has a column (or linear combination of columns) equal to e. When normalizing compressed belief states, it suffices to divide by the appropriate component (or linear combination of components). The combination of Perseus with VDC makes it possible to tackle large POMDPs that exhibit suitable structure, as will be demonstrated in Section 5.3.

5.2.2 Symbolic Point-Based Value Iteration

Alternatively, the curse of dimensionality that plagues Perseus can be overcome by compactly representing α-vectors with algebraic decision diagrams (ADDs). The papers by Hoey et al. [49] and Hansen et al. [46] describe how to efficiently implement classic value iteration with ADDs. Since Perseus is very close to classic value iteration, the integration described in those papers can be applied directly to Perseus. The idea is simply to represent all vectors and matrices by ADDs and to perform symbolically the matrix operations for DP backups (or point-based backups). Intuitively, ADDs aggregate together the entries of vectors (or matrices) that are identical. This way, matrix operations can be done efficiently by performing the arithmetic operations applied to identical entries only once.

Although reward functions and conditional probability tables of factored POMDPs can often be represented compactly by ADDs, belief states and α-vectors computed by Perseus may not have a lot of identical values, resulting in potentially large ADDs. If this situation happens, we can always shrink the ADDs by aggregating similar (instead of identical) entries [138, 36]. We use this approximation to keep the size of ADDs bounded for α-vectors. We can do the same for belief states or alternatively, we can also use the factored approximation proposed by Boyen and Koller [17]. Recall from Chapter 4 that probability distributions are often well approximated by breaking some correlations, which allows a compact factored representation as a product of independent marginals. In our symbolic implementation of Perseus, we use this factored representation for belief states and ADDs for α-vectors. The resulting algorithm is essentially the same as the one described in Table 5.2, but uses ADDs and factored representations as the underlying data structures of vectors and matrices. Hence, this symbolic version of Perseus can effectively circumvent both sources of intractability, as will be demonstrated in the following section.

5.3 Experiments

This section describes some experiments to illustrate the scalability of compressed BPI, compressed Perseus and symbolic Perseus. Large synthetic network management problems (Section 5.3.1) and an assistive technology task (Section 5.3.2) of up to 33 million states are used as benchmarks. Given the size of those problems, the algorithms of Chapters 2 and 3 cannot be used since they run out of memory. The techniques of Chapter 4 can handle state spaces of this magnitude; however, they are also inadequate since the policy and value function representations they use grow exponentially. Only the algorithms described in this chapter can tackle these problems since they simultaneously mitigate both sources of intractability.

Machines	$\lvert\mathcal{S}\rvert$	$\lvert\mathcal{O}\rvert$	$\lvert\mathcal{A}\rvert$
16	$65,536$	2	33
19	$524,288$	2	39
22	$4,194,304$	2	45
25	$33,554,432$	2	51

Table 5.3: Problem sizes for networks of 16, 19, 22 and 25 machines.

5.3.1 Network Management Problems

We now report on some experiments with large synthetic network management problems. These are essentially the same as in Section 4.3, but with networks of 16, 19, 22 and 25 machines. Recall that a system administrator must keep as many machines as possible running in a network organized in a *cycle* or *3legs* configuration by doing nothing, pinging a machine or rebooting a machine at each time step (see Section 4.3 for more details). Table 5.3 indicates problem sizes as the number of machines increases in a network.

Figures 5.1 and 5.2 show the average expected reward earned by policies computed by BPI after the POMDP has been compressed by VDC. Expected rewards are averaged over 500 policy runs of 60 steps, starting with a belief state where all machines are working. Note that the ruggedness of the graphs is mainly due to the variance in the reward samples. As expected, decision quality improves as we increase the number of nodes used in BPI and the number of basis functions used in truncated Krylov iteration. Also interesting are some of the jumps in the reward surface of some graphs, suggesting phase transitions with respect to the dimensionality of the compression. For instance, the graph for a 3-leg network of 19 machines in Figure 5.1 exhibits an important jump in expected rewards around 180 basis functions.

The graph in Figure 5.3 shows the time taken by BPI on a *cycle* network of 25 machines (other problems exhibit similar behavior). Truncated policy iteration takes from 4902 seconds to 12408 seconds (depending on size and configuration) to compress POMDPs to 250 dimensions.[1]

We also ran Perseus on the compressed network problems. Tables 5.4 and 5.5 compare the results of compressed BPI and Perseus as well as two simple heuristics. The expected rewards are averaged over 500 policy runs of 60 steps starting from an initial belief state

[1]Reported running times are the cputime measured on 3 GHz linux machines.

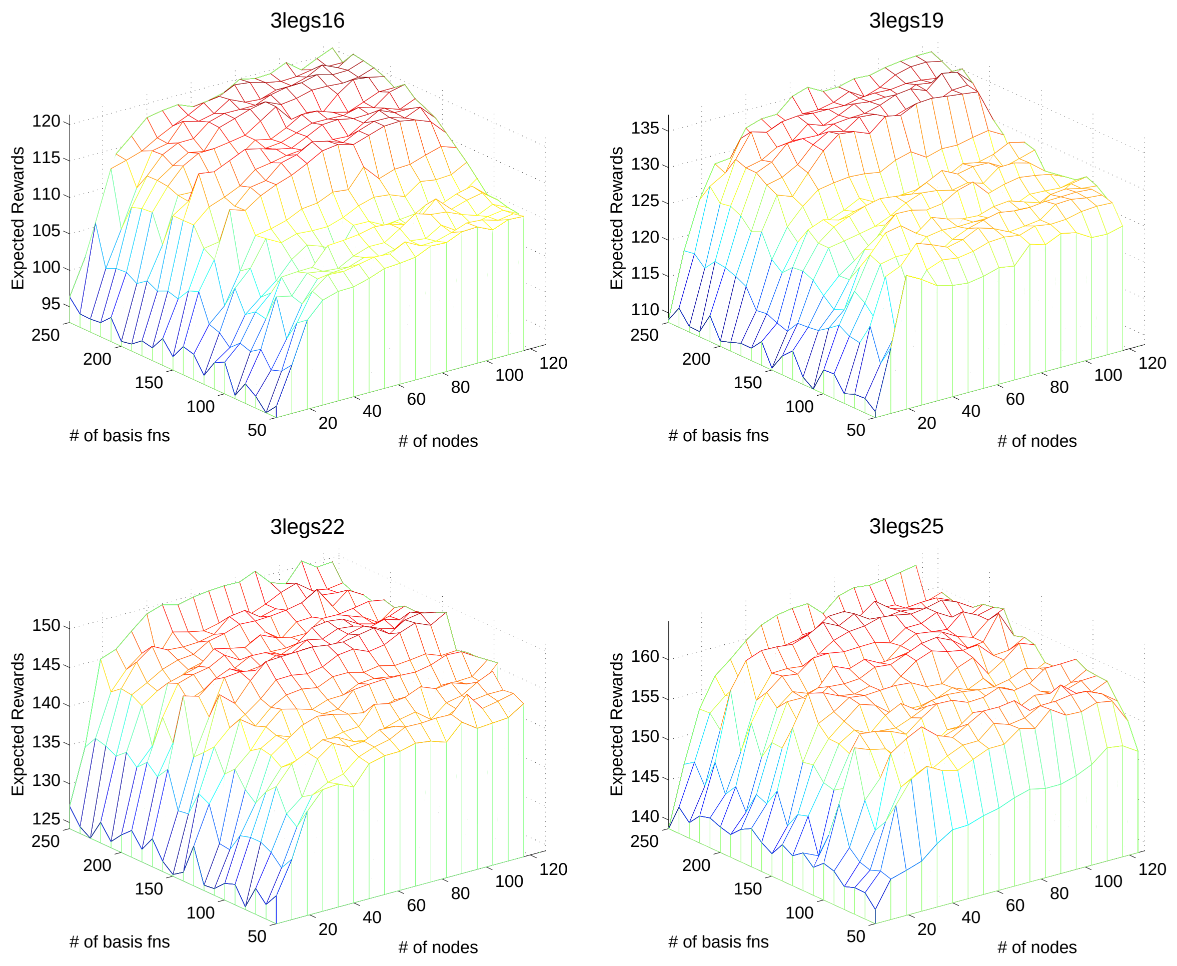


Figure 5.1: Expected reward of policies found by compressed BPI for *3legs* networks of 16, 19, 22 and 25 machines.

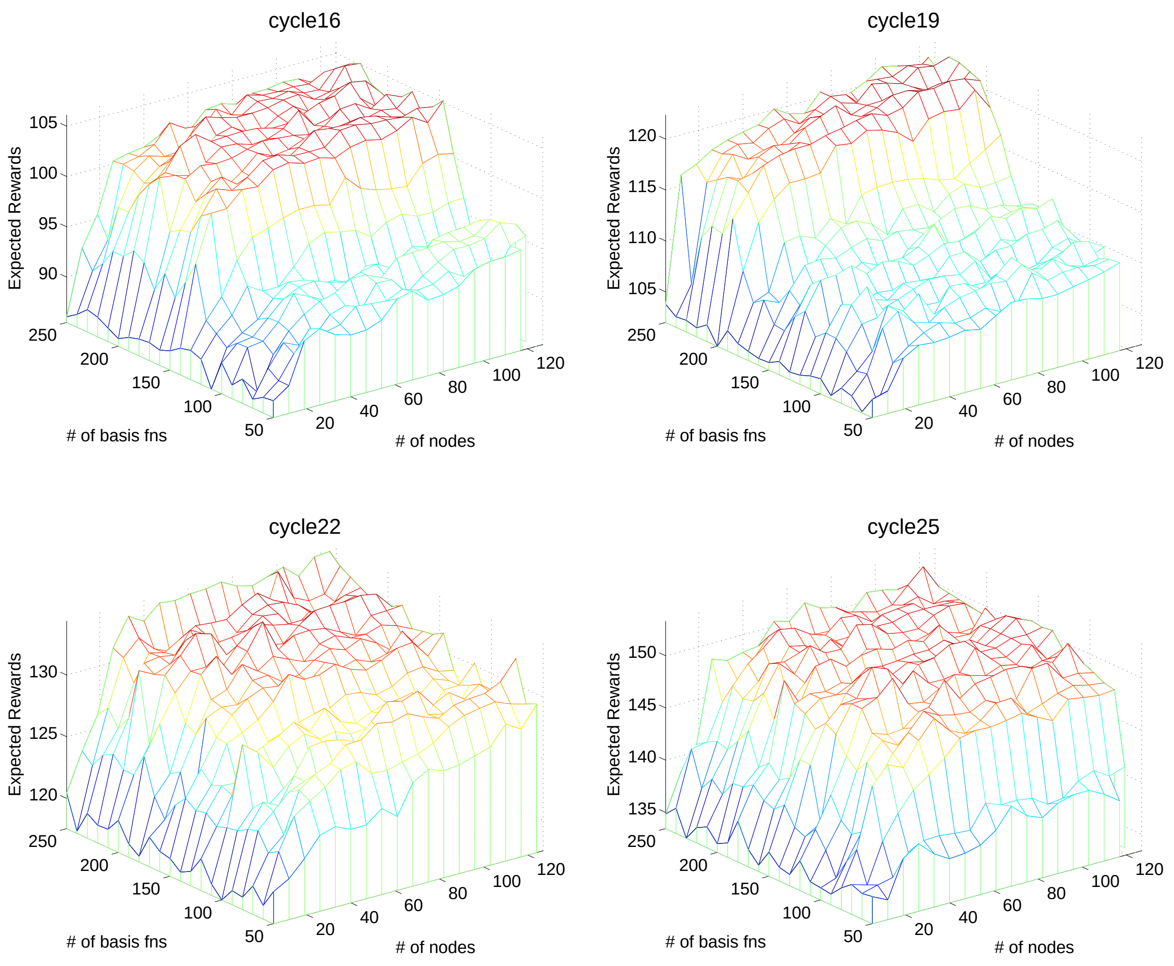


Figure 5.2: Expected reward of policies found by compressed BPI for *cycle* networks of 16, 19, 22 and 25 machines.

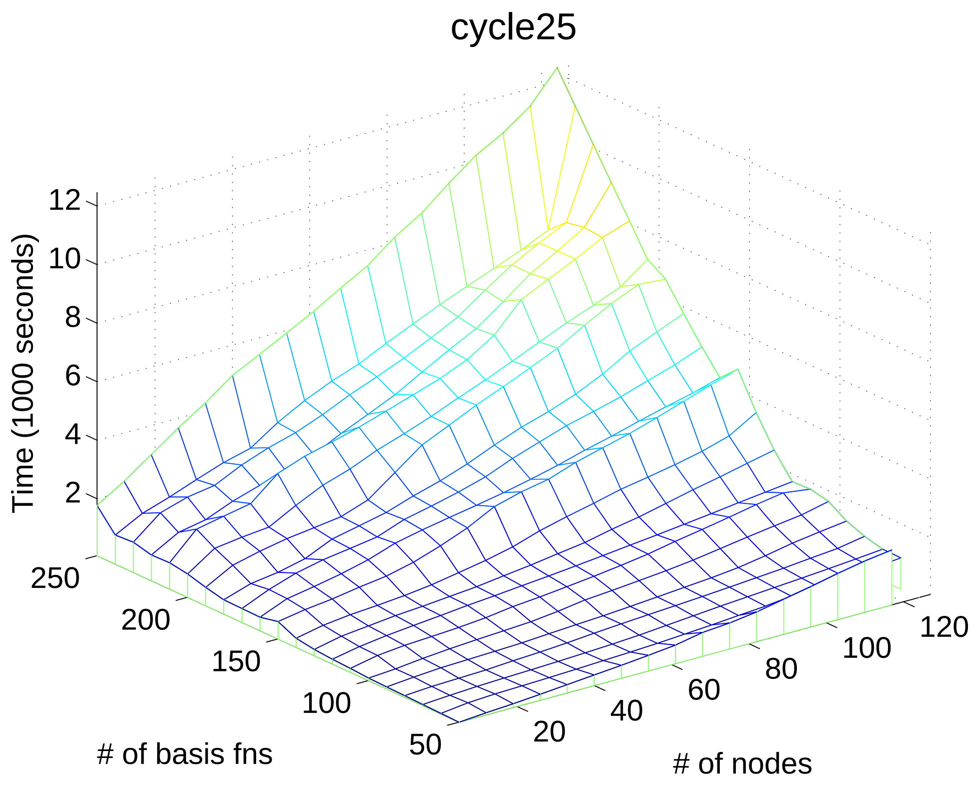


Figure 5.3: Running time of BPI on compressed versions of a *cycle* network of 25 machines as we vary the number of basis functions and the number of nodes.

where all machines are running. The results reported for compressed BPI and Perseus are those for the best policy found with fewer than 120 nodes (BPI) or fewer than 50 iterations (Perseus) after compressing the problems with 250 basis functions. The *doNothing* policy is a passive one that lets the network evolve without rebooting nor pinging any machine. The *heuristic* policy estimates the probability of failure[2] of each machine at each stage and reboots the machine most likely to be down if its failure probability is greater than threshold p_1 or pings it when it is greater than threshold p_2. Settings of $p_1 = 0.8$ and $p_2 = 0.15$ were used.[3] This heuristic policy performs very well and therefore offers a strong competitor to compressed BPI and Perseus. It is possible to do better than the heuristic policy by optimizing the choice of the machine that the system administrator may reboot or ping. Since a machine is more likely to fail when neighboring machines are down, it is sometimes better to choose (for reboot) a machine surrounded by working machines. However, since the system administrator doesn't know exactly, which machines are up or down due to partial observability, such a tradeoff is difficult to evaluate and sometimes not worthwhile. In general, compressed BPI and Perseus outperform the heuristic policy on the *3legs* problems and match it on the *cycle* problems. Note also that compressed BPI tends to generate slightly better policies than compressed Perseus; however, compressed Perseus is significantly faster than compressed BPI. The running times are those for BPI and Perseus alone (i.e., excluding the time taken by truncated Krylov iteration to compress the problems since it is the same). The longer running times for BPI are explained by the need to solve linear programs, which Perseus avoids. Finally, the solution size reported is the number of α-vectors used to represent the value function.

We also tried symbolic Perseus on the same network problems. Unfortunately, the ADDs used to represent α-vectors became too large after a few iterations and the algorithm ran out of memory. Even though the network problems exhibit a significant amount of additive separability, ADDs are not able to exploit this type of structure. In contrast, VDC does exploit additive separability, which explains the success of compressed BPI and Perseus.

In summary, this experiment clearly demonstrates the need to mitigate *both* sources of intractability. First, given the exponentially large state space, techniques that enumerate

[2]Due to the large state space, approximate monitoring was performed by factoring the joint.

[3]These values were determined through enumeration of all threshold combinations in increments of 0.05, choosing the best for 25-machine problems.

Problems	Algorithms	Expected Reward	Solution Size	Time (seconds) compression	solution	total
3legs16	BPI	fail	–	–	–	–
	BPI+VDC	120.9	126	4902	12518	17420
	Perseus	fail	–	–	–	–
	Perseus+VDC	115.4	152	4902	809	5711
	Perseus+ADD	fail	–	–	–	–
	heuristic	100.6	0	0	0	0
	doNothing	98.4	0	0	0	0
3legs19	BPI	fail	–	–	–	–
	BPI+VDC	137.0	65	5143	2866	8009
	Perseus	fail	–	–	–	–
	Perseus+VDC	126.6	30	5143	62	5205
	Perseus+ADD	fail	–	–	–	–
	heuristic	118.3	0	0	0	0
	doNothing	112.9	0	0	0	0
3legs22	BPI	fail	–	–	–	–
	BPI+VDC	151.0	111	6143	5499	11642
	Perseus	fail	–	–	–	–
	Perseus+VDC	143.1	15	6143	60	6203
	Perseus+ADD	fail	–	–	–	–
	heuristic	138.3	0	0	0	0
	doNothing	133.5	0	0	0	0
3legs25	BPI	fail	–	–	–	–
	BPI+VDC	164.8	123	7097	6596	13693
	Perseus	fail	–	–	–	–
	Perseus+VDC	162.6	33	7097	211	7308
	Perseus+ADD	fail	–	–	–	–
	heuristic	152.3	0	0	0	0
	doNothing	147.1	0	0	0	0

Table 5.4: Results for the 3legs networks. “Fail” indicates that the algorithm ran out of memory.

Problems	Algorithms	Expected Reward	Solution Size	Time (seconds) compression	solution	total
cycle16	BPI	fail	–	–	–	–
	BPI+VDC	103.9	107	12408	20723	33131
	Perseus	fail	–	–	–	–
	Perseus+VDC	103.6	65	12408	250	12658
	Perseus+ADD	fail	–	–	–	–
	heuristic	102.5	0	0	0	0
	doNothing	91.6	0	0	0	0
cycle19	BPI	fail	–	–	–	–
	BPI+VDC	121.3	103	11582	16884	28466
	Perseus	fail	–	–	–	–
	Perseus+VDC	111.2	89	11582	575	12157
	Perseus+ADD	fail	–	–	–	–
	heuristic	117.9	0	0	0	0
	doNothing	105.4	0	0	0	0
cycle22	BPI	fail	–	–	–	–
	BPI+VDC	134.3	72	7124	5319	12443
	Perseus	fail	–	–	–	–
	Perseus+VDC	129.5	69	7124	295	7319
	Perseus+ADD	fail	–	–	–	–
	heuristic	130.2	0	0	0	0
	doNothing	122.0	0	0	0	0
cycle25	BPI	fail	–	–	–	–
	BPI+VDC	151.4	68	8511	6940	15451
	Perseus	fail	–	–	–	–
	Perseus+VDC	148.0	57	8511	63	8574
	Perseus+ADD	fail	–	–	–	–
	heuristic	152.3	0	0	0	0
	doNothing	140.1	0	0	0	0

Table 5.5: Results for the cycle networks. “Fail” indicates that the algorithm ran out of memory.

states such as BPI and Perseus run out of memory. ADDs can often aggregate states by exploiting conditional and context-specific independence, but for these network problems, the exhibited independence was not sufficient to yield significant aggregations. VDC is the only technique (among those tested) that sufficiently reduced the complexity of the state space by exploiting additive separability with conditional and context-specific independence. As for policy complexity, the fact that the simple doNothing and heuristic policies perform well suggests that simple yet good policies can be found. Indeed, BPI and Perseus were both able to find relatively small policies that outperform the doNotyhing and heuristic policies. Hence the only two practical algorithms (among those tested) are Perseus+VDC and BPI+VDC. The former is significantly faster since Perseus doesn't need to solve linear programs; however, the latter finds slightly better policies.

5.3.2 Task Assistance for Persons with Dementia

The second set of experiments features an automated system designed to assist persons with dementia. People with dementia suffer from a deterioration of their cognitive faculties often resulting in memory losses and lack of autonomy. As a result, they have difficulty completing basic activities of daily living such as handwashing, toileting, dressing, eating, taking medication, etc. In collaboration with the Intelligent Assistive Technology and Systems Lab in the Department of Occupational Therapy at the University of Toronto, I designed a control system modeled as a POMDP that guides patients with memory deficiencies through the steps of handwashing by monitoring their progress with video cameras and, when necessary, prompting the next step with a verbal cue. The system represents a real-world POMDP which I next describe at a high level.

The automated system, dubbed COACH, assists a person in completing the task of handwashing. Figure 5.4 shows the sink area of a retrofitted bathroom at the Sunnybrook hospital in Toronto where patients can wash their hands. A camera mounted above the sink monitors patient movements and a hidden speaker emits verbal cues (when necessary) to guide a patient. The control system in COACH is modeled as a POMDP and is responsible for deciding when to emit a cue as well as the type of cue based on the information provided by the camera. The decision process is fraught with uncertainty, both with respect to the observability of the environment (e.g., occluded views and noisy image processing) and the effects of actions (e.g., patients do not always follow the prompts). Furthermore, the goal of the system is to satisfy a number of different objectives—some

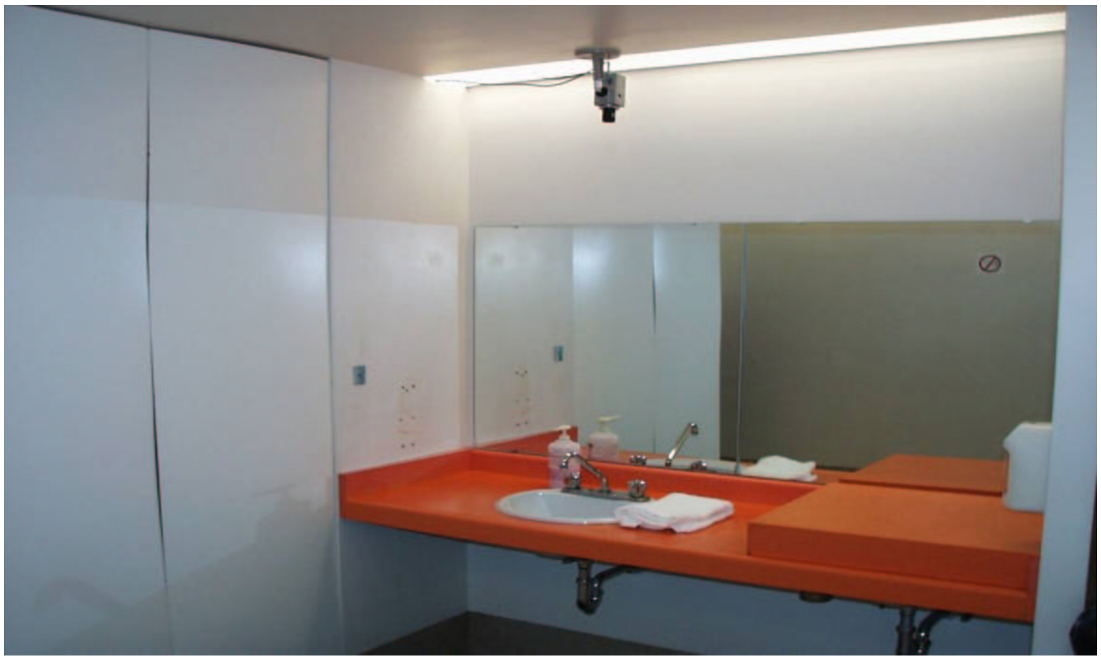


Figure 5.4: Retrofitted bathroom at the Sunnybrook hospital in Toronto for the COACH project.

more important than others—that often conflict and cannot be achieved with certainty (e.g., maximizing the probability of task completion, maximizing the number of successful steps without any help from the caregiver, etc.). The system should also have the ability to learn about each patient to adapt to their preferences since different prompting strategies will work better with each user. In that respect, POMDPs provide an ideal framework since they allow a system to plan under uncertainty with respect to conflicting goals while learning about unknown environment or user characteristics. In particular, optimal POMDP policies naturally choose courses of action that balance the importance of specific objectives with their odds of success. They also account for the long-term impact of decisions, including the value of information inherent in each action, which allows the system to actively learn about unknown parameters such as user preferences. A somewhat simplified POMDP model is now described for the handwashing task.

State variables

The state space is characterized by 4 classes of variables: those that capture the state of the *environment*, those that summarize the *steps of handwashing* that have been completed so far, those that summarize *system behavior*, and those reflecting certain hidden aspects of the patient's *personality* or *mental state*. Environment variables represent the underlying physical state of the environment. The variables used are HL (hands location: at tap, at soap, at towel, at sink, at water, away) and WF (water flow: on or off). Plan-step variables capture various aspects of the handwashing steps that the user has completed. Figure 5.5 shows the legitimate sequences of steps (any path from start to finish) that constitute a successful completion of the handwashing task. Variable PS, whose domain is the set of nodes of the plan graph (excluding L, which is shown for convenience), denotes the last step completed by the user. There are also MaxPS MaxPSRepeat variables denoting respectively the most advanced plan-step successfully completed by the user and whether that most advanced plan-step changed or repeated at the last step. These will be useful to keep track of the steps already completed in case the patient backtracks in the plan graph. System behavior variables provide a summary of the history relevant to the prediction of patient responses to a prompt. These are NP (number of prompts issued for the current plan-step: 0, 1, 2, 3, 4+), NW (number of time-steps waited since the last prompt: 0, 1, 2, 3, 4+), LP (the type of the last prompt, corresponding to the action types described below) and PL (the level of specificity of the last prompt: general, moderate or specific). Finally, user variables reflect the aspects of the user's mental state that can impact their response to prompts. The current prototype uses only two variables to provide a very crude characterization of patient type: Dm (level of dementia: high or low) and Resp (general responsiveness: high or low).

Actions and dynamics

Actions consist of playing prompts associated with each of the plan-steps in Figure 5.5 (e.g., use soap, turn on water, wet hands, rinse hands, turn off water, dry hands), doing nothing (e.g., wait) or calling the caregiver. Calling the caregiver is a terminal action that ends the process, and is presumed to result in successful task completion. Each prompt has associated with it three levels of specificity (e.g., general, moderately specific and specific). Transition probabilities describe the stochastic state changes induced by each action. These can be specified by a dynamic Bayesian network (DBN) over the state variables, with the conditional probability tables (CPTs) represented by ADDs. The transitions exhibit considerable context specific independence, leading to a relatively compact specification of the system dynamics.

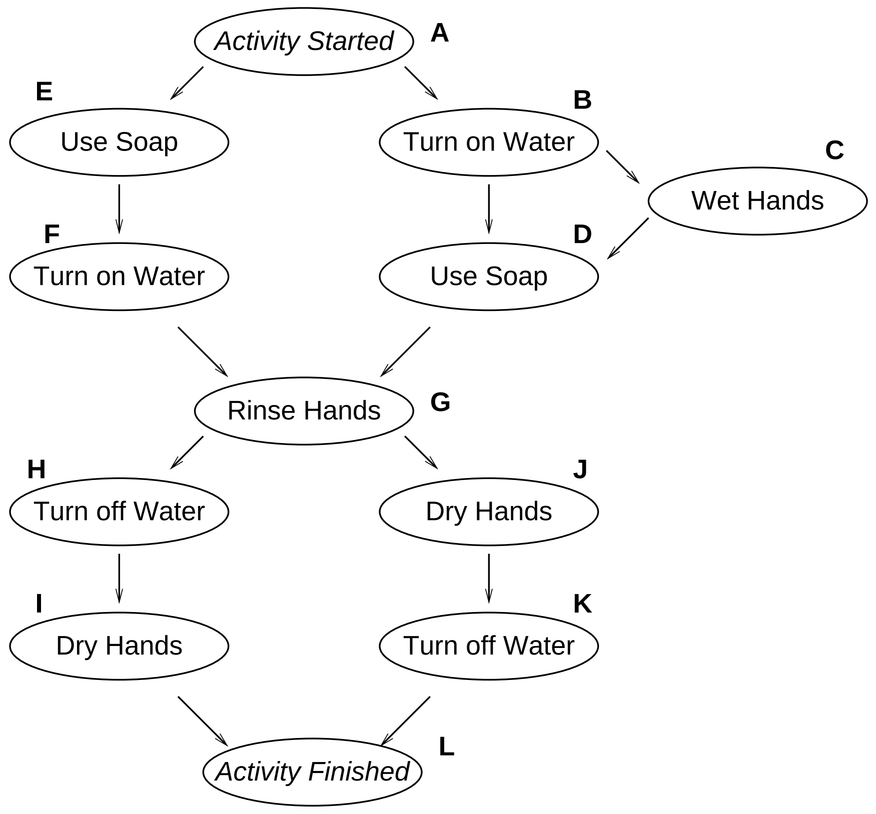


Figure 5.5: Possible sequences of subtasks to complete the handwashing task.

Since the details of the transition probabilities are beyond the scope of this thesis, only a high level overview is provided.[4] The environment variables WF and HL change stochastically depending on the current plan-step PS, the system variables NP, NW, PL, LP and the user variables. For instance, when a "turn on water" prompt is given, the probability that WF becomes "on" increases with the specificity of the prompt, when the last completed plan-step is A or E, and when the patient has low dementia and high responsiveness.

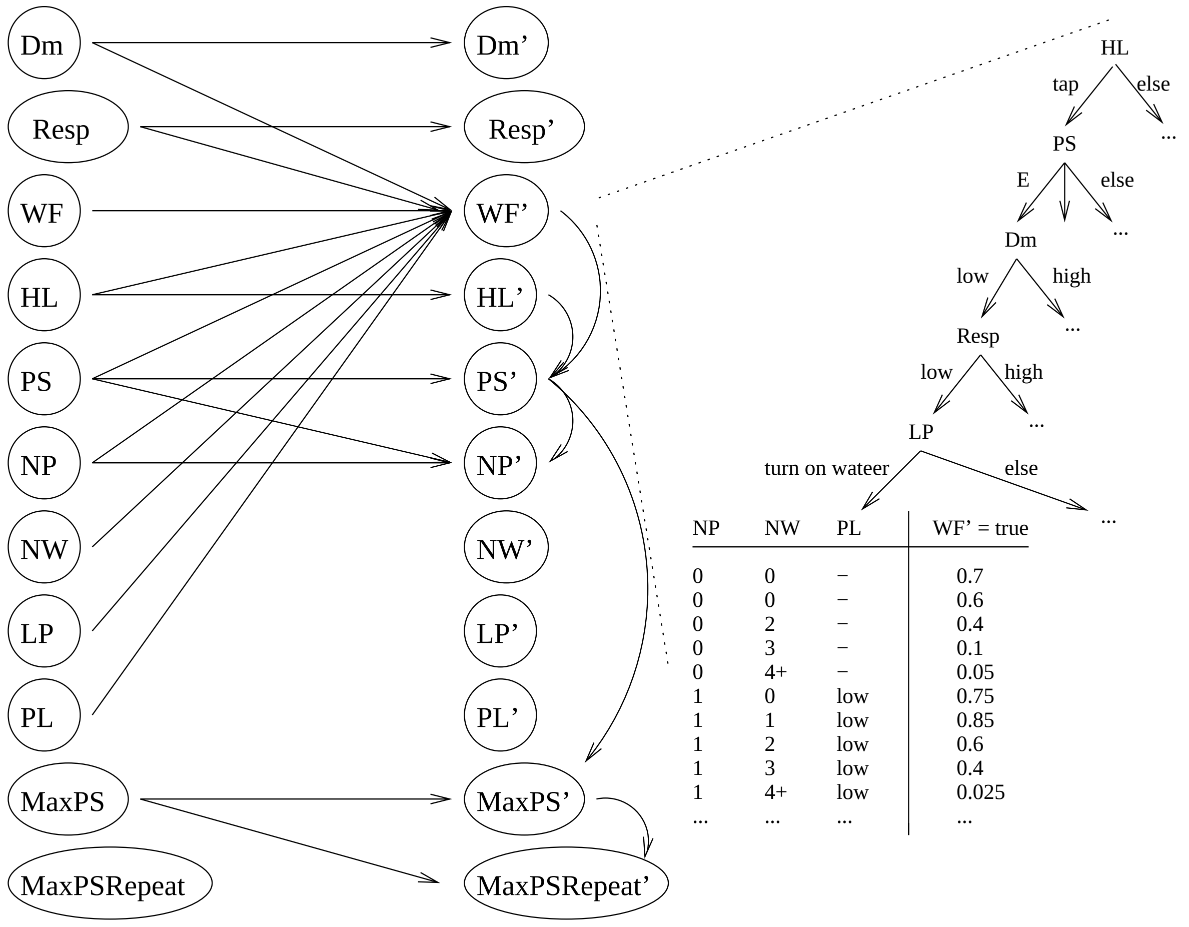
Figure 5.6 illustrates the transition dynamics for the "turn on water" prompt with a dynamic Bayesian network. A partial conditional probability table represented as a decision tree describes the dependencies of WF on prompt history (NP, NW and PL). The probability that the water will be turned on by the patient when prompted to do so increases with the first prompt, but decreases with subsequent prompts for the same step. Furthermore, the longer one waits for a response, the less likely it is to spontaneously occur. If the prior prompt was specific, the odds of response are lower still.

In general, the odds of a patient following a prompt also depend on the time (number of waits) since the last prompts as well as whether the prompt was repeated. The plan-step variables PS, MaxPS and MaxPSRepeat are updated deterministically based on the current estimates of the environment variables. For instance, when PS and MaxPS are B, if HL becomes "at soap", then PS and MaxPS advance to D. However, if WF becomes "off" then PS backtracks to A and MaxPS remains unchanged. Similarly the system variables NP, NW, LP and PL are also updated deterministically based on the history of past actions. Finally, the user variables are static and do not change values over time (though the system's beliefs do).

Observation variables

The system is equipped with a camera that monitors the sink area. Based on the images, it is possible to infer (with some noise) the position of the hands as well as the water flow. Therefore an observation variable OHL is used to denote the observed hand location, which is assumed to return the correct location 85% of the time and each incorrect position 3% of the time. Similarly an observation variable OWF denotes the observed water flow with 95% accuracy. There are no other observation variables since

[4]The transition probabilities were elicited from a domain expert, Jennifer Boger, at the Intelligent Assistive Technology and Systems Lab in the Department of Occupational Therapy at the University of Toronto.



NP	NW	PL	WF' = true
0	0	−	0.7
0	0	−	0.6
0	2	−	0.4
0	3	−	0.1
0	4+	−	0.05
1	0	low	0.75
1	1	low	0.85
1	2	low	0.6
1	3	low	0.4
1	4+	low	0.025
...	...	...	...

Figure 5.6: Dynamic Bayesian network structure of the transition function for the "turn water on" prompt. A partial CPT represented as a decision tree is given for WF.

the plan-step and the system variables are updated deterministically, and user variables are unobservable.

It is interesting to note that the hidden user variables Dm and Resp can be viewed as unknown parameters of the system dynamics. Reinforcement learning techniques could be used to infer the values of those parameters, but the system can also learn indirectly while executing a policy simply by updating its beliefs about the levels of dementia and responsiveness in response to the observations made for OHL and OWF. Intuitively, when a patient tends to follow a prompt quickly, then it is more likely to have low dementia and high responsiveness. Since the optimal policy of a POMDP executes the best action in each belief state, the course of action of an optimal policy naturally adapts to the user characteristics captured by the current belief state.

Rewards

A large reward of +100 is earned when the handwashing task is completed, that is when PS $\in \{I, K\}$. Smaller rewards of +3 are earned with the completion of each plan-step. The relative magnitude of these rewards is designed to encourage the resulting policy to pursue full task completion, but if the odds of successful completion are very low, to at least attempt to make progress through the plan. The progress rewards are associated with the MaxPS and MaxPSRepeat variables, rather than the PS variable to prevent a policy that encourages repeated achievement of the same plan-step (such as repeatedly turning the water on and off). Action costs are also incorporated to ensure that prompting only occurs when needed. Each prompt is given a small negative reward (e.g., -1, -2, -3), with the more time-consuming specific prompts penalized slightly more. Calling the caregiver has a much higher cost of -70, but still of smaller magnitude than the +100 for completing the task. This ensures that calling the caregiver will occur if the predicted odds of completion are too low (or predicted costs too high).

Experimental results

The resulting POMDP model has 33,454,080 states, 23 actions and 12 observations. Table 5.6 indicates the expected reward (averaged over 500 policy runs of 60 steps, starting from the initial belief state where Dm and Resp are uniformly distributed, PS=A, MaxPS=A, MaxPSRepeat=yes, NW=0, NP=0, LP=doNothing, PL=general, WF=off, HL=away). Results can be compared for compressed BPI (BPI+VDC), compressed

Perseus (Perseus+VDC), symbolic Perseus (Perseus+ADDs) as well as a few other heuristic policies. DoNothing corresponds to a passive policy that doesn't give any prompt to the patient. CallCaregiver corresponds to the very conservative policy of always requesting help from the caregiver, ensuring that the task completes.

Table 5.6 also includes an algorithm (MDPVI+ADDs) that computes a policy by treating the POMDP as a fully observable MDP since the problem is almost fully observable. This algorithm corresponds to the SPUDD algorithm [49], which performs symbolic value iteration for MDPs by using ADDs to represent the value function. Since the value function of MDPs consists of a single α-vector, MDPVI+ADDs runs significantly faster than the more general POMDP algorithms. On the other hand, since the resulting policy maps each state to some action, execution of the policy is complicated by the fact that several state variables are not observable nor deterministic. In general, we can always ignore partial observability by computing the most likely state and executing its corresponding action. For the COACH problem, this strategy performs really well (value of 37.6) given that the problem is almost fully observable due to the deterministic nature of the system and plan-step variables as well as the relatively high accuracy of the observations for the environment variables.

Since the dynamics of the problem exhibit a significant amount of context-specific independence, symbolic Perseus can effectively mitigate both sources of intractability to compute the best policy (value of 39.1) in Table 5.6. It also takes advantage of the fact that the handwashing task always starts in the same initial belief state to construct a policy tailored to the reachable belief region.

The results reported for compressed BPI and Perseus are for lossy compressions of 250 dimensions. Since the compressions were too lossy, BPI and Perseus converged to poor policies even though they appeared reasonable in the compressed space. We also tried compressions with up to 500 basis functions, but the quality of the resulting policies did not improve. Since VDC computes compressions for *all* policies, it is unable to find a relatively good compression with a small number of basis functions. In contrast, Perseus+ADDs represents the α-vectors of a *single* policy at each iteration, which explains its success.

The value of the optimal policy is unknown; however, it is possible to compute an upper bound. Since the optimal policy of a POMDP has access to less information than the optimal policy of the corresponding fully observable MDP, the value (with respect to the MDP model) of the policy found by MDPVI+ADDs provides an upper bound of

Algorithms	Expected	Time (seconds)		
	Reward	compression	solution	total
Perseus+ADD	39.1	–	–	128490
MDPVI+ADD	37.6	–	–	167
callCaregiver	25.0	–	–	0
BPI+VDC	18.1	75168	51242	126410
Perseus+VDC	17.8	75168	15666	90834
doNothing	13.2	–	–	0

Table 5.6: Results for the assistive technology task.

49.6. Note that this value is not realized in practice since the problem is really a POMDP (not an MDP) and that's why the policy obtained by MDPVI+ADDs only earns a value of 37.6. Since the policy obtained by Perseus+ADDs earns a higher value of 39.1, we know that the optimal value is between 39.1 and 49.6.

This experiment demonstrates once more the need to simultaneously mitigate the complexity of policy spaces and the complexity of state spaces. In contrast with the network management problems, ADDs offer a better state space reduction than VDC for this assistive technology task. ADDs turn out to be more effective than VDC because they dynamically aggregate states for a *single* policy. In contrast, VDC compresses the belief space for *all* policies. Furthermore, despite the exponentially and doubly exponentially large policy space (with respect to the observation space and the horizon length), simple yet good policies were found. Symbolic Perseus (Perseus+ADD) was the only algorithm tested that adequately mitigated both sources of intractability and as a result found the policy with the highest value.

In summary, this chapter has demonstrated the benefits of mitigating simultaneously both sources of intractability by yielding solutions to POMDPs significantly larger than those previously tackled. The network management problems and the assistive technology task have state spaces three orders of magnitude larger than the state spaces of previous test problems. Nevertheless, due to the presence of important structural properties, such as conditional independence, context-specific independence and additive separability, VDC and ADDs can sometimes mitigate the complexity of state spaces. As for the policy space, despite its exponential size with respect to the number of observations and doubly exponential size with respect to the horizon length, simple yet good policies were found by focusing on the reachable belief region and giving priority to small policies. Indeed, BPI and Perseus are often capable of quickly finding small yet good policies when they exist. By simultaneously mitigating the complexity of policy and state spaces, compressed BPI, compressed Perseus and symbolic Perseus constitute approximate, scalable algorithms that can tackle relatively large and difficult problems by exploiting structural properties (when present).

Chapter 6

Conclusion

An important problem at the center of artificial intelligence is the design of suitable control policies for automated systems. To that effect, POMDPs provide a rich and principled framework to optimize the course of actions in the presence of state and action uncertainty, incomplete dynamics and multiple interacting objectives. As a result, a wide range of sequential decision problems in robotics, operations research, human-computer interaction, preference elicitation, etc. can be naturally modeled as POMDPs.

Unfortunately, the use of POMDPs in real-world systems remains limited due to the intractability of existing solution algorithms. However, real-world problems tend to exhibit a significant amount of structure that can be exploited to improve the scalability of POMDP algorithms. Hence, this thesis analyzes problem-specific structure and describes new and existing algorithms that exploit them.

6.1 Summary

Two important sources of intractability plague the majority of discrete POMDP algorithms: the complexity of policy and state spaces. In practice, since many POMDPs have simple policies of high quality, it is often possible to circumvent the complexity of policy space by searching within a restricted class of compactly representable policies and by focusing on the reachable belief region. Chapter 3 reviews several algorithms that tackle the complexity of policy and value function spaces, and presents a new bounded policy iteration algorithm. BPI searches for a good policy represented by a small stochastic controller. It distinguishes itself by its robustness to local optima (c.f. gradient ascent) and its scalability (c.f. classic policy iteration, branch and bound, stochastic local search).

Furthermore, useful insights are provided concerning the properties of GA's local optima and PI's limitation to deterministic controllers.

Chapter 4 reviews several structural properties (e.g., sparsity, conditional independence, context-specific independence and additive separability) exhibited by factored POMDPs. These can be exploited by various techniques to mitigate the complexity of large state spaces. In particular, a new value-directed compression (VDC) technique is proposed. Belief states, which summarize histories of past actions and observations, often contain superfluous information irrelevant to the decision process. Since an optimal policy can be found as long as all policies are accurately evaluated, we can prune from belief states any information that doesn't help to estimate future rewards. Sufficient conditions to ensure a lossless compression are proposed as well as a simple Krylov iteration algorithm that finds the best *linear* loss compression. In the event where the best lossless compression is not powerful enough, suitable lossy compressions can be found by solving an optimization program or stopping the Krylov iteration algorithm early. Perhaps one of the most interesting properties of linear VDC is that the resulting compressed POMDPs can be solved (more or less) directly by most algorithms. This means that VDC can be used as a preprocessing step to most POMDP algorithms to mitigate the intractability caused by large state spaces.

To date, all existing algorithms tackle only one of the two sources of intractability. As a result they cannot tackle large POMDPs. Thus, one of the most important contributions of this thesis is the description of three new algorithms that simultaneously tackle both sources of intractability. Chapter 5 describes a compressed version of BPI by combining BPI with VDC as well as compressed and symbolic versions of Perseus by combining Perseus with VDC and ADDs. These algorithms can overcome both sources of intractability to attack POMDPs several orders of magnitude larger than previously possible when suitable structural properties are present.

6.2 Open Problems

Overcoming the complexity of policy and state spaces presents an important step toward scalable algorithms capable of solving large real-world POMDPs. This research can be extended in several directions.

Besides large state spaces, many POMDPs also have large action and observation spaces. These often arise in factored POMDPs when observations correspond to the cross-product of the domains of multiple sensors and actions correspond to the cross-product of the possible decisions for multiple parameters, options or primitive actions. In that case, the number of observations and actions is exponential with respect to the number of observation variables and primitive actions. It would be interesting to investigate the use of symbolic representations (e.g., decision trees, algebraic decision diagrams) or compression techniques to handle large action and observation spaces. In fact, algebraic decision diagrams have already been used to tackle factored MDPs with large action sets [61].

The framework of POMDPs traditionally assumes a propositional representation. One could also specify POMDPs using relational or first-order constructs, allowing for a (potentially) more compact representation of the dynamics. The additional structure captured by relational and first-order representations may be exploited to further mitigate the complexity of large state, action and observation spaces. To date, several algorithms have been proposed for relational [39, 37] and first-order [16, 15] MDPs, but not POMDPs. There also exist probabilistic [4] and decision-theoretic [108] extensions of the situation calculus language, which are equivalent to first-order POMDPs, but again no algorithms have been proposed.

Although this thesis focuses exclusively on discrete POMDPs, it is common in practice to encounter problems with continuous states, actions and observations. For instance, in robot navigation problems, the state space may correspond to a continuous floor plan, the action space to a continuous range of motor controls and the observation space to continuous sonar or laser range finder measurements. For such problems, the dynamics are rarely linear and the value functions rarely have a simple form. One can always discretize the continuous components; however, such an approximation is not always satisfactory. A few approaches have been proposed to directly solve continuous POMDPs by gradient ascent [89, 1, 94], neural networks [8, 122, 123], and Monte Carlo simulation [140]. Further research will be necessary to better understand the structural properties of continuous POMDPs and how to exploit them.

POMDP algorithms traditionally assume the complete specification of transition, observation and reward functions. When parameters of those functions are unknown, one can build a larger POMDP that seamlessly integrates Bayesian learning of those parameters with planning. The idea is to augment the state space with the unknown parameters and to learn their values during the execution of a policy based on the observations made at each time step. Furthermore, optimal policies naturally optimize the exploration/exploitation tradeoff by maximizing the combined value associated with immediate action effects and the information gathered about the unknown parameters. This approach was used in the COACH system (described in Section 5.3.2) to design an adaptive policy that learns the unknown level of dementia and responsiveness while interacting with each patient. Similar approaches were also used in the context of reinforcement learning [81, 34], imitation learning [115], preference elicitation [10], multiagent coordination [24] and coalition formation [25]. In general, the resulting augmented POMDPs are much harder to solve due to the increased complexity of the state space, but they exhibit additional structure. In particular, the dynamics of the unknown parameters that must be learned are quite simple since their values remain static. Further research is necessary to exploit the additional structure exhibited by those POMDPs.

Appendix A

Problem Descriptions

A.1 Preference Elicitation

In the preference elicitation problem, the task of the decision maker is to make a recommendation to a user. However, since the decision maker doesn't know the user's preferences, it can ask queries to gain some information before making a recommendation. The query process followed by the recommendation can be modeled by a POMDP [10] where the actions correspond to the possible queries and recommendations, the observations correspond to the user answers and the states correspond to the possible user preferences. The rewards consists of the cost associated with each query and the value (measured in terms of the user preferences) of the recommendation.

The preference elicitation problem we consider [19, 20] has 14 actions (7 queries and 7 recommendations), 2 observations (`yes` and `no`) and 7 states (6 possible preference functions and an absorbing terminal state). In this simple problem, the user has a (unknown) preference function defined over 7 possible outcomes. Table A.1 gives the utility assigned by each of the preference functions to the 7 possible outcomes. At each step, the decision maker either asks a query or makes a recommendation. Queries have a cost of 0.02 and are of the type "Is the utility of outcome o_i less than 0.9?". The user answers `yes` or `no` with certainty. When the decision maker feels that enough information has been gathered, it makes a recommendation (among 7 possible recommendations $\{r_1, \ldots, r_7\}$). Recommendations r_i, $i \leq 5$, yield outcomes o_i or o_{i+1} with equal probability, while r_6 yields o_6 or o_1 with equal probability, and r_7 yields o_7 with certainty. Once a recommendation is made, the process ends and the decision maker receives a reward corresponding to the utility assigned to the outcome by the user's actual preference function.

	o_1	o_2	o_3	o_4	o_5	o_6	o_7
pf_1	0.9	0.9	0.1	0.1	0.1	0.1	0.3
pf_2	0.1	0.9	0.9	0.1	0.1	0.1	0.3
pf_3	0.1	0.1	0.9	0.9	0.1	0.1	0.3
pf_4	0.1	0.1	0.1	0.9	0.9	0.1	0.3
pf_5	0.1	0.1	0.1	0.1	0.9	0.9	0.3
pf_6	0.9	0.1	0.1	0.1	0.1	0.9	0.3

Table A.1: Utility assigned by each preference function pf_i to each outcome o_j.

This preference elicitation problem has the following interesting property: the cost of a query is higher than the value of the information gained by a single query. As a result, myopic algorithms tend to get stuck in a local optimum that consists of recommending r_7 without asking any query. Here r_7 is a conservative recommendation that yields outcome o_7 with utility 0.3 for all preference functions.

A.2 Heaven and Hell

Heaven and Hell [20] consists of two nearly identical mazes offering a positive reward (heaven) and a negative reward (hell) in opposite locations (see Figure A.1). The agent starts in location s of one of the two mazes, but doesn't know which. It has 4 deterministic actions (move `up`, `down`, `right` or `left`) that fail only when a wall is hit, resulting in no movement. Reaching a reward-bearing state is risky since the reward is positive in one world and negative in the other. When the magnitude of the negative reward is much larger than the positive, it is important for the agent to be fairly certain about the world it is in. It can visit a priest (cells with an arrow) and pay a small price to receive an observation indicating the world it is in. When dropped with equal probability in either world, the optimal policy is to visit the priest, and then aim for the positive state given the world it is in. Since there is a price to pay when visiting the priest, myopic algorithms tend to avoid the priest, gaining no knowledge about the world, and consequently avoiding all reward-bearing states due to the risk involved.

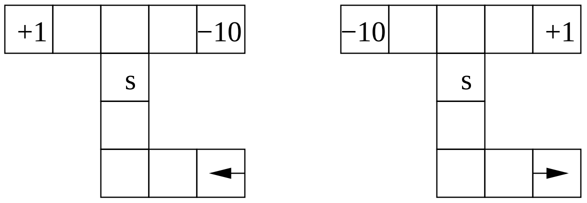


Figure A.1: Heaven and hell problem with left and right mazes.

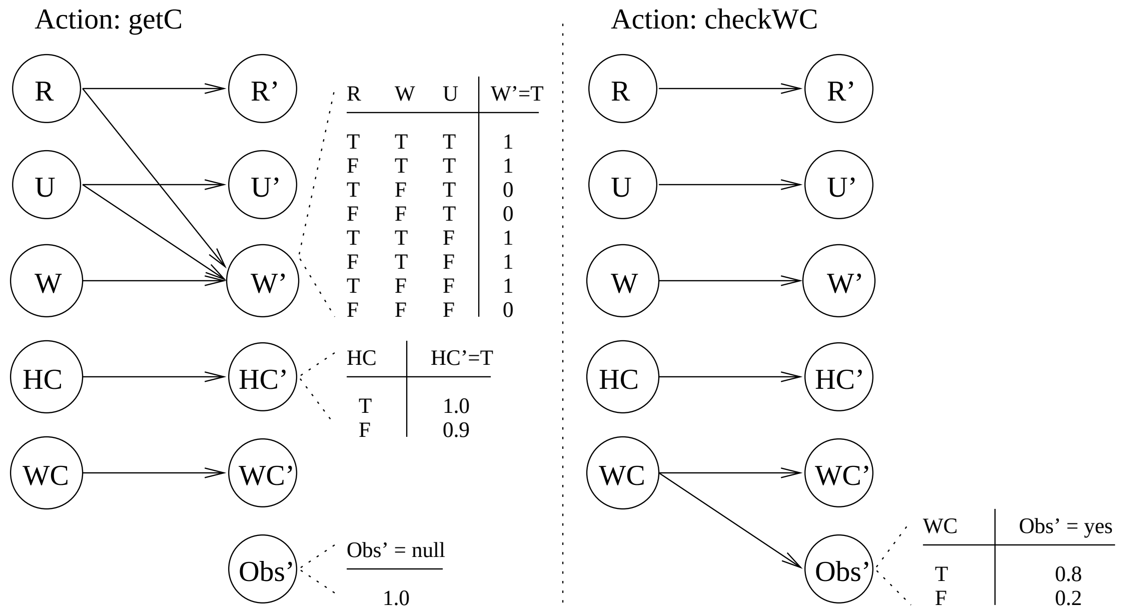
A.3 Coffee Delivery Problem

The coffee problem (proposed by Boutilier and Poole [14]) is a toy POMDP featuring a robot that must purchase coffee at a coffee shop and deliver it to a user. The state space is defined by 5 Boolean variables indicating whether the user has coffee `HC`, the user wants coffee `WC`, it is raining R, the robot is wet W, the robot carries an umbrella U. Figure A.2 describes the transition dynamics. The robot has two actions: get coffee `getC` and check whether the user wants coffee `checkWC`. All variables remain unchanged except those for which a conditional probability table is given. When the robot gets coffee, it must go across the street to the local coffee shop to buy coffee. The robot may get wet if there is rain and it doesn't carry any umbrella. The robot doesn't make any observation when it gets coffee, however it observes (with some noise) whether the user wants coffee when executing the `checkWC` action. The robot earns rewards when the user wants coffee and has coffee, and it is penalized otherwise. The robot is further penalized if it gets wet. A small cost of 1.0 is incurred each time it gets coffee, and 0.5 each time it checks whether the user wants coffee. Figure A.3 describes the reward function.

A.4 Spoken-Dialog System

The spoken-dialog system (developed by Williams [144]) consists of an automated travel agent that must elicit the departure and destination cities of a customer. The agent's task is to interact with the customer to identify the departure and destination cities of its itinerary, however this is complicated by the noisy nature of speech recognition. Figure A.4 gives the structure of the dynamic Bayesian network modelling the task.

The agent can `greet` the customer, `query` the customer (e.g., “Where are you travelling?”), `confirm` statements (e.g., “So you want to go to city a, is that right?”), `submit` a pair of locations to book a flight or simply `end` the conversation. The state space is



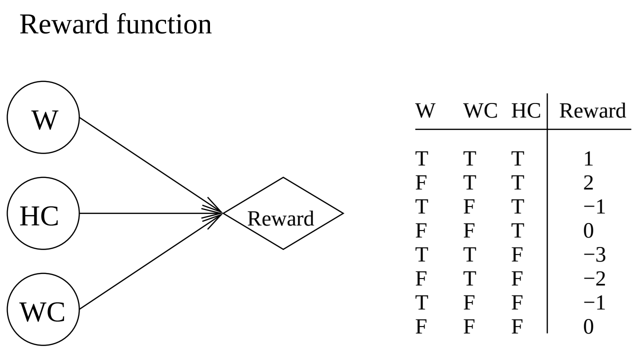
R	W	U	W'=T
T	T	T	1
F	T	T	1
T	F	T	0
F	F	T	0
T	T	F	1
F	T	F	1
T	F	F	1
F	F	F	0

HC	HC'=T
T	1.0
F	0.9

Obs' = null
1.0

WC	Obs' = yes
T	0.8
F	0.2

Figure A.2: Transition dynamics for the coffee delivery problem. Variables remain unchanged unless a conditional probability table is specified.



W	WC	HC	Reward
T	T	T	1
F	T	T	2
T	F	T	−1
F	F	T	0
T	T	F	−3
F	T	F	−2
T	F	F	−1
F	F	F	0

Figure A.3: Reward function for the coffee delivery problem.

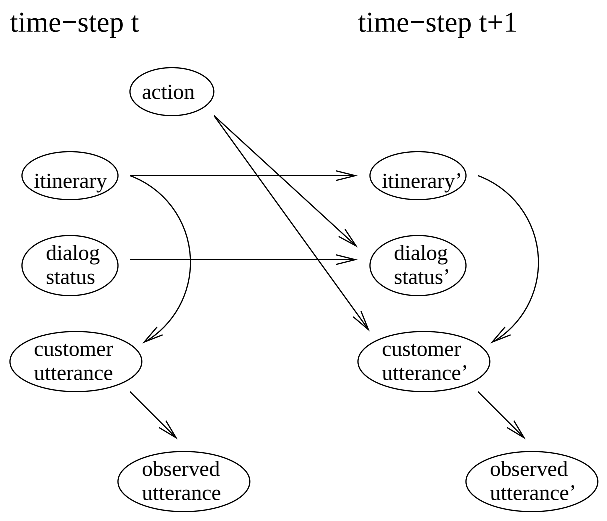


Figure A.4: Dynamic Bayesian network of the spoken-dialog system.

defined by the customer's `itinerary` (i.e., departure and arrival cities), the customer's `last utterance` and the `status` of the dialog (e.g., the itinerary is not mentioned, mentioned but not grounded, or grounded). In the reduced model considered, there are 3 possible cities, 2 queries (`ask-departure` and `ask-arrival`, 6 confirmation statements (3 possible departures and 3 possible arrivals), and 18 utterances (3 cities, 3 departures, 3 arrivals, 6 departure-arrival pairs, `yes`, `no` or `null`).

The customer's `itinerary` is a static variable that never changes — though the agent's belief about the customer's itinerary varies over time. The dialog `status` evolves over time according to the finite state automaton in Figure A.5. Initially, the departure and arrival cities are not mentioned. Once the agent recognizes a city, it becomes mentioned but not grounded. After it is recognized a second time, it becomes grounded. The customer's utterances depend probabilistically on the agent's queries and the customer's itinerary (see Table A.2).

Due to the noisy nature of speech recognition, the agent's observations are `perceived utterances` with a `confidence level` (e.g., high or low) correlated with the customer's actual utterances. The probability of perceiving a correct utterance with high confidence is 0.425, a correct utterance with low confidence is 0.35, an incorrect utterance with low confidence is 0.015 and an incorrect utterance with high confidence is 0.004.

The agent is rewarded +10 for submitting a correct itinerary, −10 for an incorrect itinerary, −5 for ending the dialog without submitting any itinerary. The agent also gets

P(utterance = null | itinerary = (x_1,x_2), action) = 0.1

P(utterance = from-x_1-to-x_2 | itinerary = (x_1,x_2), action = greet) = 0.54
P(utterance = from-x_1 | itinerary = (x_1,x_2), action = greet) = 0.18
P(utterance = to-x_2 | itinerary = (x_1,x_2), action = greet) = 0.18

P(utterance = from-x_1 | itinerary = (x_1,x_2), action = query-from) = 0.225
P(utterance = x_1 | itinerary = (x_1,x_2), action = query-from) = 0.585
P(utterance = from-x_1-to-x_2 | itinerary = (x_1,x_2), action = query-from) = 0.09

P(utterance = to-x_2 | itinerary = (x_1,x_2), action = query-from) = 0.225
P(utterance = x_2 | itinerary = (x_1,x_2), action = query-from) = 0.585
P(utterance = from-x_1-to-x_2 | itinerary = (x_1,x_2), action = query-from) = 0.09

P(utterance = from-x_1 | itinerary = (x_1,x_2), action = confirm-from-x_3) = 0.03375
P(utterance = x_1 | itinerary = (x_1,x_2), action = confirm-from-x_3) = 0.10125
P(utterance = yes | itinerary = (x_1,x_2), action = confirm-from-x_3) = 0.765, $x_1 = x_3$
P(utterance = no | itinerary = (x_1,x_2), action = confirm-from-x_3) = 0.765, $x_1 \neq x_3$

P(utterance = to-x_2 | itinerary = (x_1,x_2), action = confirm-to-x_3) = 0.03375
P(utterance = x_2 | itinerary = (x_1,x_2), action = confirm-to-x_3) = 0.10125
P(utterance = yes | itinerary = (x_1,x_2), action = confirm-to-x_3) = 0.765, $x_2 = x_3$
P(utterance = no | itinerary = (x_1,x_2), action = confirm-to-x_3) = 0.765, $x_2 \neq x_3$

Table A.2: Transition function for customer utterances. Here, x_1, x_2 and x_3 are variables representing possible cities.

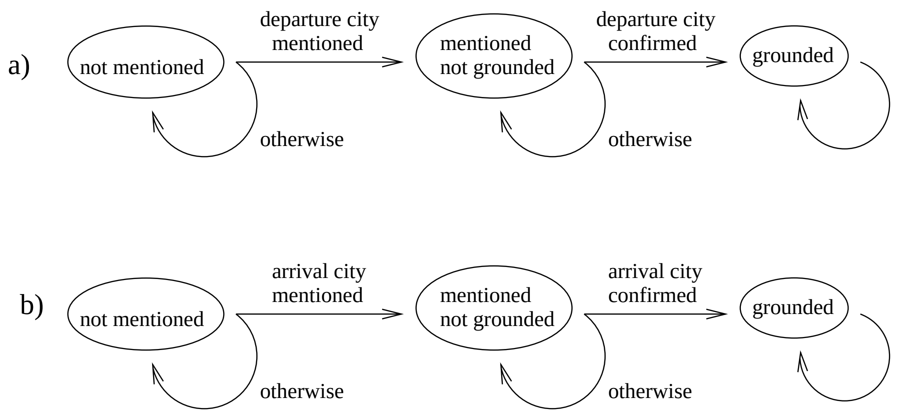


Figure A.5: Finite-state automaton describing the evolution of the dialog status: departure city (a) and arrival city (b).

-3 for asking about the departure or arrival city when it has already been mentioned and -2 for confirming the departure or arrival city when it has already been grounded. When none of the above rules apply, a cost of -1 is incurred at each step.

Bibliography

[1] Douglas Aberdeen and Jonathan Baxter. Scaling internal-state policy-gradient methods for POMDPs. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 3–10, Sydney, Australia, 2002.

[2] K. J. Aström. Optimal control of Markov decision processes with incomplete state estimation. *Journal of Mathematical Analysis and Applications*, 10:174–205, 1965.

[3] Fahiem Bacchus and Adam Grove. Graphical models for preference and utility. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 3–10, Montreal, 1995.

[4] Fahiem Bacchus, Joseph Y. Halpern, and Hector J. Levesque. Reasoning about noisy sensors and effectors in the situation calculus. *Artificial Intelligence*, 111:171–208, 1999.

[5] Andrew Bagnell and Jeff Schneider. Covariant policy search. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 1019–1024, Acapulco, Mexico, 2003.

[6] Andrew Bagnell and Jeff Schneider. Policy search by dynamic programming. In *Advances in Neural Information Processing Systems*, Vancouver, BC, 2003.

[7] Leemon Baird and Andrew Moore. Gradient descent for general reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 968–974, 1998.

[8] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-dynamic programming.* Athena Scientific, Belmont, MA, 1996.

[9] Avrim Blum and John Langford. Probabilistic planning in the Graphplan framework. In *Proceedings of the Fifth European Conference on Planning*, pages 319–332, Durham, UK, 1999.

[10] Craig Boutilier. A POMDP formulation of preference elicitation problems. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, pages 239–246, Edmonton, AB, 2002.

[11] Craig Boutilier, Richard Dearden, and Moises Goldszmidt. Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121:49–107, 2000.

[12] Craig Boutilier, Nir Friedman, Moises Goldszmidt, and Daphne Koller. Context-specific independence in Bayesian networks. In *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence*, pages 115–123, Portland, OR, 1996.

[13] Craig Boutilier, Relu Patrascu, Pascal Poupart, and Dale Schuurmans. Constraint-based optimization with the minimax decision criterion. In *Proceedings of the Ninth International Conference on Principles and Practice of Constraint Programming*, pages 168–182, Kinsale, Ireland, 2003.

[14] Craig Boutilier and David Poole. Computing optimal policies for partially observable decision processes using compact representations. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 1168–1175, Portland, OR, 1996.

[15] Craig Boutilier, Raymond Reiter, and Bob Price. Symbolic dynamic programming for first-order MDPs. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 690–697, Seattle, WA, 2001.

[16] Craig Boutilier, Raymond Reiter, Mikhail Soutchanski, and Sebastian Thrun. Decision-theoretic, high-level agent programming in the situation calculus. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, pages 355–362, Austin, Texas, 2000.

[17] Xavier Boyen and Daphne Koller. Tractable inference for complex stochastic processes. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 33–42, Madison, WI, 1998.

[18] Ronen I. Brafman. A heuristic variable-grid solution method for POMDPs. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 727–733, Providence, 1997.

[19] Darius Braziunas. Stochastic local search for POMDP controllers. Master's thesis, University of Toronto, Toronto, 2003.

[20] Darius Braziunas and Craig Boutilier. Stochastic local search for POMDP controllers. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, pages 690–696, San Jose, CA, 2004.

[21] Anthony R. Cassandra, Leslie Pack Kaelbling, and James A. Kurien. Acting under uncertainty: Discrete Bayesian models for mobile robot navigation. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 963–972, 1996.

[22] Anthony R. Cassandra, Leslie Pack Kaelbling, and Michael L. Littman. Acting optimally in partially observable stochastic domains. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 1023–1028, Seattle, 1994.

[23] Anthony R. Cassandra, Michael L. Littman, and Nevin L. Zhang. Incremental pruning: A simple, fast, exact method for POMDPs. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, pages 54–61, Providence, RI, 1997.

[24] Georgios Chalkiadakis and Craig Boutilier. Coordination in multiagent reinforcement learning: A Bayesian approach. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 709–716, Melbourne, Australia, 2003.

[25] Georgios Chalkiadakis and Craig Boutilier. Bayesian reinforcement learning for coalition formation under uncertainty. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1090–1097, New York, NY, 2004.

[26] Hsien-Te Cheng. *Algorithms for Partially Observable Markov Decision Processes.* PhD thesis, University of British Columbia, Vancouver, 1988.

[27] Lonnie Chrisman. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 183–188, Los Altos, CA, 1992.

[28] Vasek Chvatal. *Linear Programming.* W. H. Freeman & Company, 1983.

[29] Anne Condon. The complexity of stochastic games. *Information and Computation*, 96(2):203–224, 1992.

[30] Thomas Dean and Keiji Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142–150, 1989.

[31] A. Doucet, S. J. Godsill, and C. Andrieu. On sequential Monte Carlo sampling methods for Bayesian filtering. *Statistics and Computing*, 10:197–208, 2000.

[32] A. Drake. *Observation of a Markov process through a noisy channel.* PhD thesis, Massachusetts Institute of Technology, 1962.

[33] Denise Draper, Steve Hanks, and Daniel Weld. A probabilistic model of action for least-commitment planning with information gathering. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 178–186, Seattle, 1994.

[34] Michael Duff. *Optimal Learning: Computational procedures for Bayes-adaptive Markov decision processes.* PhD thesis, University of Massassachusetts Amherst, 2002.

[35] J. Eckles. *Optimum replacement of stochastically failing systems.* PhD thesis, Department of Engineering-Economic Systems, Stanford University, Palo Alto, CA, 1966.

[36] Zhengzhu Feng and Eric A. Hansen. Approximate planning for factored POMDPs. In *Proceedings of the Sixth European Conference on Planning*, Toledo, Spain, 2001.

[37] Alan Fern, SungWook Yoon, and Robert Givan. Approximate policy iteration with a policy language bias. In *Advances in Neural Information Processing Systems*, Vancouver, BC, 2003.

[38] Robert Givan, Thomas Dean, and Matthew Greig. Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence*, 147:163–223, 2003.

[39] Carlos Guestrin, Daphne Koller, Chris Gearhart, and Neal Kanodia. Generalizing plans to new environments in relational MDPs. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 1003–1010, Acapulco, Mexico, 2003.

[40] Carlos Guestrin, Daphne Koller, and Ronald Parr. Max-norm projections for factored MDPs. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 673–680, Seattle, WA, 2001.

[41] Carlos Guestrin, Daphne Koller, and Ronald Parr. Multiagent planning with factored MDPs. In *Advances in Neural Information Processing Systems*, pages 1523–1530, Vancouver, BC, 2001.

[42] Carlos Guestrin, Daphne Koller, and Ronald Parr. Solving factored POMDPs with linear value functions. In *IJCAI-01 Workshop on Planning under Uncertainty and Incomplete Information*, Seattle, WA, 2001.

[43] Carlos Guestrin, Shobha Venkataraman, and Daphne Koller. Context specific multiagent coordination and planning with factored MDPs. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, pages 253–259, Edmonton, Alberta, 2002.

[44] Eric A. Hansen. An improved policy iteration algorithm for partially observable MDPs. In *Advances in Neural Information Processing Systems*, pages 1015–1021, Denver, CO, 1997.

[45] Eric A. Hansen. Solving POMDPs by searching in policy space. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 211–219, Madison, Wisconsin, 1998.

[46] Eric A. Hansen and Zhengzhu Feng. Dynamic programming for POMDPs using a factored state representation. In *Proceedings of the Fifth International Conference on AI Planning Systems*, pages 130–139, Breckenridge, CO, 2000.

[47] Milos Hauskrecht. Incremental methods for computing bounds in partially observable Markov decision processes. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 734–739, Providence, 1997.

[48] Milos Hauskrecht. Value-function approximations for partially observable Markov decision processes. *Journal of Artificial Intelligence Research*, 13:33–94, 2000.

[49] Jesse Hoey, Robert St-Aubin, Alan Hu, and Craig Boutilier. SPUDD: Stochastic planning using decision diagrams. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 279–288, Stockholm, 1999.

[50] Ronald A. Howard and James E. Matheson. *Readings on the Principles and Applications of Decision Analysis.* Strategic Decision Group, Menlo Park, CA, 1984.

[51] Nathanael Hyafil and Fahiem Bacchus. Conformant probabilistic planning via CSPs. In *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling*, pages 205–214, Trento, Italy, 2003.

[52] Nathanael Hyafil and Fahiem Bacchus. Utilizing structured representations and CSPs in conformant probabilistic planning. In *Proceedings of the European Conference on Artificial Intelligence*, pages 1033–1034, Valencia, Spain, 2004.

[53] Tommi Jaakkola, Satinder P. Singh, and Michael I. Jordan. Reinforcement learning algorithm for partially observable Markov decision problems. In *Advances in Neural Information Processing Systems*, volume 7, pages 345–352, 1995.

[54] Michael James and Satinder Singh. Learning and discovery of predictive state representations in dynamical systems with reset. In *Proceedings of the Twenty First International Conference on Machine Learning*, Banff, Alberta, 2004.

[55] Leslie Pack Kaelbling, Michael Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

[56] Sham Kakade. A natural policy gradient. In *Advances in Neural Information Processing Systems*, pages 1531–1538, Vancouver, BC, 2001.

[57] J. Kakalik. Optimum policies for partially observable Markov systems. Technical Report 18, Operations Research Center, MIT, 1965.

[58] Keiji Kanazawa, Daphne Koller, and Stuart Russell. Stochastic simulation algorithms for dynamic probabilistic networks. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 346–351, Montreal, 1995.

[59] Ralph L. Keeney and Howard Raiffa. *Decisions with Multiple Objectives.* John Wiley and Sons, New York, 1976.

[60] Kee-Eung Kim. *Representations and algorithms for large stochastic planning problems.* PhD thesis, Brown University, Rhode Island, 2001.

[61] Kee-Eung Kim and Tom Dean. Solving factored MDPs with large action spaces using algebraic decision diagrams. In *Proceedings of the 7th Pacific Rim International Conference on Artificial Intelligence*, pages 80–89, Tokyo, Japan, 2002.

[62] Daphne Koller and Ronald Parr. Computing factored value functions for policies in structured MDPs. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 1332–1339, Stockholm, 1999.

[63] Daphne Koller and Ronald Parr. Policy iteration for factored MDPs. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 326–334, Stanford, 2000.

[64] Nicholas Kushmerick, Steve Hanks, and Daniel Weld. An algorithm for probabilistic least-commitment planning. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 1073–1078, Seattle, 1994.

[65] Nicholas Kushmerick, Steve Hanks, and Daniel Weld. An algorithm for probabilistic planning. *Artificial Intelligence*, 76(1-2):239–286, 1995.

[66] Long-Ji Lin and Tom M. Mitchell. Memory approaches to reinforcement learning in non-Markovian domains. Technical Report CMU-CS-92-138, Carnegie Mellon University, 1992.

[67] Michael Littman, Anthony Cassandra, and Leslie Kaelbling. Learning policies for partially observable environments: Scaling up. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 362–370, San Francisco, CA, 1995.

[68] Michael L. Littman. Memoryless policies: Theoretical limitations and practical results. In Dave Cliff, Philip Husbands, Jean-Arcady Meyer, and Stewart W. Wilson, editors, *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, pages 238–245, Cambridge, MA, 1994. The MIT Press.

[69] Michael L. Littman. *Algorithms for Sequential Decision Making.* PhD thesis, Brown university, Rhode Island, 1996.

[70] Michael L. Littman, Anthony R. Cassandra, and Leslie Pack Kaelbling. Efficient dynamic-programming updates in partially observable Markov decision processes. Technical Report CS-95-19, Brown University, 1995.

[71] Michael L. Littman, Richard S. Sutton, and Satinder Singh. Predictive representations of state. In *Advances in Neural Information Processing Systems*, pages 1555–1561, Vancouver, BC, 2001.

[72] John Loch and Satinder Singh. Using eligibility traces to find the best memoryless policy in partially observable Markov decision processes. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 323–331, Madison, 1998.

[73] William S. Lovejoy. Computationally feasible bounds for partially observed Markov decision processes. *Operations Research*, 39:162–175, 1991.

[74] William S. Lovejoy. A survey of algorithmic methods for partially observed Markov decision processes. *Annals of Operations Research*, 28:47–66, 1991.

[75] Christopher Lusena, Judy Goldsmith, and Martin Mundhenk. Nonapproximability results for partially observable Markov decision processes. *Journal of Artificial Intelligence Research*, 14:83–103, 2001.

[76] Omid Madani, Steve Hanks, and Anne Condon. On the undecidability of probabilistic planning and infinite-horizon partially observable Markov decision problems. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, pages 541–548, Orlando, FL, 1999.

[77] Stephen Majercik and Michael Littman. Contingent planning under uncertainty via stochastic satisfiability. *Artificial Intelligence*, 147:119–162, 2003.

[78] Stephen M. Majercik and Michael L. Littman. MAXPLAN: A new approach to probabilistic planning. In *Proceedings of the Fourth International Conference on AI Planning Systems*, pages 86–93, 1998.

[79] Stephen M. Majercik and Michael L. Littman. Contingent planning under uncertainty via stochastic satisfiability. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, pages 549–556, Orlando, FL, 1999.

[80] Peter Marbach, Oliver Mihatsch, and John N. Tsitsiklis. Call admission control and routing in integrated service networks using neuro-dynamic programming. *IEEE Journal on Selected Areas in Communications*, 18(2):197–208, 2000.

[81] J.J. Martin. *Bayesian decision problems and Markov chains.* John Wiley & Sons, New York, 1967.

[82] David McAllester and Satinder Singh. Approximate planning for factored POMDPs using belief state simplification. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 409–416, Stockholm, 1999.

[83] R. Andrew McCallum. Overcoming incomplete perception with utile distinction memory. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 190–196, Amherst, MA, 1993.

[84] R. Andrew McCallum. Instance-based utile distinctions for reinforcement learning with hidden state. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 387–395, Lake Tahoe, CA, 1995.

[85] R. Andrew McCallum. Hidden state and reinforcement learning with instance-based state identification. *IEEE Transations on Systems, Man, and Cybernetics*, 26(3):464–473, 1996.

[86] J. McCarthy and P.J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1965.

[87] Nicolas Meuleau, Milos Hauskrecht, Kee-Eung Kim, Leonid Peshkin, Leslie Pack Kaelbling, Thomas Dean, and Craig Boutilier. Solving very large weakly coupled Markov decision processes. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 165–172, Madison, WI, 1998.

[88] Nicolas Meuleau, Kee-Eung Kim, Leslie Pack Kaelbling, and Anthony R. Cassandra. Solving POMDPs by searching the space of finite policies. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 417–426, Stockholm, 1999.

[89] Nicolas Meuleau, Leonid Peshkin, Kee-Eung Kim, and Leslie Pack Kaelbling. Learning finite-state controllers for partially observable environments. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 427–436, Stockholm, 1999.

[90] George E. Monahan. A survey of partially observable Markov decision processes: Theory, models and algorithms. *Management Science*, 28:1–16, 1982.

[91] Michael Montemerlo, Joelle Pineau, Nick Roy, Sebastian Thrun, and Vandi Verma. Experiences with a mobile robotic guide for the elderly. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, pages 587–592, Edmonton, Alberta, 2002.

[92] Andrew Ng, H. Jin Kim, Michael Jordan, and Shankar Sastry. Autonomous helicopter flight via reinforcement learning. In *Advances in Neural Information Processing Systems*, Vancouver, BC, 2003.

[93] Andrew Ng, Ronald Parr, and Daphne Koller. Policy search via density estimation. In *Advances in Neural Information Processing Systems*, pages 1022–1028, Denver, CO, 2000.

[94] Andrew Y. Ng and Michael Jordan. PEGASUS: A policy search method for large MDPs and POMDPs. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 406–415, Stanford, CA, 2000.

[95] Brenda Ng, Leonid Peshkin, and Avi Pfeffer. Factored particles for scalable monitoring. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, pages 370–377, Emonton, Alberta, 2002.

[96] N. Onder and M.E. Pollack. Contingency selection in plan generation. In *Proceedings of the Fourth European Conference on Planning: Recent Advances in AI Planning*, pages 364–376, 1997.

[97] N. Onder and M.E. Pollack. Conditional, probabilistic planning: A unifying algorithm and effective search control mechanisms. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, pages 577–584, 1999.

[98] Tim Paek and Eric Horvitz. Conversation as action under uncertainty. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 455–464, Stanford, CA, 2000.

[99] C. H. Papadimitriou and J. N. Tsitsiklis. The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.

[100] Ronald Parr and Stuart Russell. Approximating optimal policies for partially observable stochastic domains. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1088–1094, Montreal, 1995.

[101] Relu Patrascu, Pascal Poupart, Dale Schuurmans, Craig Boutilier, and Carlos Guestrin. Greedy linear value-approximation for factored Markov decision processes. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, pages 285–291, Edmonton, AB, 2002.

[102] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, 1988.

[103] M.A. Peot. *Decision-theoretic planning.* PhD thesis, Department of Engineering-Economic Systems and Operations Research, Stanford University, Palo Alto, 1998.

[104] Leonid Peshkin, Nicolas Meuleau, and Leslie P. Kaelbling. Learning policies with external memory. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 307–314, San Francisco, CA, 1999.

[105] Joelle Pineau. *Tractable Planning Under Uncertainty: Exploiting Structure.* PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburg, 2004.

[106] Joelle Pineau, Geoff Gordon, and Sebastian Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 1025–1032, Acapulco, Mexico, 2003.

[107] David Poole. Probabilistic partial evaluation: Exploiting rule structure in probabilistic inference. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 1284–1291, Nagoya, Japan, 1997.

[108] David Poole. Decision theory, the situation calculus and conditional plans. *Electronic Transactions on Artificial Intelligence*, pages 105–158, 1998.

[109] K.-M. Poon. A fast heuristic algorithm for decision-theoretic planning. Master's thesis, The Honk-Kong University of Science and Technology, 2001.

[110] Pascal Poupart and Craig Boutilier. Value-directed belief state approximation for POMDPs. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 497–506, Stanford, 2000.

[111] Pascal Poupart and Craig Boutilier. Vector-space analysis of belief-state approximation for POMDPs. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, pages 445–452, Seattle, WA, 2001.

[112] Pascal Poupart and Craig Boutilier. Bounded finite state controllers. In *Advances in Neural Information Processing Systems*, Vancouver, BC, 2003.

[113] Pascal Poupart, Craig Boutilier, Relu Patrascu, and Dale Schuurmans. Piecewise linear value function approximation for factored MDPs. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, pages 292–299, Edmonton, AB, 2002.

[114] Pascal Poupart, Luis E. Ortiz, and Craig Boutilier. Value-directed sampling methods for monitoring POMDPs. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, pages 453–461, Seattle, WA, 2001.

[115] Bob Price and Craig Boutilier. A Bayesian approach to imitation in reinforcement learning. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 712–720, Acapulco, Mexico, 2003.

[116] M. L. Puterman. *Markov Decision Problems.* Wiley, New York, 1994.

[117] Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems.* MIT Press, 2001.

[118] Benjamin Van Roy. *Learning and value function approximation in complex decision processes*. PhD thesis, MIT, EECS, 1998.

[119] Nick Roy and Geoffrey Gordon. Exponential family PCA for belief compression in POMDPs. In *Advances in Neural Information Processing Systems*, pages 1635–1642, Vancouver, BC, 2002.

[120] Matthew Rudary and Satinder Singh. A nonlinear predictive state representation. In *Advances in Neural Information Processing Systems*, Vancouver, BC, 2003.

[121] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. PWS, Boston, 1996.

[122] Brian Sallans. Learning factored representations for partially observable Markov decision processes. In *Advances in Neural Information Processing Systems*, pages 1050–1056, Denver, CO, 2000.

[123] Brian Sallans and Goeff Hinton. Using free energies to represent Q-values in multiagent reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1075–1081, Vancouver, BC, 2001.

[124] Dale Schuurmans and Relu Patrascu. Direct value-approximation for factored MDPs. In *Advances in Neural Information Processing Systems*, pages 1579–1586, Vancouver, BC, 2001.

[125] Ross D. Shachter. Evaluating influence diagrams. *Operations Research*, 34:871–882, 1986.

[126] Ross D. Shachter. Probabilistic inference and influence diagrams. *Operations Research*, 36:589–605, 1988.

[127] Ross D. Shachter and Mark A. Peot. Decision making using probabilistic inference methods. In *Proceedings of the Eighth Conference on Uncertainty in Artificial Intelligence*, pages 276–283, Stanford, CA, 1992.

[128] Satinder Singh, Michael James, and Mathew Rudary. Predictive state representations: A new theory for modeling dynamical systems. In *Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence*, pages 512–519, Banff, Alberta, 2004.

[129] Satinder Singh, Michael Littman, Nicholas Jong, David Pardoe, and Peter Stone. Learning predictive state representations. In *Proceedings of the Twentieth International Conference on Machine Learning*, pages 712–719, 2003.

[130] Satinder P. Singh, Tommi Jaakkola, and Michael I. Jordan. Learning without state-estimation in partially observable Markovian decision processes. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 284–292, 1994.

[131] Satinder P. Singh, Tommi Jaakkola, and Michael I. Jordan. Reinforcement learning with soft state aggregation. In *Advances in Neural Information Processing Systems*, pages 361–368, 1995.

[132] Richard D. Smallwood and Edward J. Sondik. The optimal control of partially observable Markov processes over a finite horizon. *Operations Research*, 21:1071–1088, 1973.

[133] Trey Smith and Reid Simmons. Heuristic search value-iteration for POMDPs. In *Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence*, pages 520–527, Banff, Alberta, 2004.

[134] Edward J. Sondik. *The optimal control of partially observable Markov Decision Processes*. PhD thesis, Stanford University, Palo Alto, 1971.

[135] Edward J. Sondik. The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs. *Operations Research*, 26:282–304, 1978.

[136] Matthijs T. J. Spaan and Nikos Vlassis. Perseus: Randomized point-based value iteration for POMDPs. Technical Report IAS-UVA-04-02, Informatics Institute, University of Amsterdam, 2004.

[137] Matthijs T. J. Spaan and Nikos Vlassis. A point-based POMDP algorithm for robot planning. In *IEEE International Conference on Robotics and Automation*, pages 2399–2404, New Orleans, 2004.

[138] Robert St-Aubin, Jesse Hoey, and Craig Boutilier. APRICODD: Approximate policy construction using decision diagrams. In *Advances in Neural Information Processing Systems*, pages 1089–1095, Denver, 2000.

[139] Joseph A. Tatman and Ross D. Shachter. Dynamic programming and influence diagrams. *IEEE Transations on Systems, Man, and Cybernetics*, 20(2):365–379, 1990.

[140] Sebastian Thrun. Monte Carlo POMDPs. In *Advances in Neural Information Processing Systems*, pages 1064–1070, Denver, 1999.

[141] Naftali Tishby, Fernando C. Pereira, and William Bialek. The information bottleneck method. In *37th annual Allerton conference on communication, control and computing*, pages 368–377, 1999.

[142] Nikos Vlassis and Matthijs T. J. Spaan. A fast point-based algorithm for POMDPs. In *Proceedings of the Annual Machine Learning Conference of Belgium and the Netherlands*, pages 170–176, Brussels, Belgium, 2004.

[143] Chelsea C. White. A survey of solution techniques for the partially observed Markov decision process. *Annals of Operations Research*, 32:215–230, 1991.

[144] Jason Williams. A probabilistic model of human/computer dialogue with application to a partially observable Markov decision process. PhD first year report. Department of Engineering, University of Cambridge., August 2003.

[145] Bo Zhang, Qingsheng Cai, Jianfeng Mao, and Baining Guo. Planning and acting under uncertainty: A new model for spoken dialogue systems. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, pages 572–579, Seattle, WA, 2001.

[146] Nevin L. Zhang and Wenju Liu. Planning in stochastic domains: Problem characteristics and approximation. Technical Report HKUST-CS96-31, Hong Kong University of Science and Technology, 1996.

[147] Nevin L. Zhang and Weihong Zhang. Speeding up the convergence of value-iteration in partially observable Markov decision processes. *Journal of Artificial Intelligence Research*, 14:29–51, 2001.

[148] Rong Zhou and Eric A. Hansen. An improved grid-based approximation algorithm for POMDPs. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 707–714, Seattle, 2001.