Approximate Value-Directed Belief State Monitoring for Partially Observable Markov Decision Processes

by

Pascal Poupart

B.Sc., McGill University, 1998

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES

(Department of Computer Science)

We accept this thesis as conforming
to the required standard

The University of British Columbia

November 2000

Abstract

Partially observable Markov decision processes (POMDPs) provide a principled approach to planning under uncertainty. Unfortunately, several sources of intractability currently limit the application of POMDPs to simple problems. The following thesis is concerned with one source of intractability in particular, namely the belief state monitoring task. As an agent executes a plan, it must track the state of the world by updating its beliefs with respect to the current state. Then, based on its current beliefs, the agent can look up the next action to execute in its plan. In many situations, an agent may be required to decide in real-time which action to execute next. Thus, efficient algorithms to update the current belief state would be desirable. Unfortunately, exact belief state monitoring turns out to be very time consuming for many domains.

This thesis introduces a value-directed framework to analyze and design approximation methods that speed up the monitoring task. The goal of approximate belief state monitoring is to trade monitoring accuracy for efficiency. Thus, this framework outlines a principled approach to quantify the impact of approximating belief states on the original plan. Since at any point in time, an action is executed based on the current belief state, it may be possible that a less desirable action ends up being executed as a result of the approximations used to infer the current belief state.

The framework developped covers a wide range of approximation methods including projection schemes and density trees. First, several bounds on the loss in decision quality due to approximate belief state monitoring are derived. Then, given a class of approximation methods, a few search algorithms are proposed to seek a relatively good approximation scheme within the given class. These algorithms essentially try to minimize the bounds derived. Next, a vector space analysis is performed to gain some insights regarding which properties of approximation methods are likely to ensure a minimal impact on decision quality. Finally, faster algorithms (than the previous ones) are designed to search for approximation methods that exhibit such properties.

Contents

List of Tables

List of Figures

Acknowledgements

I would like to acknowledge the invaluable contribution of many people who have helped me directly or indirectly in the accomplishment of this work. First and foremost, my supervisor, Craig Boutilier, has played a key role in guiding my first steps in the world of research. Without his support and continuous advice, this work would not have been possible. Craig has been a mentor as well as a friend and his generous support is greatly appreciated.

I am also indepted to David Poole and Bob Price for their constructive comments which contributed to the final form of this thesis. I would also like to thank Bob for the many interesting and helpful discussions on various topics related to this work.

I am grateful to the Laboratory for Computational Intelligence at the University of British Columbia for providing me with a stimulating environment to learn and carry out research. I also thank the department of computer science at the University of Toronto for their hospitality while visiting in the final year of my Master's degree.

Finally, my special thanks go my parents who have provided me with a warm and stable environment in the early stages of my life. This work is in part the fruit of their love and their care which have contributed to shape the man I am today.

Pascal Poupart

The University of British Columbia
November 2000

Chapter 1

Introduction

1.1 Problem statement

Partially observable Markov decision processes (POMDPs) have received considerable attention as a framework for decision theoretic planning. Their generality allows one to model uncertainty in action effects, sensor observations and state of knowledge. They also allow the practitioner to find an optimal plan (that takes into account this uncertainty) with respect to one or several concurrent objectives. Despite their attractiveness as a conceptual model, POMDPs have limited applicability as they are intractable:

- The *offline* search for a solution (optimal plan) to a POMDP is often intractable [28, 30].
- The *online* execution of an optimal plan often requires belief state monitoring which can be itself intractable.

This thesis focuses on the online problem, namely *belief state monitoring.* In this work, it is assumed that a policy (plan) is represented by a mapping from belief states (probability distributions over the states of the world) to actions. When that is the case, the agent must update its current belief state at each time step in order to know which action to execute next. Unfortunately, the belief state monitoring task tends to be a computationally intensive process which is unacceptable in practice (especially for real-time applications). In short, a belief update requires the computation of a probability distribution over the whole state space. In most domains, the state space is large so this computation is prohibitively expensive.

One of the main reasons why state spaces for certain problems are large is because the states are defined by a large collection of features or *random variables*. For such problems, each state corresponds to a joint instantiation of those variables and, consequently, the number of states is *exponential* in the number of variables. This is a serious problem since with only a few dozens of variables the state space grows larger than the internal memory of most of today's personal computers. Therefore, a very common approach to perform belief monitoring is to try to reason at the variable level instead of the state level. Dynamic Bayesian Networks (DBNs) provide a way to do this by exploiting variable independencies. Unfortunately, as Boyen and Koller [3] have shown, most if not all the variables tend to become correlated over time for non-trivial problems and therefore belief inference remains intractable.

Due to the lack of effective exact inference methods to tackle large state space problems, several researchers have investigated approximate methods. These include:

- Sampling [37, 20]
- Density Trees [26]
- Projection [3]
- Variational methods [21]

These methods trade monitoring accuracy for monitoring efficiency. When an agent's estimate of the current belief state is inexact, it is possible that it will execute a different action than the one indicated by the exact belief state. As time passes, if the belief state is repeatedly approximated, then the sequence of actions executed may be completely different than the one prescribed by the policy for the true underlying state. Hence, an important question for approximate monitoring methods is: how do approximations influence decision quality?

In the literature, when an analysis of the approximation quality is performed, it is usually done by measuring some distance between the exact and approximate belief states instead of evaluating the expected total loss. This is because general dynamical systems free of any decision process are usually considered and the absence of a decision process gives some freedom as for which metric to use when evaluating the error between the approximate and exact belief states. Common distance measures that have been used include KL-divergence (a measure of entropy), L_1, L_2 and L_∞. Although they are well-known distance measures with useful theoretical properties, they do not translate easily in a meaningful measure of the loss in performance for an agent. In the presence of a decision process (i.e., POMDPs), the decision maker is primarily interested in how much expected utility is lost due to approximations. Since the expected utility of a POMDP policy is given by its corresponding value function, I propose in the following chapters a framework to carry out a *value-directed* analysis of approximation methods.

1.2 Illustrative Example

Let's illustrate the importance of a value-directed analysis with a simple factory problem. Imagine a process in which two parts are stamped from the same machine A. Each part may be faulty with a certain probability. Since the parts are stamped by the same machine, their fault probabilities are correlated. If the parts are faulty, subsequent processing on some machine B may induce a monetary loss to the factory. Therefore, a controller gets to decide whether to reject the parts or to continue their processing.

In the event where the parts are processed individually by machine B, the decision to reject or process each part is independent and the correlation between their fault probability can be ignored safely.[1] One could devise an approximation method whereby the approximate belief state is the same as the exact belief state except that the correlation between fault probabilities is ignored. If the correlation is strong, the distance between the exact and approximate belief states may be great (from a KL-divergence, L_1, L_2 or L_∞ point of view), but the factory won't suffer any loss.

On the other hand, if the parts are processed jointly by machine B, the decision to reject or process the parts is influenced by the correlation between their fault probability. If the consequences for processing faulty parts are disastrous (e.g., explosion), then it may not be wise to ignore a weak correlation even if the distance measured by KL-divergence, L_1, L_2 or L_∞ is small.

1.3 Contributions

The main contribution of this thesis consists of a theoretical framework to analyze value-directed belief state approximation in POMDPs. The framework provides a novel view of approximation methods and their impact on decision quality. A general technique is proposed to derive bounds on the loss in expected return associated with a given approximation method. The key to those error bounds is the use of the value function as a distance measure. The bounds computed are then used to search for a good approximation scheme that minimizes the loss in expected utility. Algorithms to compute such bounds and to search within a class of approximation methods are proposed for projection schemes and density trees. Those algorithms are not restricted to projection schemes or density trees as they can be used for any "linear" approximation method.

Unfortunately, these algorithms are computationally intensive: they require, in the worst case, a quadratic increase in the solution time to solve a POMDP. As

[1] The decisions are independent assuming that the result of the decision for one part cannot be observed before the decision for the other part is made.

mentioned earlier, solving POMDPs is already intractable, so the applicability of those algorithms is limited to trivial problems. The running time can be considerably improved by introducing a vector space formulation of the approximation problem. This formulation allows us to search for a good approximation method in a time comparable to that of solving POMDPs. This makes the value-directed framework practical for problems that we are currently able to solve using state of the art solution algorithms. The price paid for this efficiency consists of looser bounds and consequently, the approximation schemes returned by those efficient search algorithms have lower performance guarantees. On the other hand, experiments suggests that in practice the expected loss in utility remains roughly the same on average.

Another aspect of the vector space formulation is the novel view it provides for understanding how some approximation methods alter policies. An important observation shows that the difference in utility for alternative courses of action varies more in some directions of the belief space than others. If we consider belief states as points in belief space, we can characterize an approximation by the *direction* of the vector corresponding to the difference between the exact and approximate belief points. Intuitively, approximations with directions where the difference in utility for executing alternative courses of action varies slowly are less likely to impact decisions and therefore are more accurate. The vector space formulation enables us to identify very quickly approximation schemes with such directions of higher accuracy. Finally, this analysis of approximation direction explains in part why KL-divergence, L_1, L_2 and L_∞ do not translate well in a measure of loss in expected total return. Roughly speaking, they are *distance* measures and they do not differentiate between equally distant approximations that have different *directions*.

1.4 Outline

The core of this work resides in Chapters 3 and 4 where some algorithms to compute error bounds and to search for good approximation schemes are presented. The framework is introduced in Chapter 3 and the vector space formulation is presented in Chapter 4. An outline of each chapter follows.

Chapter 2 presents a review of the POMDP model and introduces the notation used throughout the thesis. The main approaches (value iteration and policy iteration) used to solve exactly POMDPs are summarized and the sources of intractability that prevent them from being used effectively are described. Next, dynamic Bayesian networks (DBNs) are introduced as a popular method for belief state monitoring and some explanations are provided concerning the complexity of belief inference using them. Among the different approximation methods for belief state estimation, projection schemes and density trees are detailed as they will be the focus of this framework. Projection schemes can be easily integrated to DBNs for a significant speed up whereas density trees simplify greatly the error bound analysis. Finally the assumptions made throughout this thesis are stated.

Chapter 3 introduces the value-directed approximation (VDA) framework for belief state monitoring. There are two components to this framework: the first computes a bound on the loss in expected return for a given POMDP and a given approximation method, and the second describes greedy search procedures to find a good approximation method within a certain class of methods (projection schemes or density trees) for a given POMDP. Although the algorithms to compute the bounds and to conduct the searches are executed offline, they are computationally intensive. In fact their complexity is similar to, but worse than, the algorithms to solve POMDPs (value iteration or policy iteration) which are known to be intractable.

To that effect, Chapter 4 proposes a more efficient approach based on a vector space formulation. Essentially, the running time of those algorithms is improved by loosening the error bounds. Moreover, the vector space formulation provides a different viewpoint for analysing partial belief state monitoring. The emphasis is on the *direction* of an approximation and how its knowledge can be exploited efficiently to estimate the loss in expected utility.

Chapter 5 describes some empirical results obtained by applying the methods developed in this framework. First, the factory example introduced in Section 1.2 is revisited with specific transition probabilities and reward functions. As a proof of concept, it is shown how for a specific prior, the choice of a projection scheme that minimizes KL-divergence, L_1 or L_2 norms yields a different choice than that prescribed by a value-directed approach. Following, several experiments are carried out on three test problems. They evaluate and compare the running time of the search algorithms, the quality of the error bounds and the average expected loss incurred in practice.

Finally, Chapter 6 summarizes the contributions of this thesis and elaborates on future extensions to the framework. In particular, one goal of the author is to extend this work to value-directed sampling since sampling is one of the most popular and effective method for tracking dynamical systems. Another possibility consists in integrating this framework for value-directed approximations with the algorithms to solve POMDPs. Ultimately, when solving POMDPs, it would be nice to take into account the fact that monitoring will be approximate. This may lead to ways to solve POMDPs in a *bounded-optimal* fashion.

Chapter 2

Partially Observable Markov Decision Processes (POMDPs)

The POMDP concept was first introduced in the control theory and operations research communities [9, 1, 35, 31, 6, 29] 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 do planning under uncertainty [4, 22]. Compared to other methods, POMDPs have the advantage of a well-founded theory since they can be viewed as a special (continuous) case of the well-known Markov decision processes (MDPs) which are rooted in probability theory, decision theory and utility theory.

In this chapter, an overview of the POMDP framework is presented with an emphasis on the concepts that will recur throughout the thesis. For instance, the reader should pay attention to the notion of a *conditional plan* and its value function representation as an *α-vector*. These will be instrumental in defining the notion of *plan switching* when analyzing how the course of action gets modified by an approximation. The reader should also pay special attention to the *incremental pruning* algorithm. This will ease the understanding of the algorithms to bound error and to search for good approximation schemes presented in later chapters.

Finally, we describe *dynamic Bayesian networks* for exact belief inference, as well as *projection schemes* and *density trees* for approximate inference, as they will serve to illustrate the value-directed framework.

This chapter also provides an overview of the sources of intractability when solving a POMDP. Although these are not the focus of this work, they limit the type of experiments that can be carried out when testing the value-directed framework. On the other hand, the research community has made some progress on several fronts and I have tried to incorporate some of the techniques developed to speed up the solution of POMDPs. For instance, *factored representations* allow us to solve some POMDPs with significantly larger state spaces than before [2, 18, 15]. They also allow us to improve the running time of the algorithms developed in this thesis. As mentioned earlier, the bounding algorithms and the search algorithms are similar to incremental pruning and therefore can benefit from the same type of speed up.

Lastly, this chapter addresses an important question regarding when and why belief state monitoring is required. In the literature, there are two common representations for POMDP policies: as a *mapping from belief states to actions*; and as a *finite state controller.* The latter representation doesn't require belief state monitoring at execution time. Thus, one might seek an algorithm (such as Hansen's policy iteration [13, 14]) that outputs a finite state automaton to circumvent the belief state monitoring problem altogether. I argue that policy iteration is not always suitable for modeling and efficiency reasons, and even when it is suitable, it may still be desirable to monitor belief states for explanatory purposes.

2.1 Model Description

POMDPs provide a nice framework to model uncertainty in a planning problem. They allow action effects and state observations to be modeled probabilistically. Formally, a POMDP is described as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{Z}, T, Z, R, h, \gamma \rangle$ where:

- $\mathcal{S}$ is the *set of states* of the world.

- $\mathcal{A}$ is the *set of actions* the agent can execute.

- $\mathcal{Z}$ is the *set of observations* the agent can experience.

- $T : \mathcal{S} \times \mathcal{A} \rightarrow \Delta(\mathcal{S})$ is the *state transition function* describing the dynamics of the world. $\Delta(\mathcal{S})$ is the set of all probability distributions over the set $\mathcal{S}$ of states. We write $T(s, a, s')$ to denote the probability of a transition to state s' given that it was in state s at the previous stage and that the agent executed action a.

- $Z : \mathcal{A} \times \mathcal{S} \rightarrow \Delta(\mathcal{Z})$ is the *observation function.* $\Delta(\mathcal{Z})$ is the set of all probability distributions over the set $\mathcal{Z}$ of observations. We write $Z(a, s', z')$ to denote the probability with which the agent may experience observation z' given that it just executed action a and the world made a transition to state s'.

- $R : \mathcal{S} \times \mathcal{A} \rightarrow \Re$ is the *reward function.* We write $R(s, a)$ to denote the reward received by the agent when the world is in state s and the agent executes action a.

- h is the planning *horizon.*

- γ is the *discount factor.*

2.1.1 State Space $\mathcal{S}$

The world is modeled by a set $\mathcal{S}$ of distinct states. In this thesis, $\mathcal{S}$ is assumed to be finite for computational purposes, but in general, the POMDP framework allows countably many states [33]. At any point in time, the relevant features of the world are summarized by a state $s \in \mathcal{S}$. More precisely, the i^{th} feature is represented by variable X_i which can be instantiated to any value in the (finite) domain $\mathcal{D}_i$ of X_i. Each state s corresponds to a joint instantiation of all the variables $X_1, X_2, \ldots, X_n$

encoding the world's relevant features. Assuming a total of n variables, the number of states in $\mathcal{S}$ is then exponential in n since $\mathcal{S} = \mathcal{D}_1 \times \mathcal{D}_2 \times \ldots \times \mathcal{D}_n$. Note that this exponential relation between variables and states will be responsible for the intractable size of α-vectors (Section 2.4) and the intractability of belief state monitoring (Section 2.5).

2.1.2 Action space $\mathcal{A}$

An agent living in this world seeks to influence its state by executing actions from the set $\mathcal{A}$ ($\mathcal{A}$ is also assumed to be finite for computational reasons). Roughly speaking, the agent's goal is to choose actions that will influence the world in a way that desirable states are visited frequently. On the other hand, as opposed to classical planning, where actions are assumed to be deterministic, POMDPs allow uncertainty to be modeled for action effects. 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.

2.1.3 Transition function T

The stochastic nature of action effects is captured by the transition function T. Time is discretized in consecutive steps called time steps at which the agent gets to execute an action in $\mathcal{A}$. Here it is assumed that any action in $\mathcal{A}$ can be attempted in any state of the world and at any time step. After each action, the world makes a transition to a new state, so that over time, the world goes through a sequence of states $(s_0, s_1, \ldots, s_t, \ldots)$ where t denotes the time step index. Assuming the transition function is the same at every time step (i.e., the transition function is *stationary*), then $T(s, a, s') = Pr(s'|s, a)$ denotes the probability that the world makes a transition to state s' when action a is executed in state s. This transition function also assumes the Markov property, namely that the next state s_{t+1} of the world depends only on the current state s_t and the current action a_t, and is independent of past history (past states and past actions).

$$Pr(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_0, a_0) = Pr(s_{t+1}|s_t, a) \tag{2.1}$$

2.1.4 Observation space $\mathcal{Z}$

After executing an action, the agent makes an observation $z \in \mathcal{Z}$. This observation provides some information as to what state the world just made a transition to. In fully observable MDPs, $\mathcal{Z} = \mathcal{S}$ and the agent knows exactly the current state. In partially observable MDPs, which is the focus of this thesis, $\mathcal{Z}$ differs from $\mathcal{S}$. Observations provide only partial information to the agent since the same observation may be experienced in several states. Therefore, the agent remains uncertain about the current state of the world and it is said to have only *beliefs* about the current state. Those beliefs are usually represented by a probability distribution b over the set $\mathcal{S}$ of world states. We write $b(s)$ to denote the probability of s when the agent has beliefs b.

2.1.5 Observation function Z

Partial observability is modeled by the observation function Z. Assuming the observation function is the same at every time step (i.e., it is stationary), then $Z(a, s', z') = Pr(z'|a, s')$ denotes the probability that the agent experiences observation z' after executing action a and making a transition to state s'. The transition and observation functions can be used to update the current belief state at each time step. That is, using Bayes rule, one can infer b_{t+1} from b_t, a_t and z_{t+1} as follows:

$$\begin{aligned} b_{t+1}(s') &= \frac{\sum_{s \in \mathcal{S}} b_t(s) Pr(s'|s, a_t) Pr(z_{t+1}|a_t, s')}{K} & (2.2) \\ &= \frac{\sum_{s \in \mathcal{S}} b_t(s) T(s, a_t, s') Z(a_t, s', z_{t+1})}{K} & (2.3) \end{aligned}$$

In the above equations, the denominator is the normalizing constant $K = \sum_{s'} \sum_s b_t(s) Pr(s'|s, a_t) Pr(z_{t+1}|a_t, s')$ that ensures that b is a probability distribution. If we think of b as a row vector, we can rewrite Equation 2.2 in matrix notation by transforming the transition and observation functions into linear operators. Define $M_{a,z}$ to be an $|\mathcal{S}| \times |\mathcal{S}|$ matrix such that $M_{a,z}(i,j) = T(s_i, a, s_j)Z(a, s_j, z)$ is the likelihood of the world to transition from state s_i to state s_j when action a is executed and observation z is experienced. We can further decompose the matrix $M_{a,z}$ into a product of matrices $M_a M_z$. M_a is a transition matrix such that $M_a(i,j) = T(s_i, a, s_j)$ and M_z is a diagonal observation matrix such that $M_z(i,i) = Z(a, s_i, z)$. In matrix notation, we can compute b_{t+1} from b_t, a_t and z_{t+1} as follows:

$$\begin{aligned} b_{t+1} &= \frac{b_t M_{a_t, z_{t+1}}}{K} && (2.4) \\ &= \frac{b_t M_{a_t} M_{z_{t+1}}}{K} && (2.5) \end{aligned}$$

2.1.6 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 classical planning 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.

2.1.7 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 between 0 and 1 indicating by how much a reward should be scaled down for every time step delay.[1] Finally, expression 2.6 shows the *expected total discounted reward* that an agent seeks to maximize for a given horizon h and a given discount factor γ. Expression 2.7 is the equivalent in matrix notation assuming R_a is a column vector such that $R_a(s) = R(s,a)$.

$$\begin{aligned}\text{Expected total discounted reward} &= \sum_{t=0}^{h} \gamma^t \sum_{s \in \mathcal{S}} b_t(s) R(s, a_t) && (2.6)\\ &= \sum_{t=0}^{h} \gamma^t b_t R_{a_t} && (2.7)\end{aligned}$$

2.2 Policies

2.2.1 Definition

Given a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{Z}, T, Z, R, h, \gamma \rangle$ specifying a POMDP, an important question is: what action should an agent execute at each time step to maximize its expected total return? In order to answer this question, we define Π to be the set of all *policies* π (action strategies) that an agent can execute. Roughly 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 the initial belief state b_0 and the history (sequence) of actions and observations experienced so far ($hist_t = \langle a_0, z_1, a_1, z_2, \ldots, a_{t-1}, z_t \rangle$). Hence, a policy π is a mapping from initial belief states and histories to actions.

For a given initial belief state, a 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. A conditional plan is intuitively defined as a mapping from histories to actions. The execution of a conditional plan essentially consists of the traversal of its corresponding tree from the root to a leaf by interleaving action execution and observation gathering. After traversing a node and executing its corresponding action a, the next action to execute is associated with the child node

[1] For computational purposes, the infinite-horizon case considered in this thesis always assumes a discount factor smaller than 1.

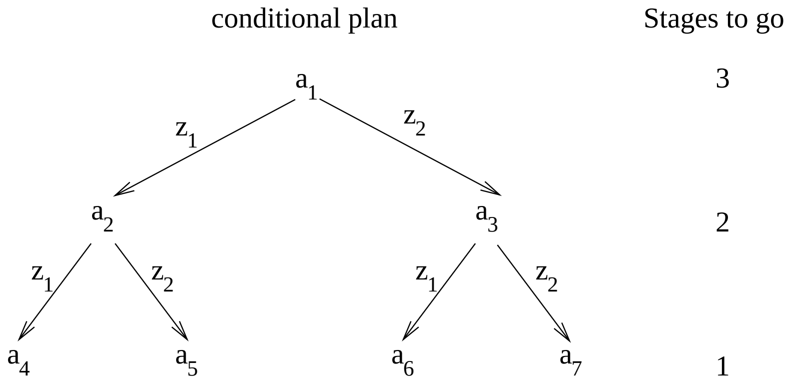


Figure 2.1: Tree representation of a three-step conditional plan

determined by the branch labelled with the next observation z experienced by the agent.

In general, we can define recursively k-step conditional plans 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 an action a followed by an observation strategy σ_k. In turn, an observation strategy $\sigma_k : \mathcal{Z} \rightarrow \Gamma_{k-1}$ 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 corresponding in Figure 2.1. Note that one-step conditional plans correspond to an action only (i.e., $\Gamma_1 = A$) since an observation strategy is unecessary.

Unfortunately, as the number of steps increases, so does the number and the length of histories, and it is infeasible to represent a mapping over all such histories. Alternatively, it is possible to summarize histories by belief states which have a fixed-size finite representation ($O(|\mathcal{S}|)$). A belief state is a sufficient statistic that encodes all the necessary information from previous actions and observations for planning purposes. For instance, belief state b_t summarizes the relevant information encoded by the initial belief state b_0 and the sequence of actions and observations up to time step t.

$$b_t \equiv \langle b_0, a_0, z_1, a_1, z_2, \ldots, a_{t-1}, z_t \rangle \tag{2.8}$$

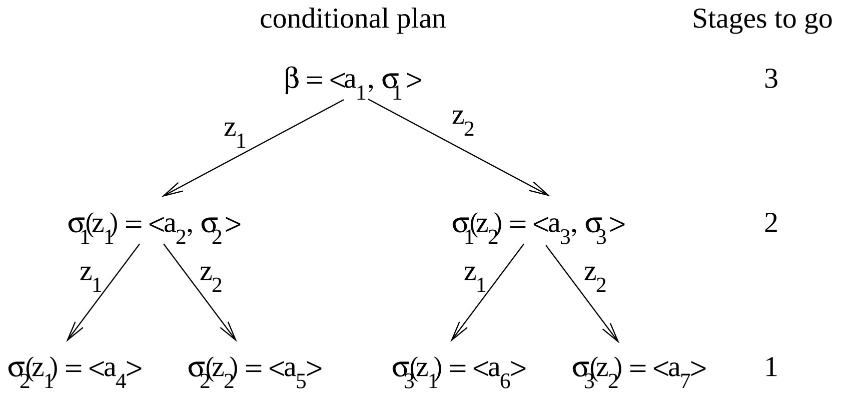


Figure 2.2: Recursive definition of a conditional plan

Intuitively, this follows from the fact that the transition function $T(b_{t-1}, a_{t-1}, b_t) = Pr(b_t|b_{t-1}, a_{t-1})$ depends only on the previous belief state b_{t-1} (due to the Markov property). Since $b_{t-1} \equiv \langle b_0, a_0, z_1, a_1, z_2, \ldots, a_{t-2}, z_{t-1}\rangle$, we can rewrite Expression 2.8 recursively as follows:

$$b_t \equiv \langle b_{t-1}, a_{t-1}, z_t\rangle \tag{2.9}$$

Expression 2.9 is not surprising since we had already derived Equation 2.4 which tells us how to update b_t from b_{t-1}, a_{t-1} and z_t. This update equation is rewritten below for convenience.

$$b_t = \frac{b_{t-1}M_{a_{t-1},z_t}}{K}$$

Thus we can also define a policy π as mappings $\pi_t(b)$ (one mapping for each time step t) from belief states to actions. When the mappings $\pi_t(b)$ are the same for all time steps, we say that the policy is stationary and we simply write $\pi(b)$. In practice, representing a policy as mappings from belief states to actions can be problematic because the belief space is an $(|\mathcal{S}| - 1)$-dimensional continuous space. Fortunately, a key result by Sondik [34] allows us to circumvent this problem. The explanations regarding Sondik's solution are deferred to the next section after value functions are introduced.

2.2.2 Evaluation

Given the set of all policies Π, we need a mechanism to evaluate and compare policies. Each policy π has a corresponding value function $V^{\pi} : \Delta(\mathcal{S}) \rightarrow \Re$ which is defined as a mapping from initial belief states to expected total reward. For a given initial belief state, the value returned is the expected sum of the discounted rewards earned at each time step as defined by Equation 2.10. Here $\pi_t(b_t)$ is the action a_t prescribed by policy π at time step t for belief state b_t.

$$V^{\pi}(b_0) = \sum_{t=0}^{h} \gamma^t b_t R_{\pi_t(b_t)} \tag{2.10}$$

We can use value functions to order policies. A decision theoretic agent prefers π to π' when $V^{\pi}(b) \geq V^{\pi'}(b)$ for all $b \in \Delta(\mathcal{S})$. 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 ($V^{\pi^*}(b) \geq V^{\pi}(b)\ \forall \pi, b$).

As with policies, representing a value function can be problematic because its domain is an $|\mathcal{S}|$-dimensional continuous space corresponding to the belief space. Fortunately, a key result of Sondik [34] 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 an $|\mathcal{S}|$-dimensional vector α_{β} such that $V^{\beta}(b) = \sum_{s \in \mathcal{S}} b(s) \alpha_{\beta}(s)$. If we consider α_{β} as a column vector, then in matrix notation, $V^{\beta}(b) = b \cdot \alpha_{\beta}$. Furthermore, using the recursive definition of conditional

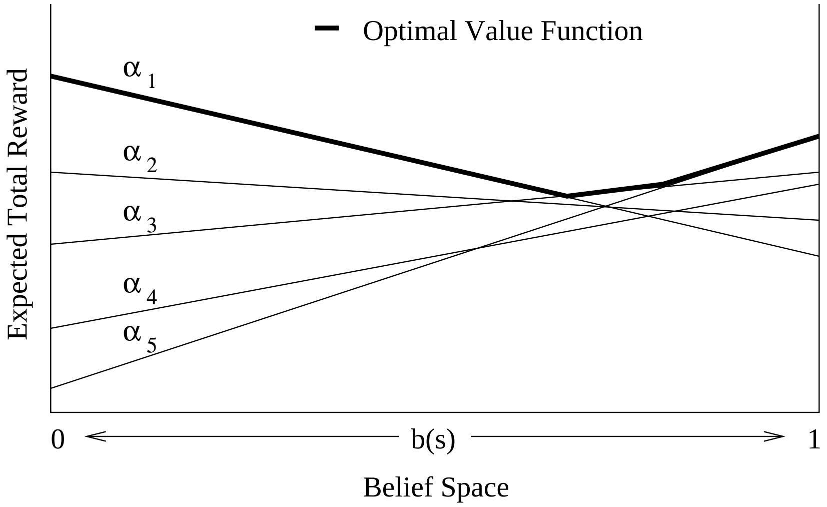


Figure 2.3: Geometric View of Value Function

plans, we can compute α-vectors recursively:

$$\alpha_{\langle a,\sigma\rangle} = R_a + \sum_{z \in \mathcal{Z}} M_{a,z}\alpha_{\sigma(z)} \tag{2.11}$$

For a finite horizon h, an optimal policy π^h consists of the best conditional plans for each initial belief state. Although there 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.

Figure 2.3 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 $\beta = \langle a, \sigma\rangle$ as defined in Equation 2.11. The upper surface of those α-vectors is a piecewise-linear and convex (PWLC) function corresponding to the optimal value function.

In general, as shown by Sondik [34], optimal value functions for finite-horizon problems are PWLC and can be represented succinctly by a finite collection of α-vectors. As for infinite-horizon POMDPs, the optimal value function is convex but not necessarily piecewise-linear. On the other hand, there always exists a sequence of PWLC value functions that converge in the limit to the optimal value function. This has considerably influenced the design of algorithms to find optimal policies, since most of the algorithms restrict their search to the set of policies with PWLC value functions.

2.3 Solution Algorithms

In the past 30 years, several algorithms have been proposed to solve POMDPs. We can distinguish value iteration algorithms and policy iteration algorithms. Value iteration algorithms indirectly compute an optimal policy by iteratively refining its value function, whereas policy iteration algorithms directly refine a policy until it is optimal. Below, the incremental pruning algorithm [38] is reviewed as it is the fastest known value iteration algorithm [5] for the class of *polynomially action-output-bounded* POMDPs [27] (a class of tractable POMDPs also reviewed below). Note as well that many aspects of this algorithm will be reused by the algorithms introduced for error bound computation and approximation search. Among the policy iteration algorithms, Hansen's algorithm [13, 14] is reviewed because it is conceptually simple (compared to Sondik's algorithm [35, 36]), it has been observed to outperform value iteration algorithms on several examples and it circumvents the need for belief state monitoring by computing policies represented by finite state controllers.

2.3.1 Incremental Pruning

Incremental pruning was first introduced by Zhang and Liu [38] and then analyzed by Cassandra, Littman and Zhang [5] from a computational complexity point of view. For finite-horizon POMDPs, this algorithm incrementally constructs a k-step optimal policy π^k by dynamic programming (DP). Given V^{k-1}, the optimal value function for $k-1$ stages to go, V^k is computed in a greedy fashion as follows. Here b_z^a denotes the belief state that follows b when action a is executed and observation z is experienced:

$$V^k(b) = \max_{a \in \mathcal{A}} \; bR_a + \gamma Pr(z|b,a)V^{k-1}(b_z^a) \tag{2.12}$$

Incremental pruning decomposes this DP backup (Equation 2.12) in three steps (Equations 2.13, 2.14 and 2.15). Two intermediary PWLC value functions ($V_{a,z}^k$ and V_a^k) are first computed and then the optimal value function (V^k) is computed. Roughly speaking, $V_{a,z}^k$ corresponds to the expected return of executing action a and making observation z followed by executing an optimal $(k-1)$-step policy π^{k-1}, whereas V_a^k corresponds to the expected return of executing action a and making some *unknown* observation z, followed by executing an optimal $(k-1)$-step policy π^{k-1}.

$$\begin{aligned} V_{a,z}^k(b) &= \frac{bR_a}{|\mathcal{Z}|} + \gamma Pr(z|b,a)V^{k-1}(b_z^a) && (2.13)\\ V_a^k(b) &= \sum_{z \in \mathcal{Z}} V_{a,z}^k(b) && (2.14)\\ V^k(b) &= \max_a \; V_a^k(b) && (2.15) \end{aligned}$$

Using Sondik's observation that PWLC value functions are representable by a finite collection of α-vectors, let $\aleph$ be a set of α-vectors whose upper surface corresponds to V. The above three equations can be rewritten in terms of sets of α-vectors as follows:

$$\begin{aligned} \aleph_{a,z}^k &= \{\frac{R_a}{|\mathcal{Z}|} + \gamma M_{a,z}\alpha | \alpha \in \aleph^{k-1}\} && (2.16)\\ \aleph_a^k &= \bigoplus_{z \in \mathcal{Z}} \aleph_{a,z}^k && (2.17)\\ \aleph^k &= \bigcup_{a \in \mathcal{A}} \aleph_a^k && (2.18) \end{aligned}$$

The symbol $\oplus$ means pairwise addition (i.e., $\aleph \oplus \aleph' = \{\alpha + \alpha' | \alpha \in \aleph \;\wedge\; \alpha' \in \aleph'\}$). The key to incremental pruning is to reduce the set of α-vectors representing

$$\begin{array}{rll} \max & x & \\ \text{s.t.} & b(\alpha - \alpha') \geq x & \forall \alpha' \in \aleph - \{\alpha\} \\ & \sum_s b(s) = 0 & \\ & b(s) \geq 0 & \forall s \in \mathcal{S} \end{array}$$

Table 2.1: LP-dominance test: vector α is dominated by $\aleph - \{\alpha\}$ when $x \leq 0$.

each value function in Equations 2.16, 2.17 and 2.18 by pruning all the dominated vectors (vectors that are not part of the upper surface). For example, in Figure 2.3, α_2 and α_4 correspond to conditional plans that are suboptimal and therefore can be removed without changing the upper surface. Dominated α-vectors can be detected in two ways: by *pointwise dominance* and by *linear programming* (LP). 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., $\alpha(s) \leq \alpha'(s)\ \forall s \in \mathcal{S}$). In Figure 2.3, 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 2.1).

The full DP procedure is given in Table 2.2. Its worst case running time requires the solution of $O(|\mathcal{A}||\aleph^{k-1}|^{|\mathcal{Z}|})$ LPs. LPs are taken as a basic operation because they are computationally intensive. This shows an exponential computational complexity with respect to the observation space. However, in practice, pruning helps to reduce considerably computation time. Moreover, for the class of polynomially action-output-bounded POMDPs [27]—which consists of POMDPs such that for all actions a the size of $\aleph_a^k$ (after pruning) grows polynomially with respect to $\aleph^{k-1}$—the reduction obtained by pruning makes the DP procedure tractable (i.e., polynomial in $|\mathcal{A}|$, $|\mathcal{Z}|$ and $|\aleph^{k-1}|$).

For finite horizon h, the incremental pruning algorithm determines an optimal policy π^h by iterating the above DP procedure h times. A sequence $\langle \aleph^0, \aleph^1, \ldots, \aleph^h \rangle$ of sets of α-vectors is computed and the optimal policy is implicitly defined by the

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

Table 2.2: DP procedure

last set $\aleph^h$. In practice, the optimal policy can be represented explicitly as a mapping from belief states to actions or as a finite state controller. If a mapping is used, then an action is stored with each α-vector of the h sets $\aleph^i$ $(1 \leq i \leq h)$. The action stored with a given vector α_β is the first action a that would be executed when implementing the corresponding conditional plan $\beta = \langle a, \sigma \rangle$. This scheme indirectly provides a mapping from belief states to actions since there is a mapping from belief states to maximizing α-vectors. At execution time, for time step t, the agent simply updates its belief state, determines the α-vector that maximizes its expected total return and executes the action stored with it. Alternatively, as pointed out by Kaelbling, Littman and Cassandra [22], the optimal policy can also be represented as a finite state controller. Explanations concerning finite state controllers are deferred to the next section when Hansen's policy iteration algorithm is introduced.

For infinite-horizon POMDPs, an ϵ-optimal policy (policy whose value function differs from the optimal infinite-horizon policy π^* by at most ϵ for every belief state) is also found by iterating the DP procedure. The idea is to find a k-step optimal policy π^k and to execute at every time step the action stored with the best α-vector in $\aleph^k$. Let's call $\hat{\pi}^k$ and $\hat{V}^k$ the policy and value function corresponding to this strategy. $\hat{\pi}^k$ differs from π^k since at time step t the action prescribed by $\hat{\pi}^k$ is stored with the maximizing α-vector in $\aleph^k$ whereas for π^k it is stored with the maximizing α-vector in $\aleph^{k-t}$. An important question concerning the quality of $\hat{\pi}^k$ is: how far is $\hat{V}^k$ from 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 [32]. 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.

2.3.2 Policy Iteration

A short description of Hansen's policy iteration algorithm is presented. It differs from value iteration algorithms because it conducts a search directly within the space of policies. On the other hand, the space of policies is huge, so it restricts its search to the set of policies that can be represented by finite state controllers.

A finite state controller is essentially a directed graph with vertices labelled by actions and edges labelled by observations (Figure 2.4). Vertices do not (necessarily) correspond to world states or belief states, they correspond to conditional plans. The vertex corresponding to conditional plan $\beta = \langle a, \sigma \rangle$ is labelled with action a and its outward edges correspond to the observation strategy σ. At run time, the agent executes the action labelling the current active vertex. It then makes an observation z which is used to update the active vertex by following the outward edge labelled 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 maximizing conditional plan, that is, the conditional plan with a corresponding α vector that maximizes the expected total return for 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 vertices of π.

Hansen's algorithm can be used to solve discounted, infinite-horizon POMDPs.

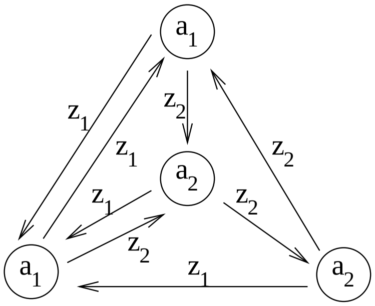


Figure 2.4: Finite state controller for a simple POMDP with two actions and two observations.

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 the value function V^{π_i} of π_i.
- *Policy improvement*: perform a dynamic programming backup and extract the next finite state automaton π_{i+1}.

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

$$\alpha_{\beta=\langle a,\sigma\rangle}(s) = R_a(s) + \gamma \sum_{s' \in \mathcal{S}} Pr(s'|s,a) \sum_{z' \in \mathcal{Z}} Pr(z'|a,s') \alpha_{\sigma(z')}(s')$$

Each equation gives the expected total reward for executing the conditional plan β associated with some vector $\alpha \in \aleph^\pi$ at some initial world state $s \in \mathcal{S}$.

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. The α-vectors in $\aleph^{\pi_i}_+$ are used to modify the vertices and the edges of the controller π_i

according to three simple rules, yielding the next controller π_{i+1}. We refer the reader to [13, 14] for the details of those rules as they are not essential to the development of this thesis. In practice, certain POMDPs have optimal policies corresponding to *infinite* state controllers. As with incremental pruning, one can find an ϵ-optimal finite state controller by ensuring that the Bellman residual is small enough. The argument is identical to the one for incremental pruning since both use the same DP procedure to compute backups.

Empirically, it has been observed that Hansen's algorithm requires fewer iterations than incremental pruning to converge to an ϵ-optimal policy. On the other hand, each iteration of Hansen's algorithm carries a policy evaluation step and a DP backup, whereas an iteration of incremental pruning only performs a DP backup. In practice, because the running time of a policy evaluation step is negligeable when compared to a DP backup, Hansen's algorithm usually runs faster. Although it runs faster, its computational complexity is as intractable as incremental pruning since both iterate the same DP procedure.

2.4 Performance issues

Solving POMDPs is a notoriously hard problem. In fact, finding an optimal policy has been shown to be PSPACE-complete for finite-horizon problems [28] and verifying the existence of a policy with a value function greater than some threshold for a given initial belief state is undecidable [30]. Yet, POMDPs remain a very attractive framework to conduct planning under uncertainty and numerous algorithms have been proposed in the literature. In practice, the PSPACE completeness and undecidability of POMDPs translate into two sources of intractability that afflict all current algorithms. In this section, a brief review of those sources of intractability and how some researchers have tried to mitigate them using factored representations is presented. Finally, as mentioned earlier, although the complexity of solving POMDPs is not the focus of this thesis, it will influence the design of some algorithms in

Chapters 3 and 4 and the experiments in Chapter 5.

2.4.1 Sources of Intractability

All current algorithms that solve POMDPs exactly are plagued by two sources of intractability:

- The number of α-vectors may grow exponentially with the size of the observation space $\mathcal{Z}$ and doubly exponentially with the horizon h.
- The dimensionality of each α-vector is equal to the size of the state space $\mathcal{S}$.

The first problem is easily observed when examining the DP procedure used in incremental pruning and in Hansen's algorithm. For each backup, $O(|\mathcal{A}||\aleph|^{|\mathcal{Z}|})$ α-vectors may be generated if no vector is pruned. Consequently, at the k^{th} iteration, there could be as many as $O(|\mathcal{A}|^k|\aleph|^{|\mathcal{Z}|^k})$ α-vectors.

As for dimensionality, most real world problems 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.

2.4.2 Factored Representation

Recently, some attention has been devoted to approaches that exploit some type of problem structure to essentially reduce the size of the state space. In particular, the use of *factored representations* that resemble classical AI representations allow the manipulation of a compact encoding of the state space. This encoding lets us reason at the variable level instead of the explicit state level, leading (potentially) to an exponential reduction in complexity. Decision trees (DTs) [2] and algebraic decision diagrams (ADDs) [18, 15] are examples of such encodings that can exploit

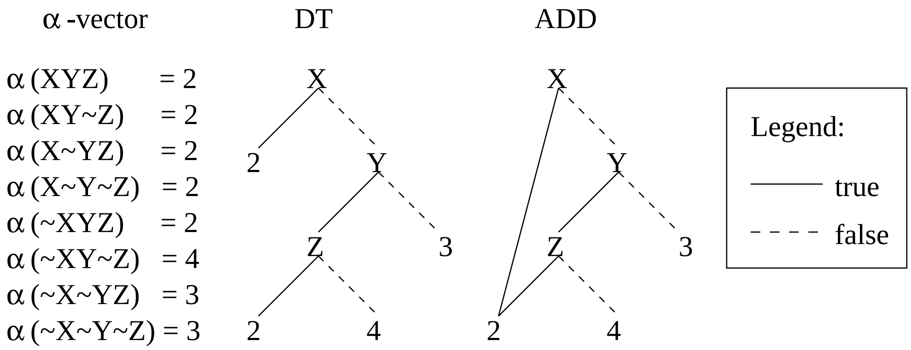


Figure 2.5: DT and ADD example

problem structure and have been shown to significantly speed up solution algorithms for several MDPs [18] and POMDPs [15].

DTs and ADDs can improve the running time of incremental pruning and Hansen's policy iteration by allowing the manipulation of an *abstract state space* instead of the explicit state space $\mathcal{S}$ [15]. More precisely, linear functions of the state space (such as α-vectors, belief states, transition functions and observation functions) can be compactly represented by aggregating some world states with identical values into a single abstract state. Figure 2.5 shows the DT and ADD representations of an α-vector. For DTs, an abstract state corresponds to a partial assignment of the state variables that define the state space. Each branch (path from the root to a leaf) encodes such a partial assignment by instantiating all the variables it traverses. The leaf at the end of a branch is labelled with the value that is shared by all the world states that satisfy the corresponding partial assignment. An ADD is roughly speaking a DT where branches are allowed to share identical subtrees. Consequently, ADDs can generate much *coarser* partitions of the state space than DTs and therefore, they will be used throughout the rest of this thesis.

Factored representations can speed up several operations on α-vectors such as addition, pointwise multiplication, dot product, pointwise dominance, LP-dominance, etc. All these operations have a running time that depends on the dimensionality of the vectors involved. When using ADDs, the dimensionality is reduced to the size of the intersection of the partitions used to represent the vectors involved. The intersection of two partitions p_1 and p_2 is a partition p such that all the abstract states in p are the intersection of an abstract state in p_1 and an abstract state in p_2.[2] For instance, when adding two vectors, α_1 and α_2, only values of abstract states that aggregate the same world states can be added together. These common abstract states happen to be the intersection of the abstract states used to represent α_1 and α_2.

Most exact algorithms for POMDPs (including incremental pruning and Hansen's policy iteration) make use of a DP procedure similar to the one described in Section 2.3.1. The running time of this procedure can be significantly improved by the use of a factored representation [15]. More precisely, the speed up comes from an improvement in the running time to solve LPs, which are the most computationally intensive operations in the procedure. ADDs can speed up LP-dominance tests to the extent where the intersection of the partitions of the α-vectors involved is smaller than the full state space. This fact is important for two reasons:

- The experiments carried out in Chapter 5 will be biased towards POMDPs that exhibit enough structure to allow α-vectors with partitions that "intersect well".
- The algorithms for computing error bounds in Chapters 3 and 4 are very similar to incremental pruning and make extensive use of linear programs that resemble LP-dominance tests. Therefore, the design of those algorithms will be influenced in a way to take advantage as much as possible of the speed up enabled by ADDs.

[2] Hansen and Feng describe an algorithm to compute the intersection of some partitions in [15].

2.5 Belief State Monitoring

The previous sections were concerned with the *offline* problem of solving a POMDP. In this section, an overview of the *online* belief state monitoring problem is presented. Depending on the representation of a policy (as a mapping from belief states to actions or as a finite state controller), the agent may or may not be required to monitor its belief state. Hence, our first concern will be to determine when and why belief state monitoring is desirable. Following, a brief overview of some exact and approximate methods for belief inference is presented. *Dynamic Bayesian networks* (DBNs) are introduced as a popular exact method. Despite its use of factored representations, the method is intractable for most non-trivial POMDPs, so we turn to approximate methods, among which *projection schemes* and *density trees* are reviewed.

2.5.1 Motivation

When a policy is represented as a mapping from belief states to actions, belief state monitoring is required; in contrast if it is represented by a finite state controller, it can be executed directly. In general, the optimal policy of any finite horizon POMDP can be represented by a finite state controller as pointed out by Kaelbling, Littman and Cassandra [22]. As for infinite-horizon POMDPs, Hansen's policy iteration algorithm can be used to find an ϵ-optimal policy, also represented as a finite state controller. At a first glance, the reader may wonder about the need for belief state monitoring, however three reasons may justify it in practice:

ϵ-optimal finite state controllers may not be appropriate representations for infinite-horizon policies

In many situations, the pratitioner may model an infinite-horizon, *undiscounted* POMDP as an infinite-horizon, *discounted* POMDP. This could be for computational purposes or simply to give a sense of urgency to the planning agent. In either case, the practitionner expects the agent to execute the policy for an infinite number of steps and it also expects the quality of the policy to remain constant over time. In other words, the expected total future reward for a given belief state should be the same regardless of the number of steps executed so far. If the policy is stationary, its quality is constant; however that is not the case for nonstationary policies. It turns out that mappings from belief states to actions represent stationary policies whereas finite state controllers don't [16]. At this point, it is not known to what extent the quality of a controller's policy deteriorates over time. In theory, an ϵ-optimal controller is guaranteed to yield an expected total discounted reward within ϵ of the optimal policy *only* for the initial time step. As time goes, the discount factor scales down the rewards as well as any divergence between the optimal policy and the controller's policy. Currently, the only way to prevent a deterioration in quality is to periodically reset the active vertex of the finite state controller to the best vertex given the current belief state. Knowledge of the current belief state requires that one perform belief state monitoring.

Most approximation algorithms for solving POMDPs represent policies as mappings from belief states to actions

As explained in the previous section, all current exact algorithms are intractable. Therefore, several heuristic approaches have been suggested to solve POMDPs approximately [17]. However, most of them assume a mapping from belief states to actions which requires belief state monitoring. Thus, the techniques described in the coming chapters are likely to integrate well with existing belief-based methods to solve POMDPs approximately.

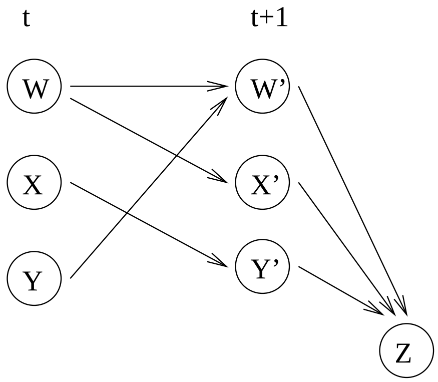
Belief state monitoring may be desirable in its on right for explanatory purposes

Knowledge of the current belief state provides useful information to the practitioner for understanding the subtleties of the policy in execution. This information can also be used to validate the reward, transition and observation functions which are usually based on rough estimates determined by an expert.

2.5.2 Dynamic Bayesian Networks (DBNs)

The task of belief state monitoring consists of updating the current belief state according to the previous action and observation. More precisely, b_{t+1} is obtained from b_t, a_t and z_{t+1} as in Equation 2.4. When represented as a vector, a belief state has dimensionality equal to the size of the state space. In general, a belief update requires some computation that is a low order polynomial in the size of the state space. When the state space is large (i.e., exponential in the number of state variables) the task becomes intractable.

Once again, factored representations have been proposed to mitigate the curse of dimensionality. Dynamic Bayesian networks (DBNs) [7] are very popular models that (potentially) allow efficient belief inference. The key is to exploit conditional independence to reason at the variable level instead of the state level. A DBN is an acyclic graph that encodes the transition and observation functions at each time step. Figure 2.6 illustrates a two-slice DBN for a simple POMDP of three state variables (W, X and Y) and one observation variable (Z). The unprimed state variables form a slice for time step t and the primed state variables form another slice for the next time step, $t + 1$. The edges in the graph summarize conditional independence between variables. That is, a variable is conditionally independent from all its ancestors given its parents. The transition and observation functions are numerically represented by conditional probability tables (CPTs), which indicate the conditional distributions $Pr(v|p_v^1, p_v^2, \ldots, p_v^n)$ of each variable v given its parents



W	Y	Pr(W’|W,Y)
1	1	0.3
1	0	0.8
0	1	0.4
0	0	0.5

Figure 2.6: Two-slice DBN

$p_v^1, p_v^2, \ldots, p_v^n$. As an example, the CPT for $Pr(W'|W,Y)$ is given in Figure 2.6.

Conditional independence can be leveraged to compute the next belief state b_{t+1} from the current belief state b_t (given some action a and some observation z). In the DBN, b_t and b_{t+1} correspond respectively to the joint distributions $Pr(W, X, Y)$ and $Pr(W', X', Y')$. A belief state update using the DBN paradigm essentially asks the question: $Pr(W', X', Y'|Z = z)$? The answer can be found using a variety of algorithms. A common method is to build a secondary structure, called a *clique tree*, and to perform belief inference on this structure. A clique tree is a compact factored representation of the joint distribution $Pr(W, X, Y, W', X', Y', Z)$ that exploits variable independence. The reader is referred to Huang and Darwiche [19] for a high level overview of clique tree construction. What is important to know is that clique tree encodings are exponential in the size of the largest clique and roughly speaking, clique size increases with dependencies (correlations).

Assuming monitoring is done using the clique tree representation of a DBN, a belief update is performed in three steps (see [19, 25] for more details on inference with DBNs using clique trees):

- Construct a clique tree that encodes the variable dependencies of the system dynamics (for a specific action and observation).

- Initialize the clique tree with the transition probabilities, the observation probabilities and the joint distribution of the state variables at the current time step (current belief state).

- Query the tree to obtain a joint distribution of the state variables at the next time step (next belief state).

The amount of computation required for those three steps is also exponential in the size of the largest clique. This suggests that monitoring can be efficient when the variables in each slice (defining each belief state) remain fairly independent as time goes. Unfortunately this rarely occurs in practice. As observed by Boyen and Koller [3], although the variables in the initial belief state may be independent and although at each time step only a small number of state variables become correlated, over time these correlations "bleed through" the DBN, rendering most (if not all) state variables dependent after some time.

Figure 2.7 shows several slices of an "unrolled" DBN (observations are omitted for simplicity). Suppose that the variables defining the current belief state at time step t are all independent. At the next time step, $t + 1$, variables sharing at least one parent become correlated and a few pairs of correlated variables show up. At subsequent time steps, the subsets of correlated variables (variables that share at least one ancestor) keep on growing until, at time step $t + 3$, the state variables are fully correlated. This is typical of most real world problems. Thus, exact belief state monitoring using DBNs usually remains prohibitively expensive.

2.5.3 Approximation Methods

In the literature, several methods have been proposed to efficiently approximate the belief state monitoring task. Among them, projection schemes and density trees are reviewed since the former integrates well with DBNs whereas the latter is closely related to DTs and ADDs.

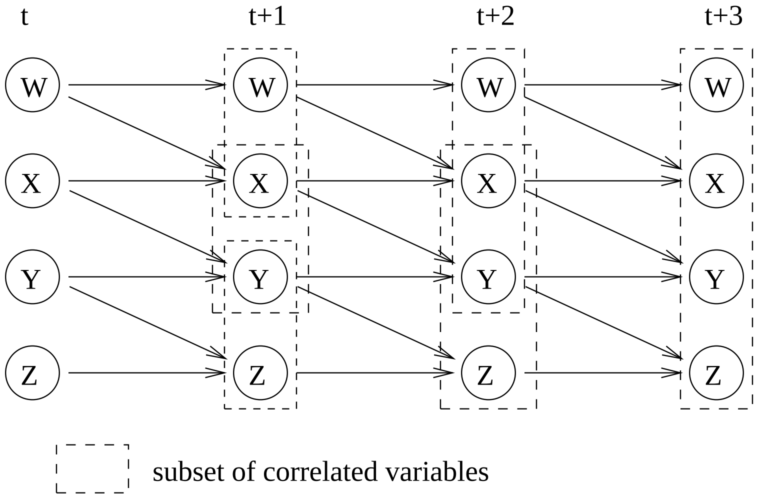


Figure 2.7: Unrolled DBN with no observation

Projection schemes

Boyen and Koller [3] devised a clever method to ensure that state variables remain fairly independent during the execution of a policy. Intuitively they consider *projection schemes* whereby the joint distribution of a belief state is approximated by the marginal distributions of some subsets of variables. They assume that these subsets partition the variable set in disjoint subsets. For each subset, its marginal is computed and the approximate belief state is formed by taking the product of the marginals as if they were independent. Thus only variables within the same subset can remain correlated in the approximate belief state.

In this thesis, a projection scheme is formally defined as a set of subsets of variables such that all variables are in at least one subset. This allows marginals with overlapping subsets of variables. For instance, the projection $S = \{XY, YZ\}$ approximates the joint $Pr(X, Y, Z)$ with the marginals $Pr(X, Y)$ and $Pr(Y, Z)$. Those marginals are stored in lieu of the belief state since they define *implicitly* the approximate belief state. An *explicit* joint of the approximate belief state can be computed by constructing a clique tree with cliques that correspond to the subsets of variables over which those marginals are defined. This clique tree, which encodes a belief state, is different from the DBN clique tree of Section 2.5.2 used for Bayesian inference. Note that some projection schemes with overlapping subsets are not useful because they don't correspond to any clique tree, therefore they will be discarded in practice. Given a projection scheme S (with a corresponding clique tree), one can perform an *approximate* belief state update as follows:

- Construct a DBN clique tree that encodes the variable dependencies of the system dynamics (for a specific action and observation) and the correlations that have been preserved by the marginals representing the current approximate belief state b_t.

- Initialize the DBN clique tree with the transition probabilities, the observation probabilities and the marginals representing the (approximate, factored) joint distribution of b_t.

- Query the DBN clique tree to obtain the joint distribution b_t^+ over the variables at the next time step.

- Project b_t^+ according to scheme S by computing the marginals representing the approximate belief state $b_{t+1} = S(b_t^+)$.

The complexity of belief state monitoring, which is exponential in the size of the largest (DBN) clique, is effectively reduced by choosing an adequate projection scheme that prevents (DBN) cliques from growing beyond some threshold.

It is also important to note that projection schemes perform *nonlinear* approximations of a belief state. In other words, the probabilities of each world state in the approximate belief state are nonlinear combinations of the probabilities of the world states in the exact belief state. For instance, the projection $S = \{X, Y\}$ maps the exact belief state b to the approximate belief state $S(b) = \tilde{b}$ according to the following nonlinear equations:

$$
\begin{aligned}
\tilde{b}(xy) &= b(x)b(y) = (b(xy) + b(x\bar{y}))(b(xy) + b(\bar{x}y)) \\
\tilde{b}(x\bar{y}) &= b(x)b(\bar{y}) = (b(x\bar{y}) + b(xy))(b(x\bar{y}) + b(\bar{x}\bar{y})) \\
\tilde{b}(\bar{x}y) &= b(\bar{x})b(y) = (b(\bar{x}y) + b(\bar{x}\bar{y}))(b(\bar{x}y) + b(xy)) \\
\tilde{b}(\bar{x}\bar{y}) &= b(\bar{x})b(\bar{y}) = (b(\bar{x}\bar{y}) + b(\bar{x}y))(b(\bar{x}\bar{y}) + b(x\bar{y}))
\end{aligned}
$$

The nonlinear properties of projection schemes will have an adverse effect on the quality of the error bounds computed in the next chapter.

Density Trees

Although, the value-directed framework introduced in this thesis is exemplified with projection schemes (mainly due to their nice integration with DBNs), it also applies to any *linear* approximation method. In fact, it is better suited for linear methods because the error bounds introduced in the next chapter can be made tighter. In this section, we briefly describe *density trees* [26] as they are a very good example of a linear approximation method and because they are closely related to DTs. The description given below is syntactically different from that of Koller and Fratkina [26], but it is conceptually equivalent.

A density tree defines an abstract state space representable by a DT that belief states are forced to fit in. The idea is to assume that all the world states aggregated in a belief state have the same value. Thus, a belief state is "compressed" in a DT by assigning to each abstract state the average probability of all the world states it aggregates. Figure 2.8, shows how some belief state b is approximated to $S(b)$ when applying some density tree S. A density tree with an abtract state space polynomially-sized in the number of variables offers an exponential reduction in the number of dimensions. It should also be clear to the reader that density tree approximations are *linear* since averaging is a linear operation. Finally, an approximate belief update using density tree S is performed as follows:

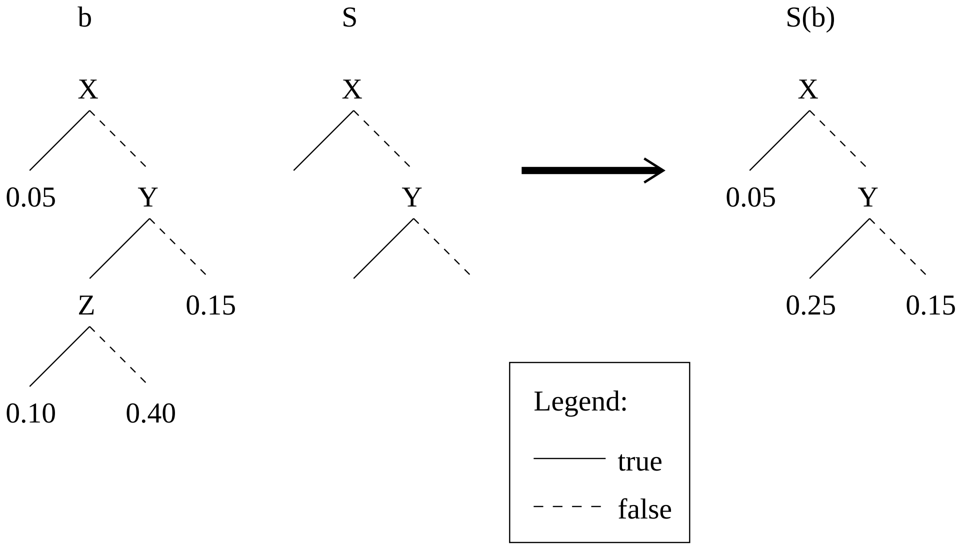


Figure 2.8: Density tree approximation

- Let b_t and $M_{a,z}$ be in factored forms such as DTs or ADDs.
- Compute $b_t^+ = b_t M_{a,z}/K$ by manipulating the factored representations.
- Compute the approximate belief state $b_{t+1} = S(b_t^+)$ by aggregating (through averaging) the leaves of b_t^+ until it satisfies the abstract state space defined by the density tree S.

2.6 Thesis Assumptions

This section ends the background material on POMDPs and belief inference by stating the assumptions underlying the value-directed framework introduced in the following chapters. The framework assumes that:

- A POMDP has been solved and an optimal (or ϵ-optimal) policy is already computed.
- The policy is given by a mapping from belief states to actions.

- The value function corresponds to (or is approximated by) the upper surface of a finite set of α-vectors.

- Since the POMDP has been solved, this set of α-vectors is assumed to be tractable. This means that the number of α-vectors as well as their *effective* dimensionality is reasonable. Here, by effective dimensionality we mean the reduced dimensionality obtained by a factored representation such as ADDs.

- Exact belief state monitoring is intractable and therefore needs to be approximated.

- The goodness of an approximation method is measured by the loss in expected total reward.

- We perform an *offline* search to find a good approximation method that enables fast *online* belief state monitoring.

Chapter 3

Value-Directed Approximations

We now introduce a general theoretical framework to analyze approximation methods from a *value-directed* point of view. The goal of this analysis is threefold:

- To provide some insights regarding the impact of approximations on decision quality.
- To develop bounds for the loss in expected value due to the use of a given approximation method.
- To develop algorithms that search within a class of methods for an approximation scheme that sacrifices as little as possible decision quality (by minimizing the bounds on expected loss).

The chapter is organized as follows. Sections 3.1 and 3.2 assume that we have been given some approximation scheme S for performing efficient belief updates, as well as the value function of an optimal policy that we wish to execute. The goal is to derive error bounds for applying the approximation scheme S while executing the optimal policy. Section 3.1 starts by analyzing the simple case where a single approximation is performed at some time step and Section 3.2 generalizes this analysis to the situation where the belief state is approximated repeatedly (i.e., at every time step).

It is important to note that this analysis is general in the sense that it can accomodate a wide range of approximation methods. In this thesis, the framework is illustrated by the class of projection schemes; however all the observations made and the bounds derived also apply to any linear approximation method (including density trees). Projection schemes were picked mainly due to their tight coupling with DBNs which are very popular for exact inference in dynamical systems.

Section 3.3 makes use of the bounds derived to develop algorithms that search for a good projection within the class of projection schemes. The class is first organized in a *lattice* which highlights the nicely structured partial ordering induced by the error bounds. Then we suggest a simple greedy algorithm that uses the lattice to guide its search.

The reader should also be warned that the framework introduced in this chapter is mainly *theoretical.* That is, the analysis performed is conceptually correct, but the proposed bounds are loose and the suggested algorithms are designed to be analyzed easily, not to be efficient. Chapter 4 addresses these practical issues.

3.1 Plan switching

Implementing a policy represented as a mapping from belief states to actions requires that one maintains a belief state, plugging this into the value function at each step, and executing the action associated with the maximizing α-vector. When the belief state b is approximated using an approximation scheme S, a suboptimal policy may be implemented since the maximizing vector for the approximate belief state $S(b)$ will be chosen rather than the maximizing vector for the exact belief state b. Furthermore this mistaken choice of vectors (hence actions) can be compounded with each further approximation at later stages of the process. To bound such error, we first define the notion of *plan switching.* We phrase our definitions in terms of finite-horizon value functions, introducing the minor variations needed for infinite-horizon problems later.

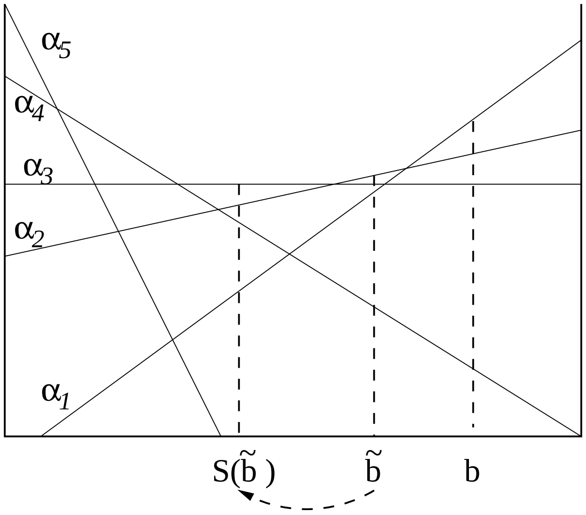


Figure 3.1: Relevant belief states at stage k

Suppose with k stages-to-go, the true belief state, had it been monitored accurately to that point, is b. However, due to previous belief state approximations we take our current belief state to be $\tilde{b}$. Now imagine our approximation scheme has been applied at time k to obtain $S(\tilde{b})$. Given $\aleph^k$, representing V^k, suppose the maximizing vectors associated with b, $\tilde{b}$ and $S(\tilde{b})$ are α_1, α_2 and α_3, respectively (see Figure 3.1). The approximation at stage k mistakenly induces the choice of the action associated with α_3 instead of α_2 at $\tilde{b}$; this incurs an error in decision quality of $b \cdot \alpha_2 - b \cdot \alpha_3$. While the optimal choice is in fact α_1, the unaccounted error $b \cdot \alpha_1 - b \cdot \alpha_2$ induced by the prior approximations will be viewed as caused by the earlier approximations; the goal at this point is simply to consider the error induced by the *current* approximation.

3.1.1 Switch set computations

The purpose of this section is to identify when the course of action is modified by an approximation scheme S. Assuming the current belief state is $\tilde{b}$, the future course of action is given by the conditional plan corresponding to the maximizing α-vector for

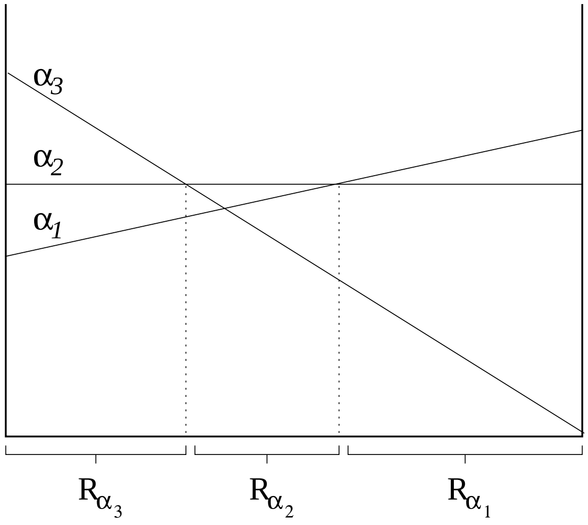


Figure 3.2: Regions for each maximizing α-vector

$\tilde{b}$. As long as S approximates $\tilde{b}$ to some belief state $S(\tilde{b})$ with the same maximizing α-vector, there is no loss in expected return since the same conditional plan gets executed. In fact, the maximizing vector α for belief state $\tilde{b}$ defines a region R_α of belief space in which $S(\tilde{b})$ can lie without altering the future course of action. In general, the set of α-vectors that make up the upper surface of a value function partition the belief space in regions R_{α_i} such that all belief states in a region are mapped to the same maximizing α-vector.[1] Figure 3.2 shows the regions R_{α_1}, R_{α_2} and R_{α_3} defined by the vectors α_1, α_2 and α_3.

The idea of this section is to pick some region R_α and to identify all the other regions $R_{\alpha'}$ such that there exists a belief state $b \in R_\alpha$ that gets mapped to $S(b) \in R_{\alpha'}$. More precisely, we identify for each $\alpha \in \aleph^k$, the set of vectors $Sw_S^k(\alpha)$ that the agent can *switch to* by approximating (with S) its current belief state $\tilde{b}$ given that $\tilde{b}$ identifies α as optimal. Formally, we define

[1] Belief states on the boundaries of adjacent regions have more than one maximizing α-vector, however this doesn't bear any consequence. If a *unique* maximizing α-vector is required, ties can be broken by always choosing the maximizing vector that comes first in some predetermined ordering of the vectors.

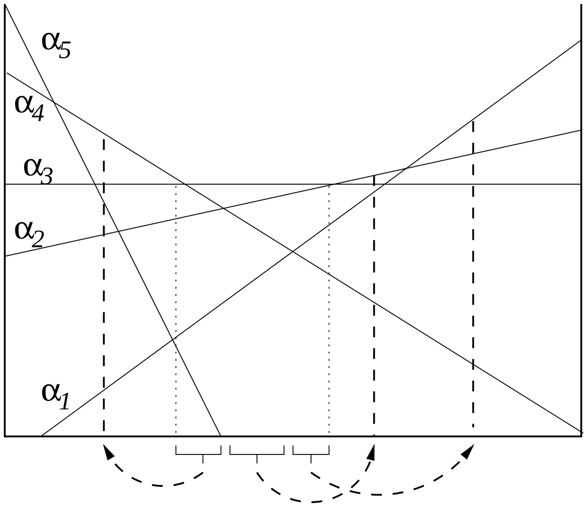


Figure 3.3: The Switch Set $Sw^k(\alpha_3)$ of α_3

$$Sw^k_S(\alpha) = \{\alpha' \in \aleph^k : \exists b \forall \bar{\alpha}(b \cdot \alpha \geq b \cdot \bar{\alpha}, S(b) \cdot \alpha' \geq S(b) \cdot \bar{\alpha})\}$$

Intuitively, this is the set of vectors we could choose as maximizing (thus implementing the corresponding conditional plan) due to belief state approximation. In Figure 3.3, we see that $Sw^k_S(\alpha_3) = \{\alpha_1, \alpha_2, \alpha_4\}$.

The set $Sw^k_S(\alpha_i)$ can be identified readily by solving a series of $O(|\aleph^k|)$ optimization problems. Each optimization problem tests the possibility of switching to a specific vector $\alpha_j \in \aleph^k$ and it is formulated as a (possibly nonlinear) program (Table 3.1). It is interesting to note the similarity between switch test programs (Table 3.1) and dominance test LPs (Table 2.1). In fact, if we remove constraints of type $S(b) \cdot (\alpha_j - \alpha_l) \geq d$, they are identical.

The objective function of a switch test program has a positive value whenever there is a belief state b such that α_i is optimal at b, and α_j is optimal at $S(b)$. In fact, we need only find a positive feasible solution, not an optimal one, to identify α_j as an element of $Sw^k_S(\alpha_i)$.

For linear approximation schemes (such as density trees), these problems

$$\begin{array}{lll}\max & x & \\ s.t. & b \cdot (\alpha_i - \alpha_l) \geq x & \forall l \neq i \\ & S(b) \cdot (\alpha_j - \alpha_l) \geq x & \forall l \neq j \\ & \sum_s b(s) = 1 & \\ & b(s) \geq 0 & \forall s \end{array}$$

Table 3.1: Switch test program to determine if there exists a belief state b such that the projection S leads to a switch from α_i to α_j. The variables are x and each component $b(s)$ of the vector b representing a belief state. α_i, α_j, α_l and S are fixed and therefore yield the variable coefficients.

are easily solvable linear programs (LPs). Unfortunately, projection schemes are nonlinear, making optimization (or identification of feasible solutions) more difficult. On the other hand, a projection scheme determines a set of linear constraints on the approximate belief state $S(b)$. In general, for POMDPs with binary variables, there is one linear constraint for each subset of the marginals defined by the projection scheme.[2] For instance, consider the projection scheme $S = \{CD, DE\}$ for a POMDP with 3 binary variables. This projection imposes the following linear constraints on $S(b)$:

$$\begin{array}{rclrcl} b(\emptyset) & = & b'(\emptyset) & b(C) & = & b'(C) \\ b(D) & = & b'(D) & b(E) & = & b'(E) \\ b(CD) & = & b'(CD) & b(DE) & = & b'(DE) \end{array}$$

Here b' denotes $S(b)$ and $b(XY)$ denotes the cumulative probability (according to belief state b) of all states that assign the value *true* to variables X and Y (i.e., $b(CD) \equiv b(cde) + b(cd\bar{e})$). $b(\emptyset)$ is the cumulative probability of all states (which is always 1). These constraints define an LP that can be used to construct a superset $\widehat{Sw}^k_S(\alpha_i)$ of $Sw^k_S(\alpha_i)$ (see Theorem 1). Given scheme $S = \{M_1, \ldots, M_n\}$, we define the LP in Table 3.2. When a feasible positive solution exists, α_j is added to the set $\widehat{Sw}^k_S(\alpha_i)$, though in fact, it may not be a member of $Sw^k_S(\alpha_i)$. If no positive solution exists, we know α_j is not in $Sw^k_S(\alpha_i)$ and it is not added to $\widehat{Sw}^k_S(\alpha_i)$.

[2] For POMDPs with *non-binary* variables, there is more than one equation per subset.

$$\begin{array}{lll}
\max & x & \\
s.t. & b \cdot (\alpha_i - \alpha_l) \geq x & \forall l \neq i \\
 & b' \cdot (\alpha_j - \alpha_l) \geq x & \forall l \neq j \\
 & b'(M) = b(M) & \forall M \subseteq M_l,\ 1 \leq l \leq n \\
 & \sum_s b(s) = 1 & \\
 & b(s) \geq 0 & \forall s \\
 & b'(s) \geq 0 & \forall s
\end{array}$$

Table 3.2: LP-switch test for projection schemes. The variables are x and each component $b(s)$ and $b'(s)$ of the vectors b and b' representing belief states. α_i, α_j, α_l and M, are fixed and therefore yield the variable coefficients.

Theorem 1 *Let $Sw_S(\alpha)$ and $\widehat{Sw}_S(\alpha)$ be the switch sets respectively constructed using the switch tests of Tables 3.1 and 3.2. Then, $Sw_S(\alpha) \subseteq \widehat{Sw}_S(\alpha)$.*

Proof Since a vector α' is added to $Sw_S(\alpha)$ and $\widehat{Sw}_S(\alpha)$ when the switch tests of Tables 3.1 and 3.2 have a positive feasible solution, it suffices to show that whenever the switch test of Table 3.1 has a positive feasible solution, then the switch test of Table 3.2 also has a positive feasible solution. Given a belief state b for which Table 3.1 has a positive objective function, then in Table 3.2, if we keep b the same and set b' to $S(b)$, the objective function is also positive. □

While the number of constraints of the type $b(M) = b'(M)$ is exponential in the size of the largest marginal, we expect that the number of variables in each marginal for a useful projection scheme will be bounded by a small constant. In this way, the number of constraints can be viewed as constant (i.e., independent of state space size).

The LP-switch tests in Table 3.1 (linear approximations) and Table 3.2 (projection schemes) are computationally intensive. Their complexity is similar to that of LP-dominance tests since they all have $O(|\mathcal{S}|)$ variables and $O(|\aleph^k|)$ constraints. Although the number of LP variables is exponential in the number of state variables, factored representations such as ADDs can be used to aggregate LP variables (which really correspond to world states). As with LP dominance tests, each constraint in LP-switch tests can be encoded with an ADD that potentially reduces the effective state space. The actual speed up induced by ADDs will be determined by the size of the intersection of the abstract state spaces defined by the ADD representation of each constraint.

In practice, density trees often have small intersections, whereas projection schemes don't. In fact one can show that for projection schemes, the resulting abstract space is always the whole state space. This is because of the constraints of type $b(M) = b'(M)$. Each such constraint where M is a single variable has an ADD representation defining a very small partition of two abstract states: the states for which variable M is true versus the states for which variable M is false. By definition, the state space is the intersection of those partitions, so if for each variable X there is a constraint $b(X) = b'(X)$, then the intersection is the whole state space. It turns out that for all projection schemes, each variable has a corresponding constraint $b(M) = b'(M)$ where M is that variable. This is because all variables must be contained in at least one marginal of the projection scheme.

Hence, LP-switch tests for projection schemes do *not* benefit from ADDs. On the other hand, it may be possible to still get a speed up by extending the ADD concept in such a way that we would aggregate, not only states with identical values, but also states whose values are linear combinations of each others. This refinement is a research problem in its own that remains to be explored.

In contrast, LP-switch tests for density trees do not suffer from this problem since density trees yield linear approximations. Linearity allows us to use the LP-switch test of Table 3.1 which doesn't have any constraints of the type $b(M) = b'(M)$.

In summary, the switch set of an α-vector identifies all other α-vectors (therefore conditional plans) that may be executed as a result of a single approximation S at the current time step. At stage k, the value function is determined by $|\aleph^k|$ α-vectors, so there are $|\aleph^k|$ switch sets and each of them requires the solution of $|\aleph^k|$

LPs, for a total of $O(|\aleph^k|^2)$ LPs. Once again, LPs are taken as a basic operation since they are computationally intensive. In comparison, when computing $\aleph^k$, the DP procedure may require as few as $\Omega(|\aleph^k|)$ LPs if no vector gets pruned. However, in practice, several α-vectors are usually pruned during the construction of $\aleph^k$, hence the running time to construct all switch sets tends to be similar to that of the DP procedure.

3.1.2 Switch set error bounds

Switch sets give us a very useful tool to bound the loss in expected value attributed to a single approximation. The idea is simply to measure the difference in expected return between the optimal conditional plan (had we not approximated the current belief state) and the worst conditional plan we could switch to. Let $B_S^j(\alpha)$ be an upper bound for the loss at stage j when approximation S is used and when α is viewed as optimal:

$$B_S^j(\alpha) = \max_b \max_{\alpha' \in Sw_S^j(\alpha)} b \cdot (\alpha - \alpha') \tag{3.1}$$

Also, let B_S^j be the greatest error introduced for any α-vector:

$$B_S^j = \max_{\alpha \in \aleph^j} B_S^j(\alpha) \tag{3.2}$$

In Equation 3.1, although the exact switch set Sw_S^j is used, one can also define B_S^j in terms of a superset $\widehat{Sw}_S^j$ with the consequence of a looser bound. In practice, since error at a belief state b is simply the expectation of the error at its component states, $B_S^j(\alpha)$ can be determined by comparing the vectors in $Sw_S^j(\alpha)$ with α componentwise (with the maximum component difference being $B_S^j(\alpha)$).

For a k-stage, finite-horizon POMDP, we can now bound the error in decision quality due to successive approximations with S. One simply has to figure out the bounds B_S^j for j stages to go. The cumulative bound U_S^k (for k stages) is the discounted sum of each stage's bound:

$$U_S^k = \sum_{j=1}^{k} \gamma^{k-j} B_S^j \tag{3.3}$$

As for infinite-horizon POMDPs, two possibilities arise: either we are given the set $\aleph^*$ corresponding to the optimal value function V^* or we only have access to the set $\aleph^k$ which implicitly determines the ϵ-optimal value function $\hat{V}^k$. For the optimal case, the bound U_S^* is pretty straightforward. It suffices to compute the switch sets of the vectors in the optimal set $\aleph^*$ and to derive the one-step error bound B_S^*. The upper bound U_S^* on the loss incurred by applying S indefinitely is simply the infinite discounted sum of B_S^*:

$$U_S^* = \frac{B_S^*}{1-\gamma} \tag{3.4}$$

As for ϵ-optimal policies, we can derive a bound $\hat{U}_S^k$, but this requires some work. We normally get an ϵ-optimal policy $\hat{\pi}^k$ by using at every step the same mapping from belief states to actions provided by the set $\aleph^k$ representing the optimal k-step value function V^k. If we had access to the set $\hat{\aleph}^k$ representing $\hat{V}^k$, we could compute the switch sets of each vector in $\hat{\aleph}^k$ and derive a one step bound $\hat{B}_S^k$. The cumulative bound $\hat{U}_S^k$ would simply be the infinite discounted sum of $\hat{B}_S^k$:

$$\begin{aligned} \hat{U}_S^k &= \frac{\hat{B}_S^k}{1-\gamma} && (3.5) \\ &\leq \frac{B_S^k + \mu}{1-\gamma} && (3.6) \end{aligned}$$

In general, $\hat{V}^k$ is not piecewise-linear nor convex so $\hat{\aleph}^k$ is not (necessarily) a finite set of vectors whose upper surface corresponds to $\hat{V}^k$. $\hat{\aleph}^k$ is a possibly infinite set of vectors such that each of them corresponds to the conditional plan that is prescribed by $\hat{\pi}^k$ for some belief state b. Unfortunately, we don't know $\hat{\aleph}^k$ and computing it is an impractical task as it is a possibly infinite set. On the other hand, we know that V^k and $\hat{V}^k$ are fairly close (within $\epsilon/2$ of each other), so one would expect B_S^k and $\hat{B}_S^k$ to be fairly close. In fact they are, but not as close as V^k and $\hat{V}^k$ are. We will show below that the difference between $\hat{B}_S^k$ and B_S^k can be bounded by some small number μ. Unfortunately there is no relationship between μ and ϵ other than $\mu \geq \epsilon$. Assuming $\hat{B}_S^k \leq B_S^k + \mu$, which we will prove shortly, then Equation 3.6 follows.

In order to show that $\hat{B}^k_S \leq B^k_S + \mu$, we need to establish 3 lemmas and to introduce some more notation. Define the L_∞ distance between two sets of α-vectors $\aleph^\pi$ and $\aleph^{\pi'}$ as follows:

$$\|\aleph^\pi - \aleph^{\pi'}\|_\infty = \max_{b \in \Delta(\mathcal{S})} \|\alpha_b - \alpha'_b\|_\infty \tag{3.7}$$

Here, α_b and α'_b are respectively the α-vectors in $\aleph^\pi$ and $\aleph^{\pi'}$ that correspond to the conditional plans prescribed by π and π' at belief state b. When the value functions V^π and $V^{\pi'}$ are piecewise-linear and convex, α_b and α'_b are essentially the maximizing vectors for belief state b. Note however that, in general, this is not the case for value functions (such as $\hat{V}^k$) that lack these properties.

The distance measure in Equation 3.7 is essentially a generalization of the L_∞ distance between value functions to an L_∞ distance between sets of α-vectors. This new distance measure gives us a handle to compare $\hat{B}^k_S$ and B^k_S. Roughly speaking, those bounds are defined in terms of α-vectors, so when $\hat{\aleph}^k$ and $\aleph^k$ are close (their L_∞ distance is small), then we are able to show that $\hat{B}^k_S$ and B^k_S are also close. Below, Lemma 1 relates the L_∞ distance between $\hat{\aleph}^k$ and $\aleph^k$ to the distance between $\hat{B}^k_S$ and B^k_S.

Lemma 1 *If $\|\hat{\aleph}^k - \aleph^k\|_\infty \leq \mu/2$ then $\hat{B}^k_S \leq B^k_S + \mu$.*

Proof Let's introduce an alternative definition for $B^k_S(\alpha)$ that is more convenient for this proof. The expression $B^k_S(\alpha)$ was initially defined in Equation 3.1 as an upper bound on the loss in expected total return due to the use of approximation S at step k assuming that the maximizing vector is α. The assumption that α is optimal implies that the current belief state $\tilde{b}$ lies in the belief space region R_α. Using this region notion, one can redefine $B^k_S(\alpha)$ with the following conceptually equivalent expression:

$$B^k_S(\alpha) = \max_{b \in \Delta(\mathcal{S})} \max_{\tilde{b} \in R_\alpha} b \cdot (\alpha_{\tilde{b}} - \alpha_{S(\tilde{b})})$$

The above expression can also be adapted to define $\hat{B}_S^k(\hat{\alpha})$, but with one slight semantic difference.

$$\hat{B}_S^k(\hat{\alpha}) = \max_{b \in \Delta(\mathcal{S})} \max_{\tilde{b} \in R_{\hat{\alpha}}} b \cdot (\hat{\alpha}_{\tilde{b}} - \hat{\alpha}_{S(\tilde{b})})$$

Since the value function $\hat{V}^k$ is not (necessarily) piecewise-linear nor convex, the region $R_{\hat{\alpha}}$ is not (necessarily) the belief space region for which $\hat{\alpha}$ dominates in $\hat{\aleph}^k$. It is rather the region for which the policy $\hat{\pi}^k$ prescribes the conditional plan corresponding to $\hat{\alpha}$. Similarly, the vectors $\hat{\alpha}_{\tilde{b}}$ and $\hat{\alpha}_{S(\tilde{b})}$ correspond to the conditional plans prescribed by $\hat{\pi}^k$ at belief states $\tilde{b}$ and $S(\tilde{b})$. We are now ready to prove the lemma:

$$\begin{aligned}
\hat{B}_S^k &= \max_{\hat{\alpha} \in \hat{\aleph}^k} \max_{b \in \Delta(\mathcal{S})} \max_{\tilde{b} \in R_{\hat{\alpha}}} b \cdot (\hat{\alpha}_{\tilde{b}} - \hat{\alpha}_{S(\tilde{b})}) \\
&= \max_{\hat{\alpha} \in \hat{\aleph}^k} \max_{b \in \Delta(\mathcal{S})} \max_{\tilde{b} \in R_{\hat{\alpha}}} b \cdot (\hat{\alpha}_{\tilde{b}} - \alpha_{\tilde{b}}) + b \cdot (\alpha_{\tilde{b}} - \alpha_{S(\tilde{b})}) + b \cdot (\alpha_{S(\tilde{b})} - \hat{\alpha}_{S(\tilde{b})}) \\
&\leq \max_{\hat{\alpha} \in \hat{\aleph}^k} \max_{b \in \Delta(\mathcal{S})} \max_{\tilde{b} \in R_{\hat{\alpha}}} \frac{\mu}{2} + b \cdot (\alpha_{\tilde{b}} - \alpha_{S(\tilde{b})}) + \frac{\mu}{2} && (3.8) \\
&\leq \max_{\alpha \in \aleph^k} \max_{b \in \Delta(\mathcal{S})} \max_{\tilde{b} \in R_{\alpha}} b \cdot (\alpha_{\tilde{b}} - \alpha_{S(\tilde{b})}) + \mu && (3.9) \\
&= B_S^k + \mu
\end{aligned}$$

In Equation 3.8, $b \cdot (\hat{\alpha}_{\tilde{b}} - \alpha_{\tilde{b}}) \leq \|\hat{\alpha}_{\tilde{b}} - \alpha_{\tilde{b}}\|_\infty \leq \mu/2$ and $b \cdot (\alpha_{S(\tilde{b})} - \hat{\alpha}_{S(\tilde{b})}) \leq \|\hat{\alpha}_{\tilde{b}} - \alpha_{\tilde{b}}\|_\infty \leq \mu/2$. In Equation 3.9, the regions $R_{\hat{\alpha}}$ are replaced by the regions R_α since every region $R_{\hat{\alpha}}$ is a subset of some region R_α. □

The above lemma provides a step in the right direction for the computation of $\hat{B}^k$, however it assumes the knowledge of $\|\hat{\aleph}^k - \aleph^k\|_\infty$. As mentioned earlier, we don't have access to $\hat{\aleph}^k$ and therefore we don't know its distance from $\aleph^k$. On the other hand, in the DP procedure, the value function $\hat{V}^k$ is not available either and yet, its distance from V^k can be evaluated. This is made possible by observing that H^d backups contract the L_∞ distance between value functions by a factor of at least γ. Assuming d is a mapping from belief states to actions, an H^d backup is similar to a DP backup except that the action executed at the first time step is not necessarily the one maximizing the expected total reward, but rather the one prescribed by the mapping d. Formally, we define H^dV as follows:

$$(H^dV)(b) = R(d(b), a) + \gamma \sum_z P(b_z^{d(b)}|b, d(b))V(b_z^{d(b)})$$

Below, Lemma 2 essentially shows that a generalized version of H^d, call it $\mathcal{H}^d$, is also a contraction mapping for the distance between sets of α-vectors. Semantically, $\mathcal{H}^d$ and H^d are the same, however $\mathcal{H}^d$ operates on sets of α-vectors whereas H^d operates on value functions.

$$\mathcal{H}^d\aleph = \{\alpha_b | \alpha_b = R_{d(b)} + \gamma \sum_z M_{d(b),z}\alpha_{b_z^{d(b)}} \text{ for some } \alpha_{b_z^{d(b)}} \in \aleph\}$$

Lemma 2 *Let $\aleph$ and $\aleph'$ be sets of α vectors, then $\mathcal{H}^d$ is a contraction mapping: $\|\mathcal{H}^d\aleph - \mathcal{H}^d\aleph'\|_\infty \leq \gamma\|\aleph - \aleph'\|_\infty$.*

Proof Let $\alpha_b^{\mathcal{H}^d\aleph}$ and $\alpha_b^{\mathcal{H}^d\aleph'}$ be the α-vectors respectively prescribed by $\mathcal{H}^d\aleph$ and $\mathcal{H}^d\aleph'$ at belief state b. Let a be the action prescribed by d at belief state b. We need to show that for any belief state b, $\|\alpha_b^{\mathcal{H}^d\aleph} - \alpha_b^{\mathcal{H}^d\aleph'}\|_\infty \leq \gamma\psi$ when $\|\aleph - \aleph'\|_\infty = \psi$.

$$\begin{aligned}
\|\alpha_b^{\mathcal{H}^d\aleph} - \alpha_b^{\mathcal{H}^d\aleph'}\|_\infty &= \|R_a + \gamma \sum_z M_{a,z}\alpha_{b_z^a}^{\aleph} - R_a + \gamma \sum_z M_{a,z}\alpha_{b_z^a}^{\aleph'}\|_\infty \\
&= \gamma\|M_a \sum_z M_z(\alpha_{b_z^a}^{\aleph} - \alpha_{b_z^a}^{\aleph'})\|_\infty \\
&\leq \gamma\|\sum_z M_z(\alpha_{b_z^a}^{\aleph} - \alpha_{b_z^a}^{\aleph'})\|_\infty \qquad (3.10) \\
&\leq \gamma\psi \qquad (3.11)
\end{aligned}$$

In Equation 3.10, M_a is a transition matrix such that the components of each row sum up to 1. Matrices with this property can only reduce the L_∞ distance. In Equation 3.11, the sum of all the matrices M_z is the identity matrix. Since $\|\alpha_{b_z^a}^{\aleph} - \alpha_{b_z^a}^{\aleph'}\|_\infty$ is at most ψ, then $\|\sum_z M_z(\alpha_{b_z^a}^{\aleph} - \alpha_{b_z^a}^{\aleph'})\|_\infty$ is also bounded by ψ. □

The above lemma shows that any $\mathcal{H}^d$ backup constitutes a contraction mapping. If we let d^k be the mapping from belief states to actions encoded by the set $\aleph^k$, then it should be clear that $\aleph^k = \mathcal{H}^{d^k}\aleph^{k-1}$ and similarly, $\hat{\aleph}^k = (\mathcal{H}^{d^k})^\infty\aleph^{k-1}$. Below, Lemma 3 shows how the contracting property of $\mathcal{H}^{d^k}$ and the fact that $\hat{\aleph}^k$ and $\aleph^k$ are obtained by $\mathcal{H}^{d^k}$ backups allow us to bound the L_∞ distance between $\hat{\aleph}^k$ and $\aleph^k$. The technique used is the same as for bounding $\|\hat{V}^k - V^k\|_\infty$ given the Bellman residual, however, working with sets of α-vectors instead of value functions.

Lemma 3 *If* $\|\aleph^k - \aleph^{k-1}\|_\infty = \psi$, *then* $\|\hat{\aleph}^k - \aleph^k\|_\infty \leq \gamma\psi/(1-\gamma)$.

Proof

$$\begin{aligned}
\|\hat{\aleph}^k - \aleph^k\|_\infty &= \|(\mathcal{H}^{d^k})^\infty\aleph^{k-1} - \mathcal{H}^{d^k}\aleph^{k-1}\|_\infty \\
&\leq \sum_{i=1}^{\infty}\|(\mathcal{H}^{d^k})^i\aleph^k - (\mathcal{H}^{d^k})^i\aleph^{k-1}\|_\infty \\
&\leq \sum_{i=1}^{\infty}\gamma^i\|\aleph^k - \aleph^{k-1}\|_\infty \\
&= \gamma\psi/(1-\gamma)
\end{aligned}$$

□

The above lemma provides the last piece of information necessary to relate $\hat{B}^k$ and B^k. To summarize, we can estimate the bounds $\hat{U}^k$ and $\hat{B}^k$ as follows:

Theorem 2 *Given the sets* $\aleph^k$ *and* $\aleph^{k-1}$ *such that* $\psi = \|\aleph^k - \aleph^{k-1}\|_\infty$, *let* $\mu = 2\gamma\psi/(1-\gamma)$, *then*

1. $\hat{B}^k \leq B^k + \mu$

2. $\hat{U}^k \leq (B^k + \mu)/(1-\gamma)$

Proof Compute $\psi = \|\aleph^k - \aleph^{k-1}\|_\infty$. By Lemmas 2 and 3, we know that

$$\|\hat{\aleph}^k - \aleph^k\|_\infty \leq \gamma\psi/(1-\gamma) = \mu/2$$

It follows that by Lemma 1,

$$\hat{B}^k \leq B^k + \mu$$

and consequently, using Equation 3.5:

$$\hat{U}^k \leq (B^k + \mu)/(1 - \gamma)$$

□

It is interesting to compare the value ψ to δ (Bellman residual) as well as the value μ to ϵ. Since the L_∞ distance between sets of α-vectors is at least as great as the L_∞ distance between the value functions they represent, it follows that $\psi \geq \delta$. Similarly, since μ and ϵ share the same relationship with respect to ψ and δ, it follows that $\mu \geq \epsilon$.

There is one important distinction that must be made between μ and ϵ. Since the DP backup is a contraction mapping for the L_∞ distance between value functions, one can make the Bellman residual, and consequently ϵ, arbitrarily small by doing enough DP backups. Although Lemma 2 shows that any $\mathcal{H}^d$ backup is a contraction mapping with respect to the L_∞ distance between sets of α-vectors, the author has been unable to prove or disprove the same for DP backups. This is an important question that remains to be solved. In the event where DP backups are not contraction mappings for the L_∞ distance between sets of α-vectors, then the practitionner has no control on ψ and μ and therefore it may not be possible to reduce them past an acceptable threshold. This means that the estimates of $\hat{U}^k$ and $\hat{B}^k$ could be much worse than their actual values.

3.2 Alternative plans

The cumulative error induced by repeated approximations can be bounded in a tighter way than simply taking the discounted sum of the B bounds (as it is done to compute the U bounds). The idea is to generate the set of *alternative plans* that may be executed as a result of both current and *future* approximations. Suppose that an agent, due to approximation at k stages to go, changes its belief state from b to $S(b)$. This can induce a change in the choice of optimal α-vector in $\aleph^k$, say from α_{β_1} to α_{β_2} where $\beta_1 = \langle a_1, \sigma_1 \rangle$ and $\beta_2 = \langle a_2, \sigma_2 \rangle$. However, even though the agent has switched and executed the action a_2 stored with α_{β_2}, it has not necessarily committed to implementing the entire conditional plan β_2. This is because further approximation at $k-1$ stages to go may cause it to depart from the implementation of the conditional plans prescribed by the observation strategy σ_2.

Suppose for instance that $\sigma_2(z) = \beta_3$. If z is observed, and the agent updates its (approximate) belief state $S(b)$ accurately to obtain $S(b)'$, then the maximizing vector at the next stage is necessarily α_{β_3}. But given that $S(b)'$ will be approximated before the maximizing vector is chosen, the agent may adopt some other continuation of the plan if α_{β_3} does not maximize value for the (second) approximated belief state $S(S(b)')$. In fact, the agent may implement the conditional plan $\beta_4 = \langle a_4, \sigma_4 \rangle$ of any vector α_{β_4} in the switch set $Sw^{k-1}(\alpha_{\beta_3})$. Notice that the value of the plan actually implemented—doing the action a_2 stored with α_{β_2}, followed by the action a_4 stored with α_{β_4}, and so on—may not be represented by any α-vector in $\aleph^k$.

3.2.1 Alt-set computations

We can actually construct the values of such plans, and thus obtain much tighter error bounds, while we perform dynamic programming. We recursively define the set of *alternative plans*, or *Alt*-set for each vector at each stage. We first define

$$Alt^1(\alpha) = Sw^1(\alpha)$$

That is, if α is optimal at stage 1, then any vector in its switch set can have its plan executed. The *future alternative set* for any $\alpha_\beta \in \aleph^k$, where $\beta = \langle a, \sigma \rangle$, is:

$$FAlt^k(\alpha_{\langle a,\sigma \rangle}) = \{\alpha_{\langle a,\sigma' \rangle} : (\forall z)\ \alpha_{\sigma'(z)} \in Alt^{k-1}(\alpha_{\sigma(z)})\} \tag{3.12}$$

If α is chosen to be executed at stage k, true expected value may in fact be given by *any* vector in $FAlt^k(\alpha)$, this is due to future switching of policies at stages following k. Finally, define

$$Alt^k(\alpha) = \cup\{FAlt^k(\alpha') : \alpha' \in Sw^k(\alpha)\} \tag{3.13}$$

If α is in fact optimal at stage k for a given belief state b, but b is approximated currently and at every future stage, then expected value might be reflected by any vector in $Alt^k(\alpha)$. These vectors correspond to every possible course of action that could be adopted because of approximation: if we switch vectors at stage k, we could begin to execute (the plan associated with) any $\alpha' \in Sw^k(\alpha)$; and if we begin executing α', we could end up executing (the plan associated with) any $\alpha'' \in FAlt^k(\alpha')$.

Given these *Alt*-sets, the error associated with belief state approximation is bounded by the maximum difference in value between any α and one of its *Alt*-vectors. These *FAlt* and *Alt*-sets can be computed by dynamic programming while a POMDP is being solved. Furthermore, the DP procedure to compute them is essentially the same as the incremental pruning DP procedure for $\aleph$-sets. The construction of *FAlt*-sets and *Alt*-sets can be decomposed in three steps that are identical to Equations 2.16, 2.17 and 2.18 with the proviso that the sets computed are different. The first step (Equation 3.14) constructs an intermediary *FAlt*-set that contains vectors corresponding, roughly speaking, to the expected value of executing action a followed by an alternative plan to $\sigma(z)$. The other two steps construct *FAlt*-sets and *Alt*-sets as initially defined in Equations 3.12 and 3.13.

$$\begin{aligned}
FAlt^k(\alpha_{\langle a,\sigma\rangle}, z) &= \{\frac{R_a}{|\mathcal{Z}|} + \gamma M_{a,z}\alpha : \alpha \in Alt^{k-1}(\alpha_{\sigma(z)})\} && (3.14)\\
FAlt^k(\alpha) &= \bigoplus_{z\in\mathcal{Z}} FAlt^k(\alpha, z) && (3.15)\\
Alt^k(\alpha) &= \bigcup_{\alpha'\in Sw^k(\alpha)} FAlt^k(\alpha') && (3.16)
\end{aligned}$$

The size of *Alt*-sets tends to be intractable as it grows exponentially. On

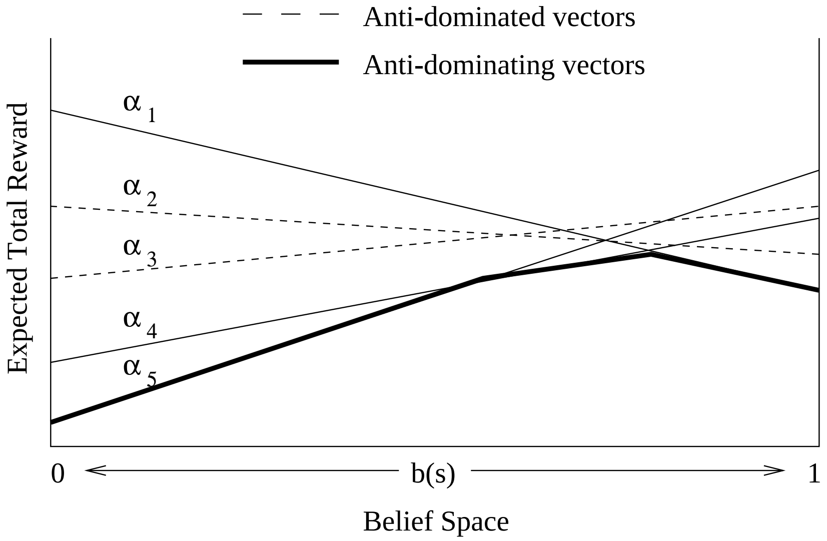


Figure 3.4: Lower surface of anti-dominating vectors

the other hand, our goal is to derive bounds on the loss in expected return. These bounds are determined by comparing the worst alternative plans to the optimal plan. The worst conditional plans correspond to the vectors that make up the *lower surface* of an *Alt*-set. Hence, we can significantly reduce the size of *Alt*-sets by pruning all *anti-dominated* vectors (vectors that are dominated from below by one or several other vectors as in Figure 3.4). This is the analog of pruning dominated vectors that are not part of the upper surface of an $\aleph$-set.

Table 3.3 shows an incremental pruning DP procedure to compute an *Alt*-set at stage k from the *Alt*-sets at stage $k-1$. The algorithm is essentially the same as that of Table 2.2 for computing an $\aleph$-set, with one slight difference: anti-pruning (pruning of anti-dominated vectors) is used instead of pruning. Two important observations can be made concerning the very close similarity of the two algorithms:

- **Implementation of the DP procedure for *Alt*-sets is easy.** Since the incremental pruning DP procedure for $\aleph$-sets is fairly straightforward, so is the incremental anti-pruning DP procedure for *Alt*-sets. Furthermore, their

```
PROCEDURE $DPaltset(\alpha, k)$
    $Alt^k(\alpha) \leftarrow \emptyset$
    for each $\alpha_{\langle a,\sigma\rangle} \in Sw^k(\alpha)$ do
        $FAlt^k(\alpha_{\langle a,\sigma\rangle}) \leftarrow \emptyset$
        for each $z \in \mathcal{Z}$ do
            $FAlt^k(\alpha_{\langle a,\sigma\rangle}, z) \leftarrow antiprune(\{\frac{R_a}{|\mathcal{Z}|} + \gamma M_{a,z}\alpha' : \alpha' \in Alt^{k-1}(\alpha_{\sigma(z)})\})$
            $FAlt^k(\alpha_{\langle a,\sigma\rangle}) \leftarrow antiprune(FAlt^k(\alpha_{\langle a,\sigma\rangle}) \oplus FAlt^k(\alpha_{\langle a,\sigma\rangle}, z))$
        $Alt^k(\alpha) \leftarrow antiprune(Alt^k(\alpha) \cup FAlt^k(\alpha_{\langle a,\sigma\rangle}))$
    return $Alt^k(\alpha)$
END PROCEDURE
```

Table 3.3: *Alt*-set DP procedure

high degree of similarity allows one DP procedure to be generalized with only a few lines of codes to handle both $\aleph$-sets and *Alt*-sets.

- **Constructing *Alt*-sets is an intractable task.** The DP procedures for $\aleph$-sets and *Alt*-sets are identical from a computational complexity point of view. We already know that solving POMDPs is intractable in general and so will be *Alt*-set construction. Moreover, at each stage k, one *Alt*-set is computed per α-vector for a total of $|\aleph^k|$ *Alt*-sets, whereas only one $\aleph$-set needs to be generated. On the other hand, as mentioned in Chapter 2, there is a class of POMDPs, namely the polynomially action-output-bounded POMDPs, for which the $\aleph$-set DP procedure is tractable. One question arises: Is the *Alt*-set DP procedure for those POMDPs also tractable? The author conjectures that the answer is *no*. Intuitively, lower surfaces of sets of α-vectors may grow exponentially while their upper surfaces grow polynomially.

3.2.2 Alt-set error bounds

Error bounds can be derived from *Alt*-sets for the cumulative loss in expected total return due to several consecutive approximations. For a k-stage POMDP, when the approximation S is used at every time step and α is the maximizing vector for the initial belief state, the cumulative error is bounded by the expression $E^k_S(\alpha)$. This bound can be defined in terms of the vectors in the set $Alt^k_S(\alpha)$:

$$E^k_S(\alpha) = \max_{b \in \Delta(\mathcal{S})} \max_{\alpha' \in Alt^k(\alpha)} b \cdot (\alpha - \alpha') \tag{3.17}$$

Alternatively, since $Alt^k_S(\alpha)$ is the union of the *FAlt*-sets of each of the vectors in $Sw^k_S(\alpha)$, the bound $E^k_S(\alpha)$ can also be defined in terms of similar bounds for *FAlt*-sets, which we denote $F^k_S(\alpha, \alpha')$:

$$\begin{aligned} F^k_S(\alpha, \alpha') &= \max_{b \in \Delta(\mathcal{S})} \max_{\alpha'' \in FAlt^k(\alpha')} b \cdot (\alpha - \alpha') && (3.18) \\ E^k_S(\alpha) &= \max_{\alpha' \in Sw^k(\alpha)} F^k_S(\alpha, \alpha') && (3.19) \end{aligned}$$

The bounds $E^k_S(\alpha)$ and $F^k_S(\alpha, \alpha')$ can be computed using simple pointwise comparison of α with each $\alpha' \in Alt^k_S(\alpha)$ or with each $\alpha'' \in FAlt^k_S(\alpha')$. Cumulative approximation error can be bounded globally using:

$$E^k_S = \max_{\alpha \in \aleph^k} E^k_S(\alpha) \tag{3.20}$$

Furthermore, $E^k_S \leq U^k_S$ since alternate vectors provide a much tighter way to measure cumulative error.

Theorem 3 *For any k stages to go ($k \geq 1$), $E^k_S \leq U^k_S$.*

Proof The proof is by induction. First show that the inequality holds for the base case at one stage to go:

$$\begin{aligned} E^1_S &= \max_{\alpha \in \aleph^1} \max_{b \in \Delta(\mathcal{S})} \max_{\alpha' \in Alt^1(\alpha)} b \cdot (\alpha - \alpha') \\ &= \max_{\alpha \in \aleph^1} \max_{b \in \Delta(\mathcal{S})} \max_{\alpha' \in Sw^1(\alpha)} b \cdot (\alpha - \alpha') \\ &= B^1_S \\ &= U^1_S \end{aligned}$$

Then assume that the inequality holds for k stages to go:

$$E^k_S \leq U^k_S$$

Finally show that the inequality also holds for $k+1$ stages to go:

$$\begin{aligned}
E_S^{k+1} &= \max_{\alpha\in\aleph^{k+1}} \max_{b\in\Delta(\mathcal{S})} \max_{\alpha'\in Sw^{k+1}(\alpha)} \max_{\alpha''\in FAlt^{k+1}(\alpha')} b\cdot(\alpha-\alpha'') \\
&= \max_{\alpha\in\aleph^{k+1}} \max_{b\in\Delta(\mathcal{S})} \max_{\alpha'\in Sw^{k+1}(\alpha)} \max_{\alpha''\in FAlt^{k+1}(\alpha')} b\cdot(\alpha-\alpha') + b\cdot(\alpha'-\alpha'') \\
&\leq \max_{\alpha\in\aleph^{k+1}} \max_{b\in\Delta(\mathcal{S})} \max_{\alpha'\in Sw^{k+1}(\alpha)} b\cdot(\alpha-\alpha') \\
&\quad + \max_{\alpha\in\aleph^{k+1}} \max_{b\in\Delta(\mathcal{S})} \max_{\alpha'\in Sw^{k+1}(\alpha)} \max_{\alpha''\in FAlt^{k+1}(\alpha')} b\cdot(\alpha'-\alpha'') \\
&\leq B_S^{k+1} + \gamma E_S^k \\
&\leq B_S^{k+1} + \gamma U_S^k \\
&= U_S^{k+1}
\end{aligned}$$

□

For an infinite-horizon problem where the optimal set of α-vectors $\aleph^*$ is available, we can compute switch sets once as in the computation of U_S^*. To compute a tighter bound E_S^*, we can construct k-stages of *Alt*-sets, backing up from $\aleph^*$.

$$\begin{aligned}
Alt^1(\alpha) &= Sw^*(\alpha) \\
FAlt^k(\alpha_{\langle a,\sigma\rangle}) &= \{\alpha_{\langle a,\sigma'\rangle} : (\forall z)\ \alpha_{\sigma'(z)} \in Alt^{k-1}(\alpha_{\sigma(z)})\} \\
Alt^k(\alpha) &= \cup\{FAlt^k(\alpha') : \alpha' \in Sw^*(\alpha)\}
\end{aligned}$$

The bound E_S^k is computed as for the finite-horizon case (Equations 3.17 and 3.20), and we set

$$E_S^* = E_S^k + \gamma^k U_S^* \tag{3.21}$$

As for infinite-horizon problems where an ϵ-optimal value function $\hat{V}^k$ was found, the derivation of an $\hat{E}_S^k$ bound appears to be complex. The problem stems from the fact that *Alt*-sets cannot be computed directly since the set of α-vectors $\hat{\aleph}^k$ is not available. On the other hand, information about the L_∞ distance between $\hat{\aleph}^k$ and $\aleph^k$ may provide a way to derive such a bound, as for $\hat{U}_S^k$. In any case, the derivation of an $\hat{E}_S^k$ is an open problem.

3.3 Value-directed search

The B, U and E error bounds derived in the previous section will now be leveraged to design algorithms to search for a good approximation method. The idea is to find an approximation scheme that *minimizes* those bounds. Unlike the previous section, which was generic in the sense that the bounds derived hold for a wide variety of approximation methods (including projection schemes, density trees and any other linear approximation method), we will now focus on algorithms to search within the class of projection schemes only. Despite their looser bounds (due to their non-linear properties which force us to construct supersets of the switch sets), projection schemes are of great interest since they mesh well with DBNs. Furthermore, as we will see in Section 3.3.1, the class exhibits a nice partial ordering illustrated by a *lattice*. This lattice turns out to be very useful to guide the search for a good projection scheme. Section 3.3.2 introduces a search algorithm which essentially does a greedy traversal of the lattice.

In the previous section, the bounds were derived assuming that the same approximation S was used at every time step and for every belief state. In fact, there was no real need to assume such a uniform approximation since the algorithms work equally well (provide legitimate bounds) when each switch set is computed with a (possibly) different approximation method. Therefore, in this section, we will conduct an independent search to find a good, possibly different, projection scheme for each switch set. Since we have to compute a switch set for each α-vector in the optimal set $\aleph$, we will store with each vector $\alpha \in \aleph$ a corresponding projection S_α. At run time, the agent will look up the maximizing vector α for its current belief state b and perform an approximation S_α that simplifies his belief state to $S_\alpha(b)$. The reader may wonder why approximation is needed if it is assumed that the agent knows its current belief state as well as the maximizing vector. As explained in Chapter 2, the state variables that define the belief state tend to become correlated over time; however, in one step, only a few correlations usually creep in. Hence, assuming a projection scheme is used at every time step to break the new correlations as they creep in, one can usually perform an exact belief update on the projected belief state. As a result, the agent can tailor its belief state approximation to provide good results for its currently anticipated course of action. This in turn will lead to much better performance than using a uniform scheme.

3.3.1 Lattice of projection schemes

We can structure the search for a projection scheme by considering the lattice of projection schemes defined by subset inclusion. Specifically, we say S_1 *contains* S_2 (written loosely $S_2 \subseteq S_1$) if every subset of S_2 is contained within some subset of S_1. This means that S_2 is a finer projection than S_1. The lattice of projections for three binary variables is illustrated in Figure 3.5. Each node represents the set of marginals defining some projection S. Above each node, the subsets corresponding to its constraining equations are listed (we refer to each such subset as a *constraint*). The finest projections (which are the "most approximate" since they assume more independence) are at the top of the lattice. Edges are labeled with the subset of variables corresponding to the single constraining equation that must be added to the parent's constraints in order to obtain the child's constraints.

It should be clear that if $S_2 \subseteq S_1$, then S_1 offers (not necessarily strictly) tighter bounds on error when used instead of S_2 at any point.

Theorem 4 *Let S_1 and S_2 be projection schemes such that $S_2 \subseteq S_1$ (every marginal of S_2 is a subset of some marginal of S_1). Then,*

1. $B^k_{S_1} \leq B^k_{S_2}$
2. $U^k_{S_1} \leq U^k_{S_2}$ *(similarly, $U^*_{S_1} \leq U^*_{S_2}$)*
3. $E^k_{S_1} \leq E^k_{S_2}$ *(similarly, $E^*_{S_1} \leq E^*_{S_2}$)*

Proof To see this, imagine that various approximation schemes are used for different α-vectors at different stages, and that S_2 is used whenever $\alpha \in \aleph^k$ is

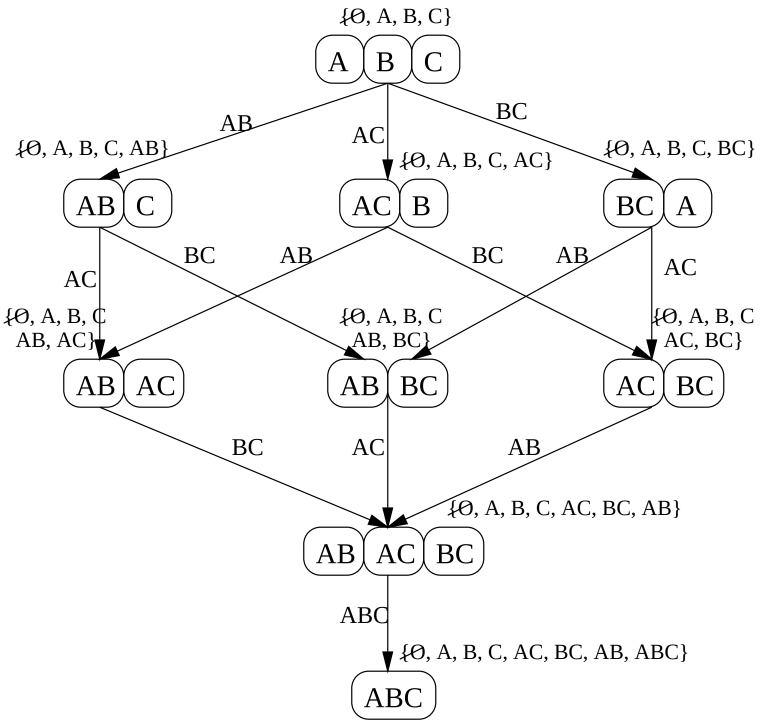


Figure 3.5: Lattice of Projection Schemes

chosen. If we keep everything fixed but replace S_2 with S_1 at α, we first observe that $Sw^k_{S_1}(\alpha) \subseteq Sw^k_{S_2}(\alpha)$. This ensures that $B^k_{S_1}(\alpha) \leq B^k_{S_2}(\alpha)$ and $B^k_{S_1} \leq B^k_{S_2}$. If all other projection operators are the same, then obviously $U^k_{S_1} \leq U^k_{S_2}$. Similar remarks apply to the infinite-horizon case. Furthermore, given the definition of *Alt*-sets, reducing the switch set for α at stage k by using S_1 instead of S_2 ensures that the *Alt*-sets at all preceding stages are no larger (and may well be smaller) than they would be if S_2 were used. For this reason, we have that $E^k_{S_1} \leq E^k_{S_2}$ (and similarly $E^*_{S_1} \leq E^*_{S_2}$). $\square$

Consequently, as we move down the lattice, the bound on approximation error gets smaller (i.e., our approximations improve, at least in the worst case). Of course, the computational effort of monitoring increases as well. The precise computational effort of monitoring will depend on the structure of the DBN for the POMDP dynamics and its interaction with the marginals given by the chosen projection scheme; however, the complexity of inference (i.e., the dominant factors in the corresponding clique tree), can be easily determined for any node in the lattice.

3.3.2 Greedy search

In a POMDP setting, the agent may have a bounded amount of time to make an online decision at each time-step. For this reason, efficient belief-state monitoring is crucial. However, just as solving the POMDP is viewed as an offline operation, so is the search for a good projection scheme. Thus it will generally pay to expend some computational effort to search for a good projection scheme that makes the appropriate tradeoff between decision quality and the complexity of belief state maintenance. For instance, if any scheme S with at most c constraints offers acceptable online performance, then the agent need only search the row of the lattice containing those projection schemes with c constraints. However, the size of this row is factorial in c. So instead we use the structure of the lattice to direct our attention toward reasonable projections.

We describe here a generic, greedy, anytime algorithm for finding a suitable projection scheme. We start with the root, and evaluate each of its children. The child that looks most "promising" is chosen as our current projection scheme. Its children are then evaluated, and so on; this continues until an approximation is found that incurs no error (specifically, each switch set is a singleton) or a threshold on the size of the projection is reached. We assume for simplicity that at most c constraints will be allowed. The search proceeds to depth $c-n$ in the lattice and at each node, at most nc children are evaluated, so a total of $O(nc^2 - cn^2)$ nodes are examined. Since c must be greater than n—the root node itself has n constraints—we assume $O(nc^2)$ complexity. The structure of the lattice ensures that decision quality (as measured by error bounds) cannot decrease at any step. We note that non-practical projections (projections for which we cannot construct a clique tree) are included in the lattice. In Figure 3.5, the only non-practical scheme is $S = \{AB, AC, BC\}$. During the search, it doesn't matter if a node corresponding to a non-practical scheme is traversed, as long as the final node is practical. If it is not practical, then the best practical sibling of that node is picked or we backtrack until a practical scheme is found. We also note that since this is a greedy approach, we may not discover the best projection with a fixed number of constraints. However, it is a well-structured search space and other search methods for navigating the lattice could be used.

We now describe an instantiation of this algorithm whereby the B-bound at a given stage k is minimized. We will later build on this basic search procedure to develop algorithms that minimize the U and E bounds of finite and infinite-horizon problems. Given a collection of α-vectors $\aleph^k$, we run the following search independently for each vector $\alpha \in \aleph^k$. The order does not matter; we will end up with a projection scheme S for each α-vector, which is applied whenever that α-vector is chosen as optimal at stage k. We essentially minimize (over S) each term $B_S^i(\alpha)$ in the bound B^k independently. For a given vector α, the search proceeds from the root in a greedy fashion. Each child S of the current node is evaluated by computing $B^k_S(\alpha)$, which basically requires that we compute the switch set $Sw^k_S(\alpha)$, which in turn requires the solution of $|\aleph^k|$ LP-switch tests. Once the projection schemes S_α for each α are found, the error bound B^k is given by the maximum bound $B^k(\alpha)$ as described in the previous section. The number of LPs that must be solved is $O(nc^2|\aleph^k|^2)$ since there are $O(|\aleph^k|)$ α-vectors and for each α-vector, the lattice search traverses $O(nc^2)$ nodes, each requiring the solution of $O(|\aleph^k|)$ LPs.

The above algorithm can be streamlined considerably. At a given node, when computing the bound $B^k_S(\alpha)$, it is not necessary to generate the entire switch set of α. Each vector $\alpha' \in \aleph^k$, if switched to, introduces an error of at most $\max_b\ b\cdot(\alpha-\alpha')$. Since $B^k_S(\alpha) = \max_{\alpha' \in Sw^k(\alpha)}\ \max_b\ b \cdot (\alpha - \alpha')$, we can test vectors α' in decreasing order of contributed error until one vector is found to be in the switch set. The bound $B^k_S(\alpha)$ is equal to the error contributed by this vector. As a result, instead of solving $|\aleph^k|$ LPs to determine the bound $B^k_S(\alpha)$ of each node visited, generally, only a few LPs need to be solved. The number of LPs can be further reduced by observing that any α-vector that is not in the switch set of a node won't be in the switch set of any of its children nor descendants. This is because descendants possess constraints that form a superset of this node's constraints. Thus, at any child, we can avoid computing the LP-switch tests for the vectors that are known not to be in that node's switch set. When those two techniques are combined, one can show that the number of LP-switch tests to be performed at each node is on average a small constant, which reduces the overall complexity of finding good projection schemes for each α-vector in $\aleph^k$ to $O(nc^2|\aleph^k|)$.

There is another possible speed-up. When testing whether two different schemes S_1 and S_2 allow switching to some α-vector, the LPs to be solved for each scheme are similar, differing only in the constraints dictated by each projection scheme. This similarity can be exploited computationally by using techniques that take advantage of the numerous common constraints if we solve similar LPs "concurrently" (for instance, by solving a stripped down LP that has only the common constraints and using the dual simplex method to account for the extra constraints). Though details are beyond the scope of this thesis, from experience, these techniques tend to be faster in practice than solving each LP from scratch. The greedy search can take full advantage of these techniques: each child has only one additional constraint (compared to its parent), so not only can structure be shared across children, but the parent's solution can be exploited as well. This computational trick could potentially reduce the worst-case running time to $O(nc|\aleph^k|)$ LPs. Further theoretical and practical investigation remains to be done.

The above B-bound search algorithm has a running time of $O(nc^2|\aleph^k|)$ LPs (or perhaps only $O(nc|\aleph^k|)$ LPs) whereas the $\aleph^k$-set DP procedure must solve at least $\Omega(|\aleph^k|)$ LPs. When integrating the B-bound search algorithm to the DP procedure for computing $\aleph^k$, the overhead incurred is at most a multiplicative factor nc^2 (or perhaps only nc). Although the usual sources of intractability (large state space and large $\aleph$-sets) limit the applicability of this search algorithm, it can be applied to POMDPs that can be readily solved today, since the overhead incurred is not too significant.

To conclude this chapter, a brief description of algorithms that extend the B-bound search to U-bounds and E-bounds is presented.

- **Finite-horizon U-bound search:** The U-bound for a k-stage POMDP is the discounted sum of the B-bounds for each stage. Hence, it suffices to minimize each B-bound in order to minimize the U-bound. Therefore, the search consists of k B-bound searches, one for each set $\aleph^i$ $(1 \leq i \leq k)$.

- **Infinite-horizon optimal U-bound search:** Similarly the bound U^* is minimized when the bound B^* is minimized. Thus, the search boils down to one B-bound search for the set $\aleph^*$.

- **Infinite-horizon ϵ-optimal U-bound search:** Once again, the upper bound

$(B^k + \mu)/(1 - \gamma)$ is minimized when B^k is minimized and therefore a single B-bound search is conducted for the set $\aleph^k$.

- **Finite-horizon E-bound search:** The bound E^k is minimized when the bounds $E^k(\alpha)$ are minimized for each vector $\alpha \in \aleph^k$. In turn, using Equation 3.19, the bound $E^k(\alpha)$ is minimized when the switch set $Sw^k(\alpha)$ contains vectors α' with corresponding error bounds $F^k(\alpha, \alpha')$ that are as small as possible. Hence one can conduct an E-bound search in the same way than a B-bound search with one slight modification: at each node, select the child with the most promising (smallest) E-bound. At i stages to go of a k-stage POMDP, the agent would first construct the *FAlt*-sets of each vector $\alpha \in \aleph^i$. Then the agent would search a good projection scheme for each vector $\alpha \in \aleph^i$. As for the B-bound search, the lattice is traversed in a greedy fashion. At each node S, the corresponding switch set $Sw^k_S(\alpha)$ is constructed and the bound $E^k_S(\alpha)$ is simply the worst bound $F^k_S(\alpha, \alpha')$ for some vector $\alpha' \in Sw^k_S(\alpha)$. Once the $|\aleph^i|$ projection searches are completed, the *Alt*-set of each vector is constructed and the process starts over for the previous stage until k stages to go.

- **Infinite-horizon optimal E-bound search:** According to Equation 3.21, the bound E^* is minimized when both bounds E^k and U^* are also minimized. Therefore, the search consists of k consecutive E-bound searches for the set $\aleph^*$. We should also conduct a B-bound search for $\aleph^*$ (to minimize U^*); however, this B-bound search is identical to the first E-bound search and therefore redundant. In practice, the projections applied would be those found during the k^{th} E-bound search.

Chapter 4

Vector Space Analysis

The value-directed framework introduced so far has allowed us to analyze the impact of approximate belief state monitoring on decision quality. This analysis was conceptually simple and remained at a fairly high level. The idea was simply to look at all the possible courses of action as a result of one approximation (switch set) or several consecutive approximations (*Alt*-set) and to derive several bounds, which, roughly speaking, measure the loss in expected total return assuming the worst possible course of action (we could switch to) was implemented. In turn, these bounds have allowed us to structure the class of projection schemes in a lattice, which can be traversed greedily to find a fairly good projection. From a decision theoretic point of view, this may all seem pretty satisfying, but from a practical perspective, the algorithms are generally intractable and from a conceptual perspective, the framework provides little insight as to which properties of projection schemes cause or prevent plan switching.

In order to provide some insights, this chapter analyzes approximation methods from a linear algebra perspective. More precisely, a vector space analysis is carried out. Appendix A provides an overview of some of the basics of linear algebra that will be useful for the development of the current chapter. Section 4.1 reveals that each projection scheme allows approximations only in some *directions* defined by a subspace. On the other hand, a similar analysis of the value function shows that α-vectors can determine "gradients" indicating the variability of the value function in different directions. Therefore, approximations in directions where the variance is similar for most α-vectors should be preferred since switching is less likely to happen.

From a practical perspective, these observations will be used in Section 4.2 to design faster algorithms than those proposed in Chapter 3, however yielding looser bounds. The LP-switch test introduced so far took into account approximation distance and direction. The idea will be to relax this LP to focus mainly on direction. As a result, the simplified switch test has an efficient solution algorithm that does not require an LP anymore.

Finally, Section 4.3 presents a vector-space search algorithm. It differs from previous value-directed search algorithms as it doesn't aim to directly minimize any error bound. Rather, it seeks a projection scheme that allows approximations in directions where α-vectors have a similar variance.

4.1 Vector space formulation

Given a projection S, let b and $b' = S(b)$ be points in belief space. Define $d = b' - b$ as the *displacement* vector that goes from b to b'. We are about to show that all such displacement vectors are part of a special *subspace* that is entirely determined by the projection S. A subspace is an important concept that allows us to characterize the possible *directions* of displacement vectors.

The subspace determined by a projection is actually defined by its marginals. For instance, assume a state space defined by the binary variables X and Y. Projection $S = \{X, Y\}$ determines a set of linear equations constraining the probability distribution b' in terms of b. These equations are the same as those of the form

$b(M) = b'(M)$ in Table 3.2.

$$\begin{aligned} b(xy) + b(x\bar{y}) + b(\bar{x}y) + b(\bar{x}\bar{y}) &= b'(xy) + b'(x\bar{y}) + b'(\bar{x}y) + b'(\bar{x}\bar{y}) \\ b(xy) + b(x\bar{y}) &= b'(xy) + b'(x\bar{y}) \\ b(xy) + b(\bar{x}y) &= b'(xy) + b'(\bar{x}y) \end{aligned}$$

Using the fact that $d = b' - b$, the above equations can be rewritten in terms of displacement vectors as follows:

$$\begin{aligned} d(xy) + d(x\bar{y}) + d(\bar{x}y) + d(\bar{x}\bar{y}) &= 0 \qquad (4.1) \\ d(xy) + d(x\bar{y}) &= 0 \qquad (4.2) \\ d(xy) + d(\bar{x}y) &= 0 \qquad (4.3) \end{aligned}$$

Geometrically, we can interpret each equation as a hyperplane and the intersection of those hyperplanes is a line through the origin representing a one-dimensional subspace (an example is given below to illustrate the resulting subspace). This subspace, which corresponds to the equations' solution space, captures the set of all displacement vectors resulting from the application of S. Since all possible displacement vectors lie on the same line, they must all have the same direction. Here, vectors with opposite orientations are assumed to have the same direction.

To illustrate, let $b(x) = 0.3$ and $b(y) = 0.4$. The approximate belief state, assuming the correlation between X and Y is broken, is computed as follows:

$$\begin{aligned} b'(xy) &= b(x)b(y) = 0.12 \\ b'(x\bar{y}) &= b(x)(1 - b(y)) = 0.18 \\ b'(\bar{x}y) &= (1 - b(x))b(y) = 0.28 \\ b'(\bar{x}\bar{y}) &= (1 - b(x)))(1 - b(y)) = 0.42 \end{aligned}$$

Figure 4.1 shows a three-dimensional belief space for belief states xy, $x\bar{y}$, $\bar{x}y$

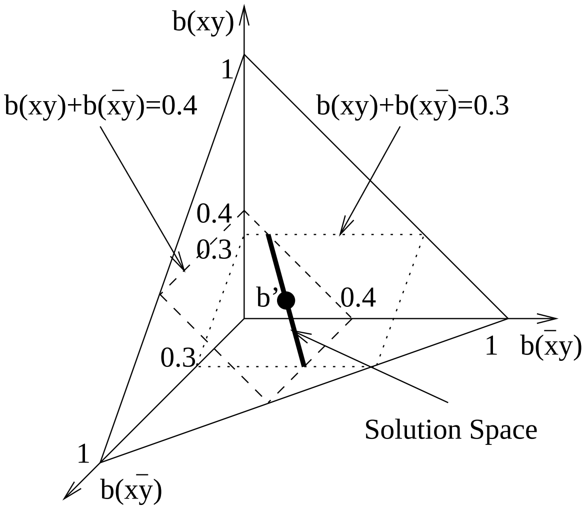


Figure 4.1: Solution space of possible exact belief states b

and $\bar{x}\bar{y}$.[1] All exact belief states b that satisfy $b(x) = 0.3$ lie in a hyperplane and similarly for $b(y) = 0.4$. Their intersection is the line of exact belief states that are mapped to b' by S. Since b' also lies on this line, then all the displacement vectors between b and b' have the same direction. Had we picked marginals different than 0.3 and 0.4, the hyperplanes would move, but remain parallel, and so would their intersection, yielding a line with the same direction.

Definition 1 *Let D_S be the subspace spanned by the set of all displacement vectors possibly induced by projection scheme S.*

In general, the set of displacement vectors induced by a projection S lies in a $(2^n - c)$-dimensional subspace, which we denote D_S. Here n is the number of state variables and c is the number of constraints (or marginals). D_S has $2^n - c$ dimensions since it is the solution space of c linearly independent equations, each corresponding to a constraint $d(M) = 0$. Equations 4.1, 4.2 and 4.3 are examples of constraints of the form $d(M) = 0$ for $M = \emptyset$, $M = X$ and $M = Y$ respectively.

[1] Only three dimensions corresponding to $b(xy)$, $b(x\bar{y})$ and $b(\bar{x}y)$ are represented since $b(\bar{x}\bar{y})$ follows from the fact that probability distributions sum up to 1.

Those equations are linearly independent since the marginals M in each constraint correspond to different subsets of variables. This becomes more obvious when we rewrite each equation $d(M) = 0$ as a dot product $v_M \cdot d = 0$. Here v_M is a vector of 0's and 1's, such that every component set to 1 corresponds to a state with all variables in M set to true, and every component set to 0 corresponds to a state with at least one variable in M set to false. The following three vectors correspond to the marginals determined by the projection $S = \{X, Y\}$.

$$\begin{array}{ccccccccc} & & & xy & x\bar{y} & \bar{x}y & \bar{x}\bar{y} & \\ v_\emptyset & = & (& 1 & 1 & 1 & 1 &) \\ v_X & = & (& 1 & 1 & 0 & 0 &) \\ v_Y & = & (& 1 & 0 & 1 & 0 &) \end{array}$$

Definition 2 *Let $D_S^\perp$ be the subspace spanned by the vectors v_M corresponding to the marginals defined by projection scheme S.*

The subspace $D_S^\perp$ is closely related to D_S since it is its null space, or in other words, $D_S^\perp$ is the set of all vectors perpendicular to every vector in D^S.

Proposition 1 *$D_S^\perp$ is the null space of D_S.*

Proof Two vectors are perpendicular to each other when their dot product is 0. Since $v_M \cdot d = 0$ for all vectors v_M that span $D_S^\perp$ and for all displacement vectors d that span D_S, it follows that all vectors in $D_S^\perp$ are perpendicular to every vector in D_S. Similarly one can show that all vectors perpendicular to D_S are in $D_S^\perp$. $\square$

4.2 Vector space switch test

The subspaces D_S and $D_S^\perp$ are very important as they allow us to design a simplified switch test. Let's say we are interested to know if $\alpha_j \in Sw(\alpha_i)$. Define $\alpha_{ij} = \alpha_i - \alpha_j$, to be a vector representing the difference in expected value for executing α_j instead of α_i. When $\alpha_{ij} \in D_S^{\perp}$, we can show that there is *no* switch from α_i to α_j. On the other hand, if $\alpha_{ij} \notin D_S^{\perp}$, there *may* be a switch. Thus, based on whether α_{ij} is in $D_S^{\perp}$ or not one can construct a superset of the switch set.

The idea behind the vector space (VS) switch test, is best explained by a picture. Figure 4.2 shows a top down view of a two dimensional belief space[2] with two different partitions. The first partition is given by the solid lines which define $|\aleph|$ regions, each corresponding to the area R_α for each vector $\alpha \in \aleph$. The second partition is given by the dotted line which separates two regions corresponding to the areas where α_i dominates α_j and vice-versa. Let's denote those areas $\dot{R}_{\alpha_i}$ and $\dot{R}_{\alpha_j}$. The LP-switch test, as defined in Section 3.1, verifies if there exists a belief state $b \in R_{\alpha_i}$ such that $S(b) \in R_{\alpha_j}$. Alternatively, a VS-switch test verifies if there exists a belief state $b \in \dot{R}_{\alpha_i}$ such that $S(b) \in \dot{R}_{\alpha_j}$. Below, Theorem 5 shows that when a VS-switch test is negative (that is, there doesn't exist any belief state $b \in \dot{R}_{\alpha_i}$ such that $S(b) \in \dot{R}_{\alpha_j}$), the corresponding LP-switch test is also negative and therefore $\alpha_j \notin Sw(\alpha_i)$. The reverse is not true, so a positive VS-switch test only indicates that α_j *may* be in $Sw(\alpha_i)$.

Theorem 5 *If* $\neg\exists\dot{b} \in \dot{R}_{\alpha_i}$ *such that* $S(\dot{b}) \in \dot{R}_{\alpha_j}$*, then* $\neg\exists b \in R_{\alpha_i}$ *such that* $S(b) \in R_{\alpha_j}$.

Proof Let's show that the converse holds, that is, if $\exists b \in R_{\alpha_i}$ such that $S(b) \in R_{\alpha_j}$, then $\exists\dot{b} \in \dot{R}_{\alpha_i}$ such that $S(\dot{b}) \in \dot{R}_{\alpha_j}$. It should be fairly obvious that $R_{\alpha_i} \subseteq \dot{R}_{\alpha_i}$ since R_{α_i} corresponds to the area where α_i dominates all other vectors in $\aleph$ and $\dot{R}_{\alpha_i}$ correponds to the region where α_i dominates α_j. By the same argument, R_{α_j} is also a subset of $\dot{R}_{\alpha_j}$. Thus, when there is a belief state $b \in R_{\alpha_i}$ such that $S(b) \in R_{\alpha_j}$, then b is also in $\dot{R}_{\alpha_i}$ and $S(b)$ is also in $\dot{R}_{\alpha_j}$ $\square$

The optimization program corresponding to a VS-switch test is given in Table 4.1. This optimization program is essentially the switch test program of Table 3.1

[2] There are 3 states s_1, s_2 and s_3 but only two dimensions are necessary since probability distributions sum up to 1.

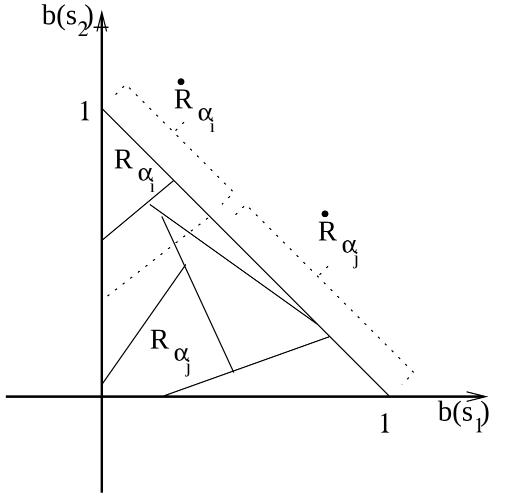


Figure 4.2: Belief space regions for each α-vector

$$
\begin{array}{lll}
\max & x & \\
s.t. & b \cdot (\alpha_i - \alpha_j) \geq x & \\
 & S(b) \cdot (\alpha_j - \alpha_i) \geq x & \\
 & \sum_s b(s) = 1 & \\
 & b(s) \geq 0 & \forall s
\end{array}
$$

Table 4.1: VS-switch test for projection schemes. The optimization program has a strictly positive objective function when there exists a belief state $b \in \dot{R}_{\alpha_i}$ such that $S(b) \in \dot{R}_{\alpha_j}$ and a non-positive objective function where there doesn't exists such a belief state.

with the exception that all the constraints involving α-vectors other than α_i and α_j have been removed. Since projection schemes yield non-linear approximations, we can define a *linear* VS-switch test as in Table 4.2. This LP is a relaxation of the LP switch test of Table 3.2. As demonstrated in Theorem 1 for LP-switch tests, linear VS-switch tests can be used to construct supersets of the switch sets obtained by a non-linear VS-switch tests. For practical reasons, we will consider from now on exclusively the linear version of VS-switch tests and we will refer to them simply as VS-switch tests.

$$\begin{array}{lll}
\max & x & \\
s.t. & b \cdot (\alpha_i - \alpha_j) \geq x & \\
 & b' \cdot (\alpha_j - \alpha_i) \geq x & \\
 & b'(M) = b(M) & \forall M \subseteq M_l,\ 1 \leq l \leq n \\
 & \sum_s b(s) = 1 & \\
 & b(s) \geq 0 & \forall s \\
 & b'(s) \geq 0 & \forall s
\end{array}$$

Table 4.2: Linear VS-switch test for projection schemes. This LP has a strictly positive objective function when there exists belief states $b \in \dot{R}_{\alpha_i}$ and $b' \in \dot{R}_{\alpha_j}$ such that $b(M) = b'(M)$ for all marginals M defined by projection S and a non-positive objective function when there doesn't exist such belief states.

We will now show how a (linear) VS-switch test is equivalent to verifying whether $D_S^{\perp}$ contains α_{ij} or not. More precisely, the VS-switch test is positive when $\alpha_{ij} \notin D_S^{\perp}$ and negative when $\alpha_{ij} \in D_S^{\perp}$. The idea is to consider α_{ij} as a *gradient* that measures the error induced by an approximation when comparing the expected total return of α_i to α_j. Recall that the vector α_{ij} measures the difference in expected total return between α_i and α_j. After an approximation, if this difference changes considerably, the agent is likely to choose the wrong maximizing α-vector. Let's call *relative error* this change in the relative assessment of α_i with respect to α_j. Formally, the relative error is quantified as follows:

$$\begin{aligned}
\text{relative error} &= b(\alpha_i - \alpha_j) - S(b)(\alpha_i - \alpha_j) \\
&= b \cdot \alpha_{ij} - S(b) \cdot \alpha_{ij} \\
&= d \cdot \alpha_{ij} \qquad (4.4)
\end{aligned}$$

Here α_{ij} can be viewed as a gradient since approximations corresponding to a displacement vector d in the same direction as α_{ij} (parrallel to α_{ij}) maximimize the magnitude of the dot product $d \cdot \alpha_{ij}$. In general, the angle between d and α_{ij} is a good indicator of approximation error. In particular, if they are perpendicular, their dot product is zero and the relative assessment of α_i and α_j remains unchanged, preventing any switch. By definition, the subspace $D_S^{\perp}$ is the set of vectors perpendicular to all displacement vectors possibly induced by S, so when α_{ij} is a member of $D_S^\perp$, all possible displacement vectors are perpendicular to α_{ij} and consequently there cannot be a switch from α_i to α_j. Below, Theorem 6 gives a formal proof of the equivalence between the VS-switch test and the subspace membership test "is $\alpha_{ij} \in D_S^\perp$?".

Theorem 6 *There exists no $b \in \dot{R}_{\alpha_i}, b' \in \dot{R}_{\alpha_j}$ such that $b'(M) = b(M)$ for all marginals M defined by S iff $\alpha_{ij} \in D_S^\perp$.*

Proof First, suppose that $\exists b \in \dot{R}_{\alpha_i}, b' \in \dot{R}_{\alpha_j}$ such that $b'(M) = b(M)$ for all marginals M defined by S, let's show that $\alpha_{ij} \notin D_S^\perp$. Since $b \in \dot{R}_{\alpha_i}$ then $b \cdot (\alpha_i - \alpha_j) > 0$. Similarly, because $b' \in \dot{R}_{\alpha_j}$ it follows that $-b' \cdot (\alpha_i - \alpha_j) > 0$. If we add those two inequalities together, we get $(b - b') \cdot (\alpha_i - \alpha_j) = d \cdot \alpha_{ij} > 0$. By definition, $\alpha_{ij} \in D_S^\perp$ iff $d \cdot \alpha_{ij} = 0 \; \forall d \in D_S$, so α_{ij} is not an element of $D_S^\perp$. Next, suppose that $\alpha_{ij} \notin D_S^\perp$, then let's show that $\exists b \in \dot{R}_{\alpha_i}, b' \in \dot{R}_{\alpha_j}$ such that $b'(M) = b(M)$ for all marginals M defined by S. Since $\alpha_{ij} \notin D_S^\perp$, then there exists a displacement vector $d \in D_S$ such that $d \cdot \alpha_{ij} \neq 0$. WLOG let $d \cdot \alpha_{ij} > 0$.[3] For any sufficiently small vector $d \in D_S$, there exist $b \in \dot{R}_{\alpha_i}, b' \in \dot{R}_{\alpha_j}$ such that $d = b - b'$. Finally, since $d \in D_S$, then by definition $d(M) = 0 \; \forall M$ and consequently $b(M) = b'(M) \; \forall M$. □

Although we could use the LP in Table 4.2 to carry out VS-switch tests, there is a much more efficient method based on the subspace membership criteria. The idea is to decompose α_{ij} in two orthogonal vectors corresponding to the projections of α_{ij} on $D_S^\perp$ and D_S. Below, $proj(\alpha, D)$ stands for the projection of the vector α on the subspace D.

$$\alpha_{ij} = proj(\alpha_{ij}, D_S^\perp) + proj(\alpha_{ij}, D_S) \tag{4.5}$$

If $\alpha_{ij} \in D_S^\perp$, then $proj(\alpha_{ij}, D_S^\perp) = \alpha_{ij}$ and consequently $proj(\alpha_{ij}, D_S)$ is the zero vector; otherwise, $proj(\alpha_{ij}, D_S^\perp) \neq \alpha_{ij}$ and consequently $proj(\alpha_{ij}, D_S)$

[3] If $d \cdot \alpha_{ij} < 0$, then pick $d' = -d$ since $-d \cdot \alpha_{ij} > 0$.

is a non zero vector. Based on this observation we can determine if $\alpha_{ij} \in D_S^{\perp}$ by measuring the length of $proj(\alpha_{ij}, D_S)$: $\|proj(\alpha_{ij}, D_S)\|_2 = 0$ when $\alpha_{ij} \in D_S^{\perp}$, and $\|proj(\alpha_{ij}, D_S)\|_2 > 0$ when $\alpha_{ij} \notin D_S^{\perp}$. In particular, the squared length of $proj(\alpha_{ij}, D_S)$ can be computed by the following equation:

$$\|proj(\alpha_{ij}, D_S)\|_2^2 = \alpha_{ij} \cdot \alpha_{ij} - \sum_{v \in \mathcal{D}_S^{\perp}} (\alpha_{ij} \cdot v)^2 \tag{4.6}$$

Here $\mathcal{D}_S^{\perp}$ is an orthonormal basis spanning the subspace $D_S^{\perp}$. Theorem 7 shows how to derive the above expression.

Theorem 7 *Let $\mathcal{D}_S^{\perp}$ be an orthonormal basis spanning the subspace $D_S^{\perp}$, then*

$$\|proj(\alpha_{ij}, D_S)\|_2^2 = \alpha_{ij} \cdot \alpha_{ij} - \sum_{v \in \mathcal{D}_S^{\perp}} (\alpha_{ij} \cdot v)^2$$

Proof

$$\begin{aligned}
\|proj(\alpha_{ij}, D_S)\|_2^2 &= \|\alpha_{ij}\|_2^2 - \|proj(\alpha_{ij}, D_S^{\perp})\|_2^2 && (4.7)\\
&= \alpha_{ij} \cdot \alpha_{ij} - \|\sum_{v \in \mathcal{D}_S^{\perp}} proj(\alpha_{ij}, v)\|_2^2 && (4.8)\\
&= \alpha_{ij} \cdot \alpha_{ij} - \sum_{v \in \mathcal{D}_S^{\perp}} \|proj(\alpha_{ij}, v)\|_2^2 && (4.9)\\
&= \alpha_{ij} \cdot \alpha_{ij} - \sum_{v \in \mathcal{D}_S^{\perp}} (\alpha_{ij} \cdot v)^2 && (4.10)
\end{aligned}$$

Equation 4.7 follows since D_S and $D_S^{\perp}$ are complementary orthogonal subspaces. In Equation 4.8, a projection on a subspace is equivalent to the sum of the projections on each vector of an orthonormal basis of this subspace. In Equation 4.9, the squared length of a sum of orthogonal vectors is the sum of the squared length of each othogonal vector. Finally, in Equation 4.10, the squared length of the projection of α_{ij} on some unit vector v is by definition the squared dot product of α_{ij} and v. □

In order to compute expression 4.6, we need an orthonormal basis that spans the subspace $D_S^{\perp}$. An orthonormal basis is a basis with two special properties: all its vectors are normal (length equal to one) and pairwise orthogonal (perpendicular to each other). In general, any spanning set of vectors can be transformed into an orthonormal basis by applying the Gram-Schmidt orthogonalization process and normalizing the resulting vectors. The subspace $D_S^\perp$ was originally defined by the spanning set of vectors v_M corresponding to the constraint for each marginal M, hence this set can be used to generate several orthonormal bases. In this thesis, we shall consider one orthonormal basis in particular—which we will refer to as $\mathcal{D}_S^\perp$—because of its factored representation. For problems involving binary variables, every vector in our choice of $\mathcal{D}_S^\perp$ consists of a sequence of 1's and -1's (before normalization). The basis vector $\bar{v}_M$ associated with subset M has a 1 in every component corresponding to a state with an even number of true variables in M and -1 in every component corresponding to a state with an odd number of true variables in M. The vector is then normalized by dividing by $\sqrt{|\mathcal{S}|}$. For instance, projection scheme $S = \{XY, YZ\}$ has six marginals ($\emptyset$, X, Y, Z, XY and YZ), yielding the following basis vectors:[4]

$$\begin{array}{ccccccccccccc}
 & & & xyz & xy\bar{z} & x\bar{y}z & x\bar{y}\bar{z} & \bar{x}yz & \bar{x}y\bar{z} & \bar{x}\bar{y}z & \bar{x}\bar{y}\bar{z} & & \\
\bar{v}_\emptyset & = & (& 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 &) & / \ \sqrt{|\mathcal{S}|} \\
\bar{v}_X & = & (& -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 &) & / \ \sqrt{|\mathcal{S}|} \\
\bar{v}_Y & = & (& -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 &) & / \ \sqrt{|\mathcal{S}|} \\
\bar{v}_Z & = & (& -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 &) & / \ \sqrt{|\mathcal{S}|} \\
\bar{v}_{XY} & = & (& 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 &) & / \ \sqrt{|\mathcal{S}|} \\
\bar{v}_{YZ} & = & (& 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 &) & / \ \sqrt{|\mathcal{S}|}
\end{array}$$

We now have all the necessary information to carry out very efficient VS-switch tests. The running time of such switch tests is much lower than that of LP-switch tests. A VS-switch test boils down to the computation of Expression 4.6 which requires O(c) dot products. In turn, each dot product requires $O(|\mathcal{S}|)$ elementary operations, for a total running time of $O(c|\mathcal{S}|)$. The use of factored repre-

[4] This definition of $\mathcal{D}_S^\perp$ can be generalized to non-binary variables.

sentations such as ADDs improves considerably this running time. More precisely, each basis vector has components that take only two values, yielding a very compact ADD representation. The number of operations required to compute the dot product of two factored vectors is essentially linear in the size of the intersection of their ADD representation. The basis vectors have a very small ADD representation since every state takes one of two possible values. Similarly, the vector α_{ij} also has a small ADD representation since we assume that the POMDP is fairly structured. Thus, the number of operations for a dot product is often a small constant independent of the size of the state space. Hence, for sufficiently structured POMDPs, the effective running time of a VS-switch test tends to be $O(c)$.

In comparison, solving the linear program of an LP-switch test is polynomial in the number of constraints c and the size of the state space. Furthermore, ADDs do not provide a speed up as important for LPs since the effective state space is the intersection of the abstract state space of all the constraints instead of only two vectors for dot products. In particular, as explained in Section 3.1, LP-switch tests for projection schemes do not benefit at all from ADDs. Hence, VS-switch tests provide a much more efficient and practical method to compute switch sets. On the other hand, this efficiency gain results from the use of relaxed constraints, which means that the switch set computed is really a superset. In turn, this has an impact on the quality of the B, U and E bounds since they will be looser, however the algorithms to compute them remain unchanged.

The running time of the greedy lattice search when using VS-switch tests is significantly improved. With LP-switch tests, it was $O(nc^2|\aleph|)$ LPs (or $O(nc^{2+k}|\aleph||\mathcal{S}|^k)$ elementary operations, assuming an LP requires a number of operations that is a polynomial of degree k in terms of c and $|\mathcal{S}|$) whereas with VS switch tests, it is $O(nc^3|\aleph||\mathcal{S}|)$ elementary operations. Note here that the upper bounds on running times are given in terms of the size of the full state space $|\mathcal{S}|$ since there are problems for which factored representations do not provide any savings, however, the reader should keep in mind that in practice, many POMDPs are structured and allow ADDs (for instance) to reduce their effective state space considerably (possibly linear in the number of variables instead of the number of states).

4.3 Vector space search

In this section, we describe an alternative search method based on the relative error (expression 4.6). We call it the vector-space search or in short the VS search. It differs from the previous B-bound search and E-bound search in that it doesn't try to minimize an error upper bound. The fact is that in practice, minimizing the *worst* possible error doesn't necessarily lead to good *average* results.

The basic idea behind the VS search is rooted in the observation that the more perpendicular is the direction of an approximation with respect to α_{ij}, the smaller is the magnitude of the relative error and consequently, the less likely a switch will occur. Thus, the VS search seeks a projection S which defines a displacement subspace D_S that is as perpendicular as possible to all gradients α_{ij}. Technically, this is done by minimizing the squared length of the projection of each gradient α_{ij} on D_S (as in Equation 4.6).

It is interesting to note that the length of $proj(\alpha_{ij}, D_S)$ has a special interpretation: it corresponds to the greatest (absolute) *relative error rate* for an approximation in some direction $d \in D_S$. The relative error rate for an approximation corresponding to the displacement vector d, is given by the relative error induced by a *unit* displacement in the direction of d:

$$\frac{d}{||d||_2} \cdot \alpha_{ij}$$

By definition, the projection of a vector α on some subspace D is:

$$||proj(\alpha, D)||_2 = \sup_{d \in D} |\frac{d}{||d||_2} \cdot \alpha|$$

Hence, by minimizing expression 4.6, we are essentially minimizing the (squared) worst relative error rate that may occur as a result of some projection S. When ignoring the distance between the exact and approximate belief states, the relative error rate gives us a handle to quantify how bad an approximation in some direction is likely to be. Each projection S constrains approximations to directions within the subspace D_S. The direction $d \in D_S$ with the highest (absolute) relative error rate is this worst relative error rate which also happens to be $\|proj(\alpha_{ij}, D_S)\|_2$. Thus, it makes sense to try to minimize expression 4.6.

When searching for a good projection S for belief space region R_{α_i}, there are $|\aleph| - 1$ α-vectors we would like to prevent switching to. Ideally this would be accomplished by minimizing simultaneously expression 4.6 for every gradient α_{ij} ($j \neq i$). In the absence of any prior concerning the relative importance of each gradient, we suggest two simple schemes:

- Minimize the sum of the squared length of each projection:

$$\sum_{j \neq i} \|proj(\alpha_{ij}, D_S)\|_2^2 = \sum_{j \neq i} (\alpha_{ij} \cdot \alpha_{ij} - \sum_{v \in D_S^\perp} v \cdot \alpha_{ij}) \qquad (4.11)$$

- Minimize the squared length of the greatest projection:

$$\max_{j \neq i} \ \|proj(\alpha_{ij}, D_S)\|_2^2 = \max_{j \neq i} \ (\alpha_{ij} \cdot \alpha_{ij} - \sum_{v \in D_S^\perp} v \cdot \alpha_{ij}) \qquad (4.12)$$

Of course, many other schemes could be proposed and if the practitionner has some prior knowledge, it should definitely be used.

Given a vector $\alpha_i \in \aleph$, the VS search finds a good projection S as follows. Starting at the root, traverse the lattice downwards in a greedy manner. At each node, pick the most promising child by minimizing Equation 4.11 or Equation 4.12. The computational complexity of a VS search is fairly low as it avoids LPs. Its running time is $O(nc^3|\aleph|^2|\mathcal{S}|)$, since one good projection must be found for each of the $|\aleph|$ regions R_α. For each region, $O(nc^2)$ nodes in the lattice are traversed, each requiring the evaluation of Equation 4.11 or 4.12 which both take $O(c|\aleph||\mathcal{S}|)$ elementary operations.

As with the B-bound and E-bound search procedures, the VS search can also be streamlined but in a different way. The constraints of a node S are essentially the same as the constraints of its parent node S' with one extra constraint corresponding to the marginal M that labels the edge connecting the two nodes. Since there is one basis vector per constraint, the following equation holds:

$$\mathcal{D}_S^\perp = \mathcal{D}_{S'}^\perp \cup \{\bar{v}_M\}$$

In turn this means that Equations 4.11 and 4.12 can be computed incrementally as the lattice is traversed downwards:

$$\begin{aligned}\sum_{j \neq i} \|proj(\alpha_{ij}, D_S)\|_2^2 &= \sum_{j \neq i} \|proj(\alpha_{ij}, D_{S'})\|_2^2 - \bar{v}_M \cdot \alpha_{ij} \\ \max_{j \neq i} \|proj(\alpha_{ij}, D_S)\|_2^2 &= \max_{j \neq i} \|proj(\alpha_{ij}, D_{S'})\|_2^2 - \bar{v}_M \cdot \alpha_{ij}\end{aligned}$$

Incremental computation reduces the running time to $O(nc^2|\aleph|^2|\mathcal{S}|)$ since only one dot product needs to be computed instead of one for each of the c constraints. This running time is significantly smaller than $O(nc^{2+k}|\aleph||\mathcal{S}|^k)$ for the B-bound or E-bound greedy search with LP-switch tests. As for the B-bound or E-bound greedy search with VS-switch tests, the running time $O(nc^3|\aleph||\mathcal{S}|)$ is comparable. The VS search has an extra $|\aleph|$ factor, but one less c factor. In practice, $|\aleph|$ is usually larger than c, so the VS search is actually slower. Again, the reader should keep in mind that the upper bounds on running times are given in terms of $|\mathcal{S}|$, but in practice, factored representation can drastically reduce the size of the effective state space for structured POMDPs.

Chapter 5

Empirical Evaluation

This chapter evaluates empirically the algorithms proposed so far. First, in Section 5.1, the factory example introduced in Chapter 1 is revisited with specific transition probabilities and reward functions. We show how a value-directed approach yields a better choice of projection schemes than distance based approaches that try to minimize the KL-divergence, L_1 or L_2 norms. The experiments carried out with this factory example are very simple and serve only as a proof of concept.

In Section 5.2, more extensive experiments are carried out to evaluate the quality of the approximation algorithms as well as their running time. The reader should be warned however that those experiments are still limited in scope and complexity, and may not be representative of typical real world problems. This is because we are not yet in a position to solve large interesting POMDPs. Therefore the three problems used to carry out the experiments are simple and fairly structured so that the number of α-vectors representing a value function remains manageable and the representation of each α-vector can be factored and made compact. The goal of these experiments is threefold:

- Compare the running time of different search algorithms.
- Compare the B, U and E error bounds, which measure the expected loss in a *worst* case scenario, to the *average* loss realized for 5000 random policy

executions.

- Compare the average loss due to projections obtained by different search algorithms.

Note that those experiments do not evaluate the belief state monitoring speed up obtained by the application of different projection schemes. This is because the focus of this thesis is on approximation quality which had not been investigated thoroughly in the literature. The reader is referred to Boyen and Koller [3] for experiments illustrating the speed up obtained by projection schemes.

5.1 Proof of Concept

Let's revisit the factory problem described in Section 1.2 to illustrate the benefits of value-directed approximation. The aim is to demonstrate that minimizing belief state error as measured by KL-divergence, L_1 or L_2 norms is not always appropriate when approximate monitoring is used to implement an optimal policy. Below, the factory problem is made concrete through a process involving seven stages, with only one or two actions per stage (thus at some stages no choice needs to be made), and no observations. Yet, even such a simple system shows the benefits of allowing the value function to influence the choice of approximation schemes.

We suppose there is a seven-stage manufacturing process whereby four parts are produced using three machines, M, *M1*, and *M2*. Parts *P1*, *P2*, *P3*, and *P4* are each stamped in turn by machine M. Once stamped, parts *P1* and *P2* are processed separately (one after the other) on machine *M1*, while parts *P3* and *P4* are processed together on *M2*. Machine M may be faulty (*FM*), with prior probability $\Pr(FM)$. When the parts are stamped by M, parts *P1* and *P2* may become faulty (*F1*, *F2*), with higher probability of fault if *FM* holds. Parts *P3* and *P4* may also become faulty (*F3*, *F4*), again with higher probability if *FM*; but *F3* and *F4* are both less sensitive to *FM* than *F1* and *F2* (e.g., $\Pr(F1|FM) = \Pr(F2|FM) > \Pr(F3|FM) = \Pr(F4|FM)$).

Stages to go	Actions	Transitions	Rewards
7) Stamp P1	Stamp P1	only affects F1 if FM at previous step then Pr(F1) = 0.8 else Pr(F1) = 0.1	no reward
6) Stamp P2	Stamp P2	only affects F2 if FM at previous step then Pr(F2) = 0.8 else Pr(F2) = 0.1	no reward
5) Stamp P3	Stamp P3	only affects F3: if FM at previous step then Pr(F3) = 0.1 else Pr(F3) = 0.05	no reward
4) Stamp P4	Stamp P4	only affects F4: if FM at previous step then Pr(F4) = 0.1 else Pr(F4) = 0.05	no reward
3) Process/Reject P1	Process P1	all variables are persistant	if F1 then 0 else 8
	Reject P1	all variables are persistant	4 for every state
2) Process/Reject P2	Process P2	all variables are persistant	if F2 then 0 else 8
	Reject P2	all variables are persistant	4 for every state

1) Process/Reject P3,P4	Process P3,P4	all variables are persistant	if F3 & F4 then -2000
			if ~F3 & ~F4 then 16
			otherwise 8

	Reject P3,P4	all variables are persistant	3.3 for every state

Table 5.1: POMDP specifications for the factory example

If *P1* or *P2* are processed on machine *M1* when faulty, a cost is incurred; if processed when OK, a gain is had; if not processed (rejected), no cost or gain is had. When *P3* and *P4* are processed (jointly) on $M3$, a greater gain is had if both parts are OK, a lesser gain is had when one part is OK, and a drastic cost is incurred if both parts are faulty (e.g., machine $M3$ is destroyed). The specific problem parameters are given in Table 5.1.

Figure 5.1 shows the dependencies between variables for the seven-stage DBN of the example.[1] It is clear with three stages to go that all the variables are correlated. If approximate belief state monitoring is required for execution of the optimal

[1] We have imposed certain constraints on actions to keep the problem simple; with the addition of several variables, the problem could easily be formulated as a "true" DBN with identical dynamics and action choices at each time slice.

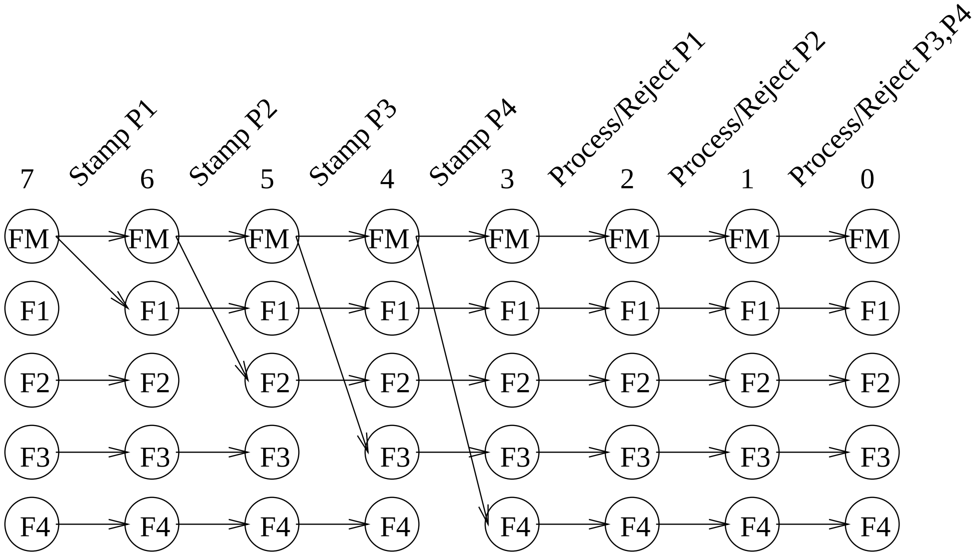


Figure 5.1: DBN for the factory example

policy (admittedly unlikely for such a simple problem!), a suitable projection scheme could be used.

Notice that the decisions to process *P1* and *P2* for 3 or 2 stages-to-go are independent: they depend only on Pr(*F1*) and Pr(*F2*), respectively, but not on the correlation between the two variables. Thus, though these become quite strongly correlated with five stages to go, this correlation can be ignored without any impact on the decision one would make at those points. Conversely, *F3* and *F4* become much more weakly correlated with three stages to go; but the optimal decision at the final stage *does* depend on their joint probability. Were we to ignore this weak correlation, we run the risk of acting suboptimally.

We ran the B-bound search algorithm (with LP-switch tests) of Section 3.3 and, as expected, it suggested projection schemes that break all correlations except for *FM* and *F3* with four stages to go, and *F3* and *F4* with three, two, and one stage(s) to go. The latter, $\Pr(F3, F4)$, is clearly needed (at least for certain prior probabilities on *FM*) to make the correct decision at the final stage; and the former, $\Pr(FM, F3)$, is needed to accurately assess $\Pr(F3, F4)$ at the subsequent stage. Thus

Correlation	L_1	L_2	KL	Loss
F1/*F2*	0.7704	0.3092	0.4325	1.0
F3/*F4*	0.9451	0.3442	0.5599	0.0

Table 5.2: Comparison of different distance measures

we maintain an approximate belief state with marginals involving no more than two variables, yet we are assured of acting optimally.

In contrast, if one chooses a projection scheme for this problem by minimizing KL-divergence, L_1-distance, or L_2-distance, different correlations will generally be preserved. For instance, assuming a uniform prior over *FM* (i.e., machine M is faulty with probability 0.5), Table 5.2 shows the approximation error that is incurred according to each such measure when only the correlation between *F1* and *F2* is maintained or when only the correlation between *F3* and *F4* is maintained. All of these "direct" measures of belief state error prefer the former. However, the loss in expected value due to the former belief state approximation is 1.0, whereas no loss is incurred using the latter. To test this further, we also compared the approximation preferred using these measures over 1000 (uniformly) randomly-generated prior distributions. If only the *F1*/*F2*-correlation is preserved with one stage to go, then in 520 instances a non-optimal action is executed with an average loss of 0.6858. This clearly demonstrates the potential advantage of using a value-directed method to choose good approximation schemes.

5.2 Experiments

Three test problems were used to carry out the more extensive experiments. The first POMDP is essentially the coffee problem introduced by Boutilier and Poole [2]. The second POMDP is a variation of the widget problem described by Draper, Hanks and Weld [10]. The third POMDP is inspired from the pavement maintenance problem described by Puterman [32]. Since the analysis of the experiments doesn't require

Problem	State Space Size		Size of ℵ		Solution Time
	full	effective	maximum	average	(sec)
Coffee	32	12	102	56	47
Widget	32	14	205	121	397
Pavement	128	85	39	16	250

Table 5.3: Solution statistics for the three test problems

any specific domain knowledge, the exact specifications of these three problems can be found in Appendix B.

Before getting into the details of the experiments, it is important to point out a few statistics regarding each test problem. Table 5.3 shows the solution time to find an optimal solution for a discounted finite-horizon of 15 steps. In fact, all the experiments that follow assume a finite discounted horizon of 15 steps (unless otherwise stated). In the table are also included the size of the state space as well as the average and maximum number of α-vectors necessary to represent the optimal value function at each of the 15 stages. We remind the reader that the size of the state space and the number of α-vectors are critical features of a POMDP that have a direct impact on the solution time, as well as the algorithms proposed in this thesis. The state space column is divided in two: the *full* state space corresponds to $|\aleph|$ and the *effective* state space is the largest intersection of abstract state spaces encountered in an LP-dominance test. Since all algorithms were implemented using ADDs, their running time depends on the size of the effective state space rather than the full state space, with one exception, LP-switch tests. As explained in Section 3.1.1, LP-switch tests do not benefit from ADDs since the intersection of all abstract state spaces (corresponding to ADD representations of the constraints defined by a projection scheme) is the full state space.

Problem	Solution Time	B-bound search		E-bound search		VS search	
		LP	VS	LP	VS	max	sum
	(sec)	(sec)	(sec)	(sec)	(sec)	(sec)	(sec)
Coffee	47	1019	30	4379	2651	151	154
widget	397	10142	109	89605	48695	707	703
Pavement	250	345	35	841	126	97	96

Table 5.4: Search running time

5.2.1 Search running time

The first experiment compares the time required to search for good projection schemes by minimizing different error bounds. Table 5.4 shows the timing of all search algorithms proposed in Chapters 3 and 4 to find a good projection for each α-vector in all the optimal value functions of the first 15 stages. All search algorithms perform a lattice search within the set of projection schemes that partition variables in disjoint subsets. This is because projection schemes with overlapping subsets of variables may or may not be practical depending on whether they allow the construction of a clique tree. The clique tree enables an explicit representation of the belief state which is often necessary for Bayesian inference. Since the focus of this thesis is on approximation quality and due to time constraints, the Bayes net infrastructure for detecting projection schemes that do not allow the construction of a clique tree was not implemented. In principle, there is no technical reason that should prevent such an implementation. Hence, the search is limited to projection schemes with disjoint subsets. Furthermore, assuming that marginals of at most two variables provide a suitable efficiency/accuracy tradeoff, the lattice is traversed until all children of a node correspond to projection schemes with a marginal of more than two variables. This last node is the projection scheme returned by the search. We should point out that in all the experiments to follow, the search algorithms are restricted to projection schemes with marginals of at most two variables (unless otherwise stated).

As a reminder, the B-bound search traverses the lattice greedily by minimizing the worst error induced by a single approximation. Switching can be determined by computing LP-switch tests (Section 3.1.1) or VS-switch tests (Section 4.2). In Table 5.4, as expected, the running time when using VS-switch tests is much less since there are no LPs to be solved. Furthermore, as mentioned above, LP-switch tests are the only algorithms that cannot benefit from the use of compact representations and therefore have a running time dependent on the size of the full state space instead of the effective state space. Note that a U-bound search is essentially the same as a B-bound search and therefore the running times are identical. A U-bound is computed by adding the E-bounds of the current and previous stages. The time required for this sum is a negligible constant when compared to the search itself.

As for the E-bound search, it also traverses the lattice greedily, but by minimizing the cumulative error induced by consecutive approximations. Hence, in addition to the lattice traversal, it constructs switch sets and *Alt*-sets. This explains why the running time for an E-bound search is much longer than for a B-bound search. Once again, switch tests in the E-bound search can be performed using LPs or the vector space formulation and as expected the latter is significantly faster than the former.

Finally, the VS search consists of a greedy traversal of the lattice whereby the relative error rate is minimized. That is, the gradients α_{ij} are made as perpendicular as possible to the subspace of displacement vectors D. Since there are several gradients to take into consideration, we can minimize (among other things) the *sum* of all relative error rates or their *maximum*. For either criteria (sum vs max), the running time is roughly the same and it is significantly faster than B-bound and E-bound search algorithms that use LP-switch tests, but a bit slower if VS-switch tests are used for the B-bound search. This is because, on one hand, the VS search does not solve any LP (when compared to LP-switch tests), but on the other hand, it has a stronger dependence on the number of α-vectors (when compared to VS-switch tests).

An important observation regarding this experiment is that the time to search for good projections can be much worse than that of solving POMDPs, which we know is already intractable. In fact, only the search procedures that avoid solving LPs scale well when larger problems need to be tackled. The VS procedures have a running time that is roughly of the same order of magnitude as the solution procedures.

5.2.2 Error bound evaluation

In the second experiment, the B, U and E bounds are compared to the average error incurred in 5000 random policy executions. Tables 5.5, 5.6 and 5.7 show the results of the experiment for each test problem. In each table there is one column per search algorithm and one row per error bound (or error average). All entries in a table are interpreted as an error bound or an error average (indicated by the row label) for a given search algorithm (indicated by the column label). The first two rows evaluate the error due to a single approximation at 15 stages to go. The average error is the average loss incurred for 5000 random initial belief states generated from a uniform distribution. Note that the same 5000 samples were used to evaluate all search algorithms in all tables. The last three rows evaluate the cumulative error due to several approximations during the execution of a 15-stage policy. The average error denotes the average cumulative loss for those same 5000 uniformly random initial belief states.

Looking at the tables, it is striking how large the B, U and E bounds are compared to the average error observed. This is explained in part by the fact that the bounds are concerned with the worst case scenario whereas the errors obtained through sampling are averaged out. The second reason is that the bounds are not tight. As mentioned in Section 3.1, the switch sets computed when dealing with

	Error	B-bound search		E-bound search		VS search	
		LP	VS	LP	VS	max	sum
Single	Average	0.001301	0.006278	0.006278	0.006278	0.001336	0.001349
Approx.	B-bound	3.284000	5.915000	4.391000	5.915000	3.284000	3.284000
Several	Average	0.014351	0.016113	0.016113	0.016113	0.015433	0.010733
Approx.	U-bound	24.85	42.47	32.60	42.47	24.85	24.85
	E-bound	13.084810	13.084810	13.084810	13.084810	13.084810	13.084810

Table 5.5: Coffee problem: error bound comparisons at 15 stages to go

	Error	B-bound search		E-bound search		VS search	
		LP	VS	LP	VS	max	sum
Single	Average	0.035180	0.035193	0.035193	0.035193	0.008152	0.008144
Approx.	B-bound	3.408000	3.627000	3.408000	3.627000	3.408000	3.408000
Several	Average	0.050858	0.050818	0.050818	0.050818	0.051912	0.051719
Approx.	U-bound	34.93	37.42	34.95	37.42	34.95	34.93
	E-bound	8.381117	8.381117	8.381117	8.381117	8.381117	8.381117

Table 5.6: Widget problem: error bound comparisons at 15 stages to go

	Error	B-bound search		E-bound search		VS search	
		LP	VS	LP	VS	max	sum
Single	Average	0.001510	0.001510	0.001510	0.001510	0.001417	0.001415
Approx.	B-bound	5.386000	5.690000	5.386000	5.690000	5.368000	5.616000
Several	Average	0.006640	0.006640	0.006640	0.006640	0.007101	0.002753
Approx.	U-bound	38.57	40.80	38.87	40.80	38.99	39.52
	E-bound	23.217647	35.392262	23.497992	35.392262	23.873747	24.384092

Table 5.7: Pavement problem: error bound comparisons at 15 stages to go

projection schemes (nonlinear approximations) are really supersets of the switch set. Switch sets are an integral part of the computation of the B, U and E bounds, meaning that they are all potentially loose. Similarly, when VS-switch tests are used instead of LP-switch tests, a potentially larger superset of the switch set is constructed. Moreover, the U bound is loose by definition, since it simply adds up the B bounds. We introduced the E bound as a tighter alternative to the U bound, however it is also loose as it relies on *Alt*-sets which are really supersets of the set of all alternative plans.

The large difference between bounds and sampled averages suggests two things:

- There may not be any point in computing those error bounds, since in practice it is very unlikely to suffer a loss as large as indicated by the bounds. Although the bounds currently appear meaningless, their relative scale may still provide useful information for choosing a good projection. It would also be interesting to see how the bounds compare to the average loss when their looseness is contained. For instance, linear approximation methods allow the computation of exact switch sets, which could significantly improve the bounds' tightness. Finally, as future work, one should consider improving the E bound by constructing *Alt*-sets as close as possible to the actual set of alternative plans.

- Search algorithms that try to minimize an error bound may not be appropriate in practice. On the other hand, all search algorithms seem to perform equally well since the error bounds and the error averages are all similar. The third experiment (next section) will shed some light regarding this observation.

5.2.3 Search algorithm comparison

In the third experiment, we compare the projection schemes found by different search algorithms. Interestingly enough, this comparison reveals that most of the projections found by algorithms that minimize a bound (i.e., B-bound search with LP or VS-switch tests and E-bound search with LP or VS-switch tests) all yield very similar (if not identical) projections. There is an explanation to this. When traversing the lattice, those algorithms always select the child with the smallest bound (B or E). However, when all children have the same bound, those algorithms all break ties in the same way, by simply choosing the first child (of some predetermined ordering). It turns out that for most problems, the nodes at the top of the lattice correspond to very fine projection schemes (that preserve few correlations) and therefore allowing switching to most if not all α-vectors representing the value function. As defined in Sections 3.1-3.2, the B and E bounds correspond respectively to the single and cumulative error associated with the α-vector in the switch set that contributes the largest such error. Hence, when all the children of a node allow switching to most or all α-vectors, the α-vector with the largest single or cumulative error is likely to be in the switch set of all children, meaning that they all have the same B or E bound. Thus at the beginning of any B or E-bound search, the search is still progressing within the top part of the lattice, yielding roughly the same projections regardless of which bound is minimized. Note that this phenomena is amplified by the fact that supersets of the switch sets are computed, and this is even worse when VS-switch tests are used. A possible solution to this problem would be to start the search at some node further down the lattice. For example, one could heuristically pick as a starting node, a projection scheme that would preserve certain correlations that appear important when examining the Bayes net dynamics and the reward functions.

As for VS search algorithms, they do not suffer from the above problem. This is because they minimize the maximum relative error rate or the squared sum of all relative error rates. These quantities are very sensitive to the marginals defined by different projection schemes. From a linear algebra point of view, the marginals of each projection scheme defines a subspace D of displacement vectors. The angles between this subspace and each gradient α_{ij} determine the relative error rates. Since those angles change with each projection scheme, ties between children rarely occur and the lattice traversal for each VS search tends to be different.

Figures 5.2, 5.3 and 5.4 compare the average cumulative error for the max VS search (minimize *maximum* relative error rate), the sum VS search (minimize *sum* of squared relative error rates) and the B-bound search with LP-switch tests. The other B-bound and E-bound search algorithms were not included in the comparisons as they yield very similar or identical projection schemes than the B-bound search with LP-switch tests. The graphs indicate the average cumulative loss in terms of the number of stages to go for the same 5000 policy executions (starting in different initial belief states chosen uniformly randomly) that were used in the experiment of the previous section. In theory, error bars cannot be negative, however, because the initial belief states were sampled randomly, it may happen that the average discounted cumulative reward is higher than the optimal expected total reward. Furthermore, error bars do not necessarily increase monotonically with the number of stages to go in general. When a switch occurs, it usually is to a plan of lower expected total reward. However, if the current plan is not optimal (due to previous approximations), a switch may be beneficial as it may be to a plan of higher expected total reward, essentially diminishing the error caused by previous approximations. On the other hand, as shown by the graphs, the average cumulative error tends to increase with the number of stages to go. Since POMDPs are assumed to have finite reward functions, this error eventually stops growing. More experiments would be required to show when the error levels off in practice.

In Figure 5.4, the graph for the pavement problem indicates a sudden large cumulative error at 7 stages to go for the B-bound search and at 8 stages to go for the max VS search. After those stages, the error diminishes for a few stages and starts growing again. This intriguing behavior is easily explained by Figure 5.5 which shows the error due to a single approximation at each stage. It turns out

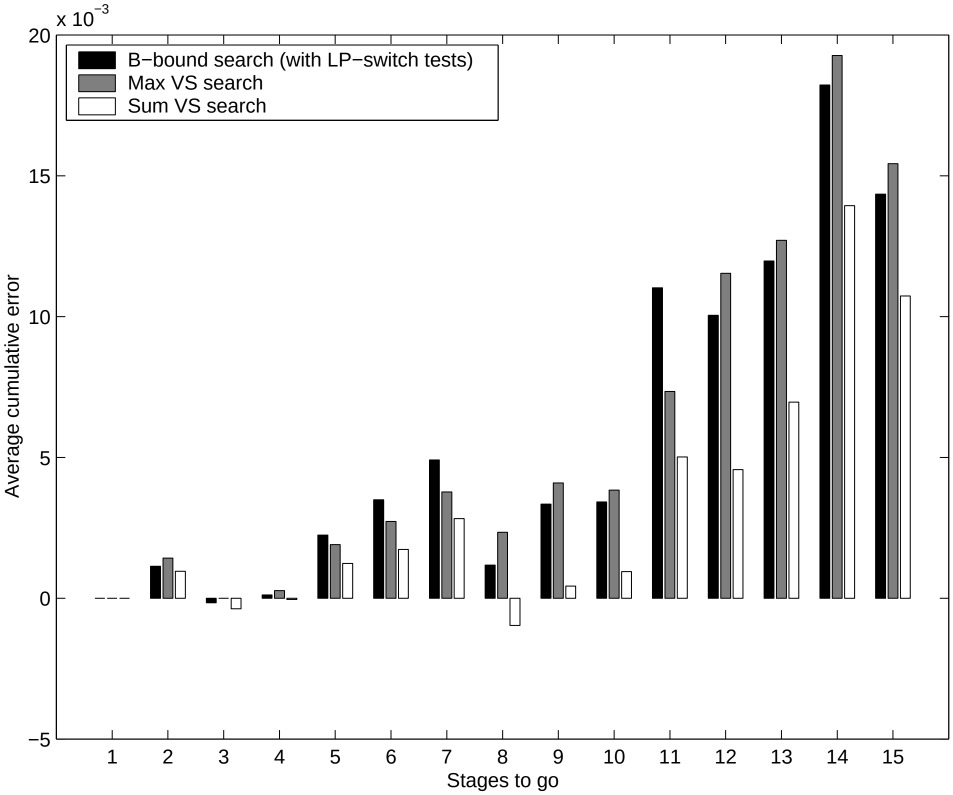


Figure 5.2: Coffee problem: comparison of the average cumulative error obtained by different search algorithms

that a large error is incurred at 7 and 8 stages to go for the B-bound search and the max VS search algorithms, which explains the sudden jump in cumulative error at those stages in Figure 5.4. Then, for a few stages, approximations do not yield any error, so the cumulative error naturally decreases due to the discount factor. It starts growing again at 13 stages to go when some errors start appearing.

For the coffee and pavement problems, the cumulative error bars corresponding to the sum VS search are lower than those corresponding to the max VS search and the B-bound search, whereas for the widget problem, the B-bound search algorithm provides the smallest cumulative errors. From those graphs it is not possible to decide which search method is likely to perform better than the others in general.

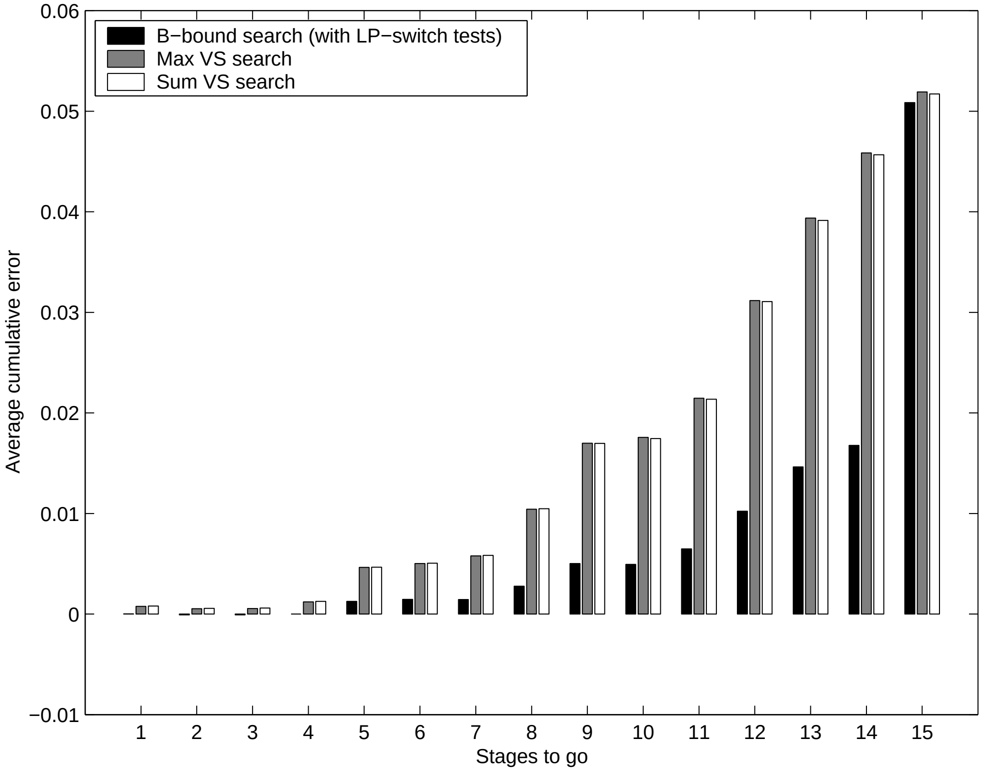


Figure 5.3: Widget problem: comparison of the average cumulative error obtained by different search algorithms

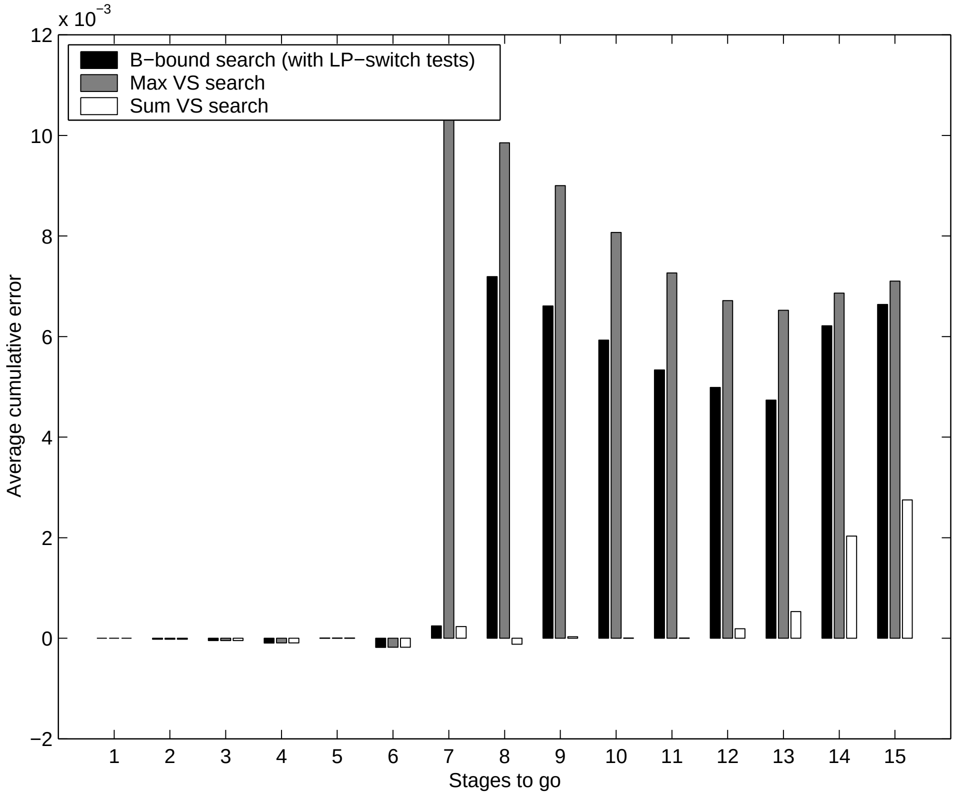


Figure 5.4: Pavement problem: comparison of the average cumulative error obtained by different search algorithms

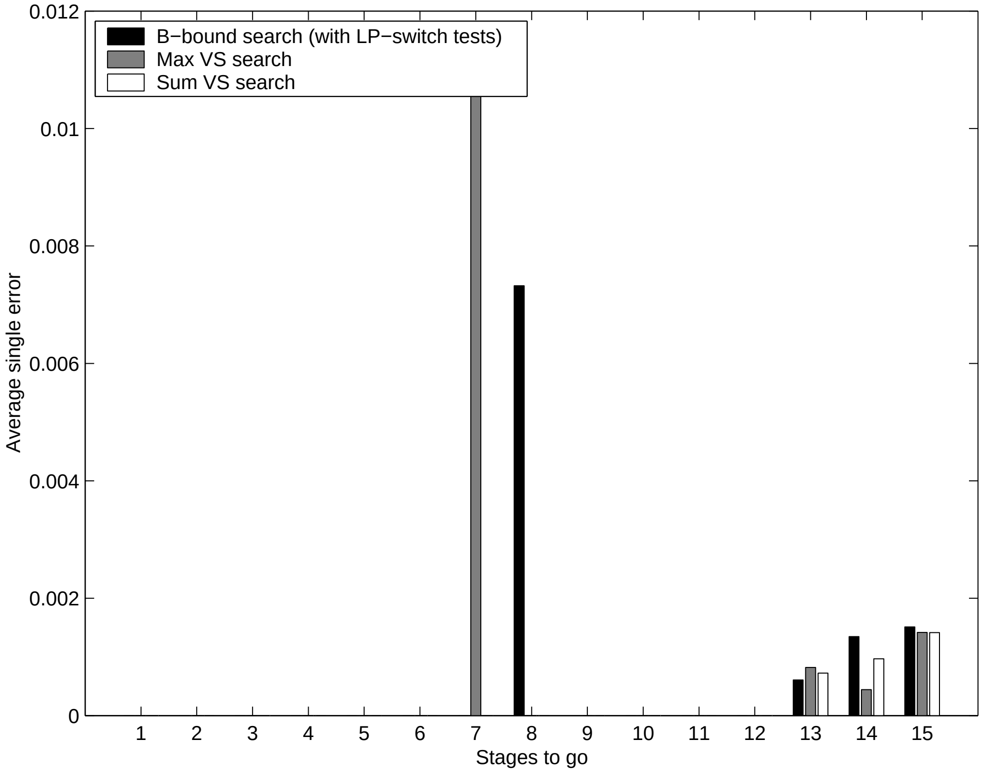


Figure 5.5: Pavement problem: comparison of the average error due to a single approximation at each stage for different search algorithms

Furthermore, the results of the experiment may be biased by the choice of a uniform prior when picking the 5000 initial random belief states. The prior definitely influences the frequency with which some alternative plans are executed and therefore, further experimentation and/or a theoretical analysis would be required to assess the impact of the prior distribution on the performance of each search algorithm.

It is also interesting to compare the average number of plan switches that occur during a policy execution when applying the projection schemes found by those three search algorithms (Figures 5.6, 5.7 and 5.8). Although the average cumulative error is the quantity that a decision theoretic agent wants to minimize, the average number of plan switches provides an interesting benchmark since VS search algorithms were designed to minimize the likelihood of switching. As with the average cumulative error, there isn't a search algorithm that clearly leads to fewer switches on average than the others for all three test problems.

If we compare the average number of switches for different test problems, it is interesting to note that good approximate belief state monitoring for the widget problem seems to be inherently harder than for the coffee problem which in turn seems to be inherently harder than for the pavement problem. Regardless of which search algorithm is used, the average number of switches is much higher for the widget problem than for the coffee problem, which in turn is also higher than for the pavement problem. This suggests a correlation with the average number of α-vectors necessary to represent the value functions. In fact, as indicated in Table 5.4, the widget problem has the highest average number of α-vectors per value-function followed by the coffee problem and finally the pavement problem. Once again, further investigation is needed to assess the potential correlation between the number of α-vectors and the likelihood of switching.

In a last comparison, we look at the progress (in terms of reducing the average cumulative error) achieved while traversing the lattice downwards. In Section 3.3, we showed that any descendant of a node has a corresponding projection scheme

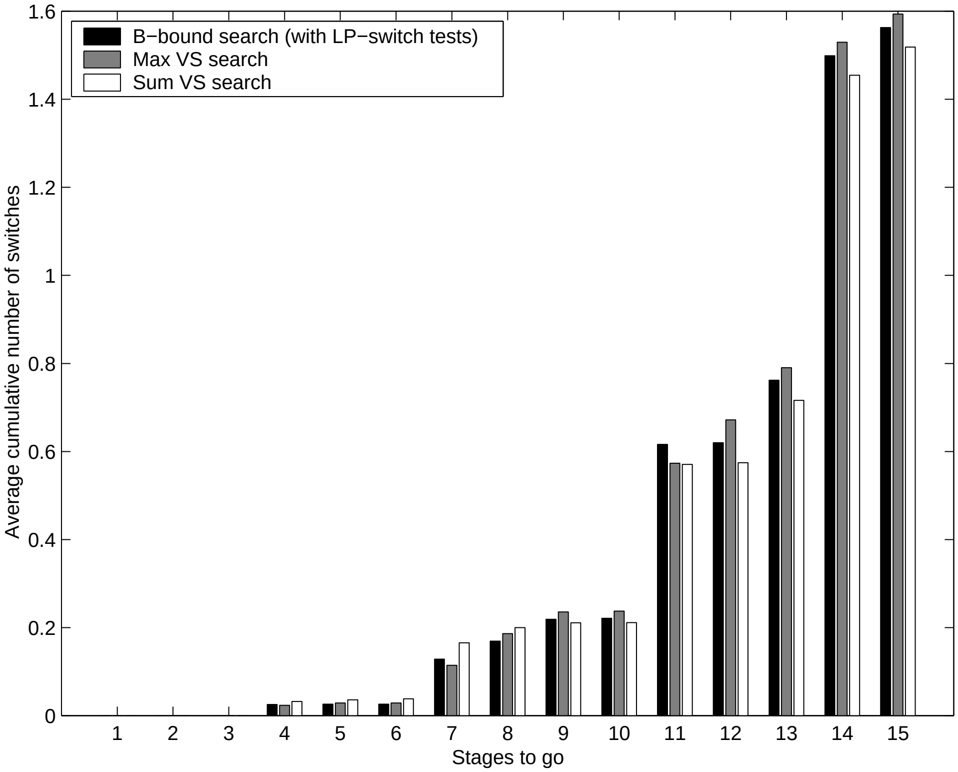


Figure 5.6: Coffee problem: comparison of the average cumulative number of switches for different search algorithms

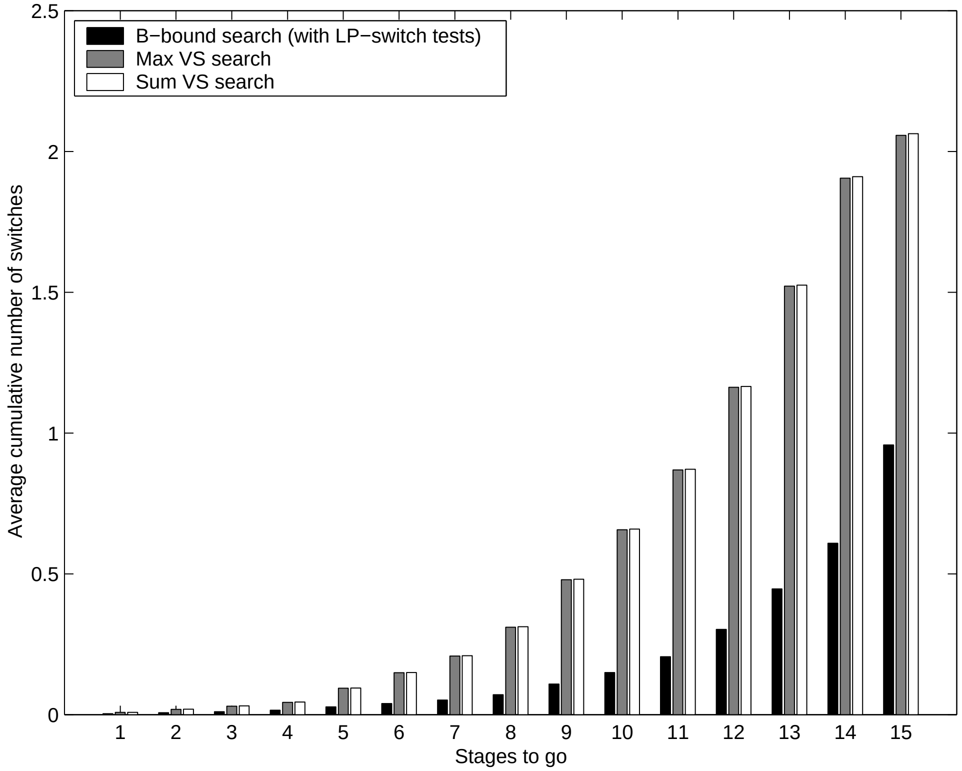


Figure 5.7: Widget problem: comparison of the average cumulative number of switches for different search algorithms

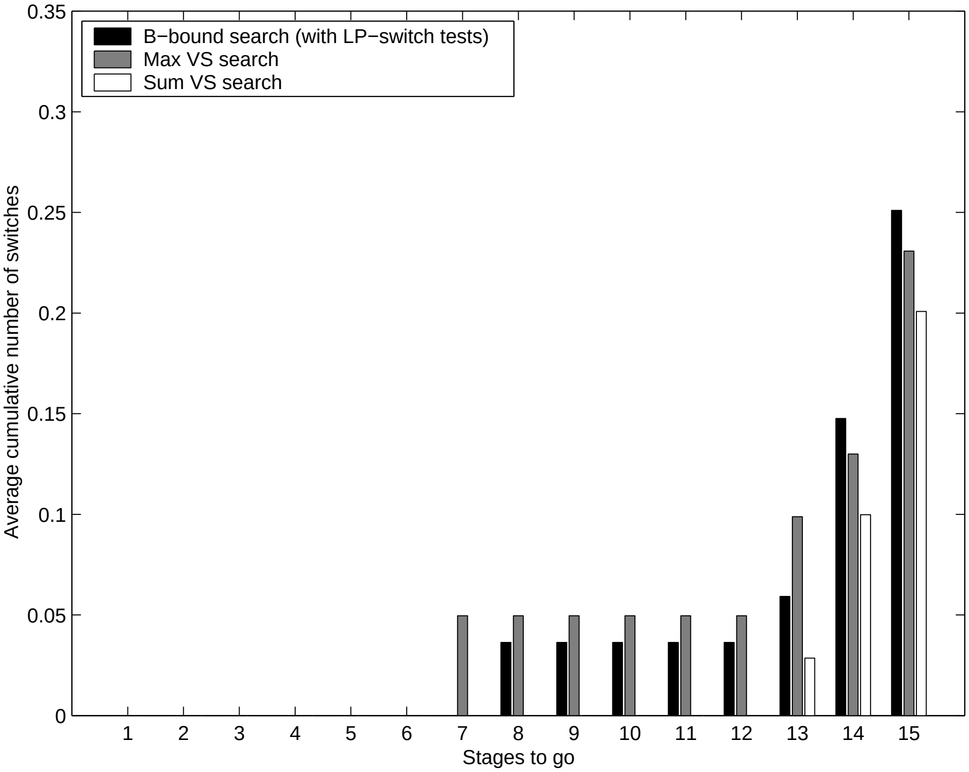


Figure 5.8: Pavement problem: comparison of the average cumulative number of switches for different search algorithms

that guarantees (not necessarily strictly) smaller B, U and E bounds. Similarly, in Section 4.3, we showed how the relative error rate decreases monotonically as we go down the lattice. In practice, the real question is: if we allow our favorite algorithm to search deeper in the lattice, does the average cumulative error diminish? Figures 5.9, 5.10 and 5.11 show that the average cumulative error does in fact tend to decrease when the number of stages to go is large. Those three graphs compare the average cumulative error when a search algorithm is restricted to projection schemes with marginals of at most one variable, at most two variables and at most three variables. Results for the B-bound search, max VS search and sum VS search are presented for the coffee problem.[2]

Note that there is only one projection scheme with marginals of at most one variable and this is when none of the correlations are preserved (i.e., all variables are independent). This naive projection scheme can be considered as a base case since it is at the top of the lattice and there are no simpler projection schemes. Figures 5.9, 5.10 and 5.11 confirm that it is usually possible to do much better than the naive base case.

[2] The widget and pavement problems yield similar results.

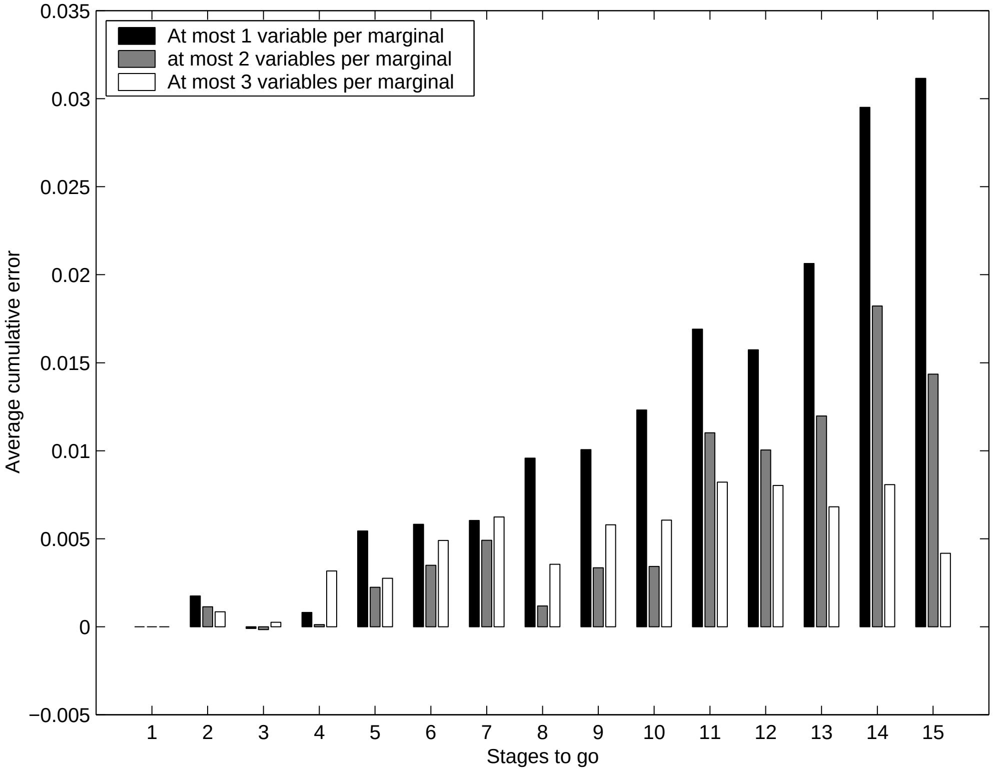


Figure 5.9: B-bound search with LP-switch tests for the coffee problem: comparison of the average cumulative error when the search is restricted to projection schemes with marginals of at most 1, 2, or 3 variables.

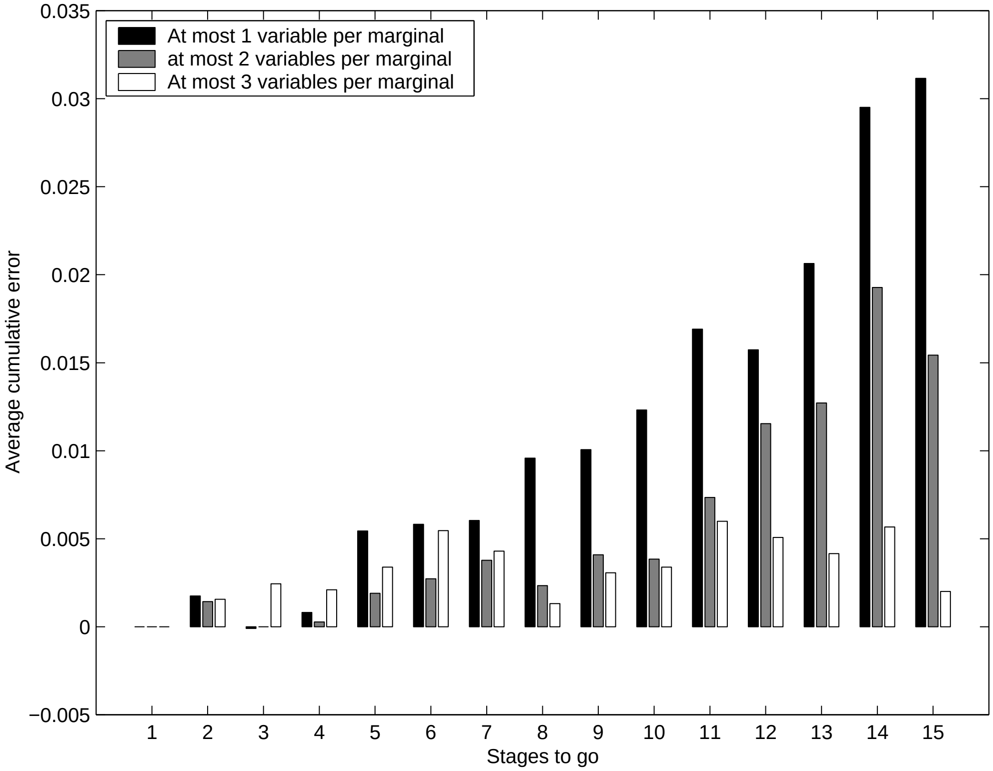


Figure 5.10: Max VS search for the coffee problem: comparison of the average cumulative error when the search is restricted to projection schemes with marginals of at most 1, 2, or 3 variables.

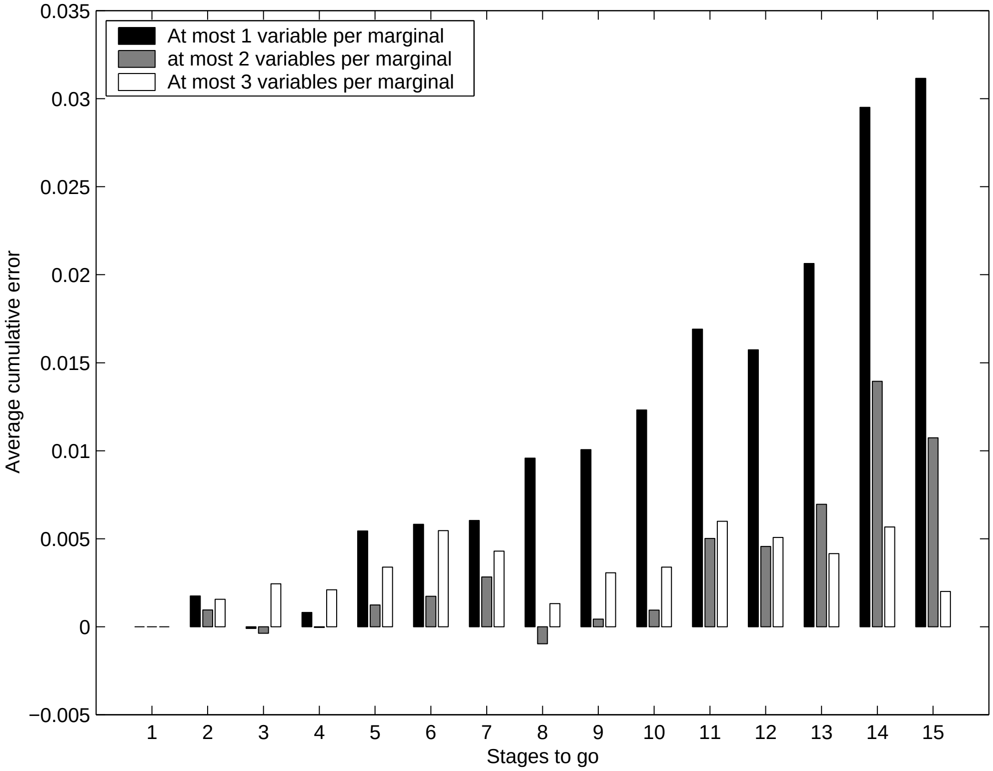


Figure 5.11: Sum VS search for the coffee problem: comparison of the average cumulative error when the search is restricted to projection schemes with marginals of at most 1, 2, or 3 variables.

Chapter 6

Conclusion

6.1 Summary

The primary contribution of this thesis consists of a value-directed framework to perform approximate but efficient belief state monitoring. This framework uses the value function to measure the loss in expected total return incurred when approximating belief states during a policy execution. In Chapter 3, various bounds (B, U and E) on the loss in expected value were derived. The general idea is to construct the set of all plans that may be executed as a result of a single approximation (switch set) or several consecutive approximations (*Alt*-set) and to measure the difference in expected total return between the best and the worst plans. These bounds are generic since they can be computed for a wide range of approximation methods including projection schemes and any linear approximation method (such as density trees). From a practical point of view, the bounds' usefulness is unclear. On one hand, as shown by the experiments in Chapter 5, the average error tends to be significantly smaller than the bounds. This is because the bounds are concerned with the worst case scenario and the algorithms to compute them introduce some looseness. On the other hand, the *relative* scale of the bounds may provide useful information to choose a good projection.

The bounds are then used to direct the search for a good projection scheme by traversing the lattice of projection schemes in a greedy fashion that seeks to minimize some bound. Although in theory the search algorithms guided by those bounds are sound, in practice, several nodes at the top of the lattice exhibit the same error bounds, which provide poor guidance to the search. Furthermore, from a computational point of view, the running time of those search algorithms (B-bound and E-bound search with LP-switch tests) are worse than that of solving a POMDP which is already intractable. As a remedy to those various problems, a vector space formulation was introduced in Chapter 4. It enabled us to define VS-switch tests as an alternative to LP-switch tests. The former has the advantage of avoiding the use of LPs which reduces considerably the overall running time of the search algorithms. On the other hand, this speed up is made possible by introducing more looseness in the bounds.

The vector space analysis also reveals that each projection defines a subspace of directions in which the belief state is approximated. Since the difference in expected return between a pair of α-vectors varies more in some directions than others we can try to select good projection schemes that allow approximations in directions of low variance for most pairs of α-vectors. Using the relative error rate as a quantity to minimize during the search (instead of some error bound) leads to the VS search algorithms defined in Chapter 4. These provide alternative search algorithms, which unlike the different B-bound and E-bound search algorithms, do not suffer from guidance problem since the relative error rate is usually different for each node (even near the top of the lattice). Furthermore, the running time of VS procedures scales better when tackling large POMDPs as it is roughly the same order of magnitude as the solution procedure.

6.2 Future Framework Enhancements

A brief experimental evaluation of the framework was carried out in Chapter 5. Although limited in scope due to the lack of efficient solution algorithms for POMDPs, the experiments enabled us to show how the different algorithms behave in practice on a few test problems. The framework came out of a theoretical study of the approximate belief state monitoring problem and therefore it suffers from several practical deficiencies. A number of directions may be taken to improve the practicality of the bounds and the algorithms. For instance, the primary drawback of the bounds is their looseness. Further research to develop algorithms that compute tighter switch sets and *Alt*-sets is necessary. Also, the bounds are concerned with the worst case scenario, so it would be nice to know under what circumstances this worst case scenario, or some other really bad scenarios, are likely to occur and with what frequency. In the end, since an agent seeks to maximize the *expectation* of its total return, what really matters is not so much the worst case, but rather the average case. Therefore, it would be desirable to have a better understanding of how approximate belief state monitoring influences the agent's returns on average.

The experiments carried out were concerned with finite-horizon POMDPs only. Although the author believes that most of the results obtained for finite-horizon problems translate into very similar results for infinite-horizon problems, further experimentation remains to be done. An important question arises for infinite-horizon problems: we know that the cumulative error tends to increase with the number of stages to go and that it remains bounded (since rewards are assumed to be finite), so in the long run, what does the average cumulative error converge to? Most importantly, how far is it from optimal? This is a key issue for which theoretical as well as practical insights would be desirable.

As for the search algorithms, they suffer from the same sources of intractability as the algorithms to solve POMDPs. In this work, we used factored representations as much as possible to counter one source of intractability, namely, the size of the state space. However, not all POMDPs allow a small enough factored representation and another source of intractability remains: the exponentially growing number of α-vectors. Hence, the proposed algorithms are intractable in the same way that solution algorithms are. Further research is necessary to develop more advanced techniques that can mitigate those sources of intractability for some classes of POMDPs. It is interesting to note that in the event where such techniques would be developed, then the solution algorithms as well as the search algorithms should become tractable for the same classes of POMDPs.

Currently, the framework provides an analysis for approximation methods such as projection schemes, density trees and any other linear approximation method. It would be nice to extend it to sampling methods as they are simple to use, very efficient and they do not assume discrete state variables. In the literature, approximate belief state monitoring methods that use some form of sampling include particle filtering [12, 11], Monte Carlo algorithms [24], condensation algorithms [20], survival of the fittest [23], etc. As with the other approximation methods, those sampling schemes have been applied to monitor general dynamical systems free of any decision process. Value-directed versions of those sampling methods would essentially concentrate the sampling efforts to sensitive areas of the belief space for which the optimal plan to execute is less certain. Sampling methods can also be combined with projection schemes for a further speed up as proposed by Doucet, Freitas, Murphy and Russell [8]. This method, called Rao-Blackwellised particle filtering, should also be analyzed from a value-directed point of view.

Finally, an interesting extension would be to integrate this value-directed framework to the algorithms that solve POMDPs. In other words, as POMDPs are being solved, the solution algorithms could take into account the fact that belief states will be approximated. Ultimately, a decision theoretic agent would seek a bounded-optimal solution to its POMDP since it would try to compute a policy with the highest expected total return given some bound on the amount of time it has to monitor its belief state during policy execution.

Appendix A

Basics of Vector Spaces

In what follows, we present a brief overview of some relevant notions of linear algebra for the development of Chapter 4. For a more thorough overview, the reader is referred to any introductory textbook of linear algebra.

Definition 3 *A **vector space** is roughly speaking a set V of vectors, with two operations (vector addition and scalar multiplication) such that:*

- *For all vectors $v_1, v_2 \in V, v_1 + v_2 \in V$*
- *For any scalar c and for any vector $v \in V$, $cv \in V$*

For our purposes, the set of all $|\mathcal{S}|$-dimensional (real) vectors is a vector space which contains all α-vectors as well as all belief states.

Definition 4 *Let V be a vector space, then a **subspace** of V is a subset W which is itself a vector space.*

Geometrically, a subspace can be viewed as a set of vectors that are unconstrained in length, but constrained in direction. For instance, let V be the vector space corresponding to $\Re^3$. The x-axis is a one dimensional subspace of V that contains all vectors of the form $\langle x, 0, 0\rangle$. Since all those vectors are parallel (they are on the x-axis), then they have the same direction (assuming we ignore orientation).

The yz-plane is another subspace which contains all vectors of the form $\langle 0, y, z\rangle$. Those vectors can have several directions but they are restricted to the yz-plane.

Definition 5 *Let W be a subspace of V, then we define the **null space** of W as the largest subspace $W^{\perp}$ of V such that for all $w \in W, w' \in W^{\perp}$, the dot product of w and w' is zero ($w \cdot w' = 0$).*

It is interesting to note that the dot product of two vectors is zero if and only if they are perpendicular. More generally, recall that the dot product of two vectors v and w is proportional to the cosine of the angle θ_{vw} between them:

$$v \cdot w = \|v\|_2 \|w\|_2 \cos\theta_{vw}$$

Hence, when two vectors are perpendicular, their dot product is zero. Since the vectors of a subspace and its null space are pairwise perpendicular, we can say that a susbpace and its null space are perpendicular subspaces. For example, the x-axis subspace is the null space of the yz-plane subspace in $\Re^3$ and they are clearly perpendicular.

Definition 6 *Let V be a vector space, then a set of vectors W is a **spanning set** of V if every $v \in V$ is a linear combination of the vectors in W. That is, $v = \sum_{w_i \in W} c_i w_i$, where $w_i \in W$ and c_i is some scalar.*

Definition 7 *The vectors of some set W are **linearly independent** if and only if $\sum_{w_i \in W} c_i w_i = 0$ is satisfied only when $c_i = 0$ for all i.*

Definition 8 *Let V be a vector space, then a **basis** B is a smallest spanning set of V composed of linearly independent vectors.*

Definition 9 *The **dimensionality** of a subspace V is given by the smallest number of linearly independent vectors necessary to span it.*

Thus the number of dimensions of a subspace V corresponds to the size of a basis for V.

Definition 10 *Let V be a vector space, then an* ***orthonormal basis*** *B is a basis of V such that:*

- *$\forall v \in B$, v is normal: $\|v\|_2 = 1$.*
- *$\forall v_1, v_2 \in B$, v_1 and v_2 are orthogonal: $v_1 \cdot v_2 = 0$.*

Any basis can be transformed into an orthonormal basis by applying the Gram-Schmidt orthogonalization process. Orthonormal bases are useful when computing the linear projection of a vector on some subspace.

Definition 11 *The* ***linear projection*** *of a vector v on some other vector w is roughly speaking the component of v that is parallel to (has the same direction as) w. Formally, the projection $proj(v, w)$ of v on w is:*

$$proj(v, w) = \frac{v \cdot w}{\|w\|_2} w$$

In turn, the linear projection of a vector v on some vector space V is roughly speaking the greatest component of v that is parallel to (has the same direction as) some vector $v' \in V$. If V has an orthonormal basis B, then one can compute the projection $proj(v, V)$ of v on V as follows:

$$\begin{aligned} proj(v, V) &= \sum_{w_i \in B} proj(v, w_i) \\ &= \sum_{w_i \in B} \frac{v \cdot w_i}{\|w_i\|_2} w_i \\ &= \sum_{w_i \in B} (v \cdot w_i) w_i \end{aligned}$$

Appendix B

Test Problems

The full specifications for the coffee delivery problem, the widget production problem and the pavement maintenance problem follow. The notation used to specify each POMDP is similar to the notation used by Hoey and St.Aubin on their SPUDD (Stochastic Planning using Decision Diagrams) website (http://www.cs.ubc.ca/ spider/jhoey/spudd/submit.html) to describe MDPs using ADDs.

Each POMDP specification includes:

- Set of state variables.
- Set of actions.
- Set of observations.
- Transition function for each action. A transition function is described by a set of conditional probability tables (CPTs) indicating the probabilistic dependencies of each state variable (at the current time step) on the state variables at the current and previous time step. Each CPT is represented compactly by a decision tree.
- Observation function for each action. An observation function describes the probabilistic depencies of each observation on the current state variables. As for transition functions, the dependencies are represented using decision trees.

- Reward function. For each action, a decision tree describes the immediate reward earned in terms of the value of the current state variables.
- Discount factor (number between 0 and 1).

B.1 Coffee Delivery

The coffee delivery problem is a variation on the problem introduced by Boutilier and Poole [2]. Roughly speaking, a robot must deliver coffee to a user when the user wants coffee (variable *wc* is true), but doesn't have any coffee (variable *hc* is false). The robot has two actions: *getC* (to get coffee) and *testC* (to verify if the user wants coffee). When the robot gets coffee, it must go across the street to the local coffee shop to buy coffee. When doing this, it may get wet (variable *w* is true) if it is raining (variable *r* is true) and the robot doesn't have any unmbrella (variable *u* is false). Generally speaking, the robot earns rewards when the user wants coffee (*wc* is true) and has coffee (*hc* is true), and it is penalized otherwise. The robot is further penalized if it gets wet. A small cost is incurred each time it gets coffee, since going across the street consumes a significant amount of energy.

```
variables (w r hc u wc)


actions (getC testC)


observations (ob_wc ob_nwc)


action getC
  r (r (1.0) (0.0))
  u (u (1.0) (0.0))
  w (r (u (w (1.0) (0.0))
          (w (1.0) (1.0)))
```

```
       (u (w (1.0) (0.0))
          (w (1.0) (0.0))))
  hc (hc (1.0) (0.9))
  wc (wc (0.1) (0.0))
endaction

action testC
  r (r (1.0) (0.0))
  u (u (1.0) (0.0))
  w (w (1.0) (0.0))
  hc (hc (0.7) (0.0))
  wc (wc (1.0) (0.3))
endaction

observation getC
  ob_wc (1.0)
  ob_nwc (0.0)
endobservation

observation testC
  ob_wc (wc (0.8) (0.1))
  ob_nwc (wc (0.2) (0.9))
endobservation

reward
  getC (w (hc (wc (0.0) (-2.0))
              (wc (-4.0) (-2.0)))
          (hc (wc (1.0) (-1.0))
              (wc (-3.0) (-1.0))))
  testC (w (hc (wc (0.5) (-1.5))
               (wc (-3.5) (-1.5)))
           (hc (wc (1.5) (-0.5))
               (wc (-2.5) (-0.5))))
endreward

discount 0.90
```

B.2 Widget Production

The following problem is a variation on the widget problem introduced by Draper, Hanks and Weld [10]. A manufacturing robot is in charge of processing widgets. A widget can be flawed (*fl*), slightly flawed (*slfl*), painted (*pa*), shipped (*sh*) or rejected (*re*). Using the *inspect* action, the robot can detect (more or less) whether or not a widget is flawed or slightly flawed. When a widget is not flawed nor slightly flawed, the goal of the robot is to *paint* it and then to *ship* it. If it is slightly flawed, it should first *repair* it, then *paint* it and *ship* it. Finally, if it is flawed, it should simply *reject* it. When the robot is done processing a widget it notifies the supervisor and starts processing a new random widget. As it notifies the supervisor, it also gets a reward if a painted widget without any flaw was shipped or a flawed widget was rejected. Otherwise, it is penalized for processing the widget improperly.

```
variables (fl slfl pa sh re)

actions (inspect repair paint ship reject notify)

observations (bad slbad ok)
```

```
action inspect
  fl (fl (1.0) (0.0))
  slfl (slfl (1.0) (0.0))
  pa (pa (1.0) (0.0))
  sh (sh (1.0) (0.0))
  re (re (1.0) (0.0))
endaction

action repair
  fl (fl (1.0) (0.0))
  slfl (slfl (fl (1.0)
                 (sh (1.0)
                     (re (1.0)
                         (0.05))))
             (0.0))
  pa (pa (1.0) (0.0))
  sh (sh (1.0) (0.0))
  re (re (1.0) (0.0))
endaction

action paint
  fl (fl (1.0) (0.0))
  slfl (slfl (1.0) (0.0))
  pa (pa (1.0)
         (sh (0.0)
             (re (0.0)
                 (0.95))))
  sh (sh (1.0) (0.0))
  re (re (1.0) (0.0))
endaction

action ship
  fl (fl (1.0) (0.0))
  slfl (slfl (1.0) (0.0))
  pa (pa (1.0) (0.0))
  sh (re (0.0) (1.0))
  re (re (1.0) (0.0))
endaction

action reject
  fl (fl (1.0) (0.0))
  slfl (slfl (1.0) (0.0))
  pa (pa (1.0) (0.0))
  sh (sh (1.0) (0.0))
  re (sh (0.0) (1.0))
endaction

action notify
  fl (0.3)
  slfl (0.3)
  pa (0.0)
  sh (0.0)
  re (0.0)
endaction

observation inspect
```

```
  bad (fl (1.0) (0.05))
  slbad (fl (0.0)
            (slfl (0.95) (0.0)))
  ok (fl (0.0)
         (slfl (0.0) (0.95)))
endobservation

observation repair
  bad (0.0)
  slbad (0.0)
  ok (1.0)
endobservation

observation paint
  bad (0.0)
  slbad (0.0)
  ok (1.0)
endobservation

observation ship
  bad (0.0)
  slbad (0.0)
  ok (1.0)
endobservation

observation reject
  bad (0.0)
  slbad (0.0)
  ok (1.0)
endobservation

observation notify
  bad (0.0)
  slbad (0.0)
  ok (1.0)
endobservation

reward
  inspect (-0.2)
  repair (-0.45)
  paint (-0.45)
  ship (0.0)
  reject (0.0)
  notify (sh (fl (-2.0)
                 (slfl (-1.0)
                       (pa (1.0) (-0.5))))
             (re (fl (1.0) (-1.0))
                 (-0.5)))
endreward

discount 0.95
```

B.3 Pavement Maintenance

This problem is inspired from the pavement maintenance problem described by Puterman [32]. The idea is to find an optimal maintenance schedule to minimize the cost of road maintenance. Every year, the local government must decide if it should *redo* the pavement, *patch* it, *inspect* it, or simply *do nothing.* This decision is based on an estimate of 7 binary attributes of the pavement (state variables a, b, c, d, e, f, g). Brand new pavement with no imperfections has all 7 attributes set to true. Over the years, the pavement gradually deteriorates and consequently the variables representing those attributes stochastically tend to become false. There are two classes of attributes, the surface attributes (variables a, b, e, f) and the internal attributes (variables c, d, g). The *patch* action fixes only the surface attributes whereas the *redo* action fixes all attributes. Each action has an inherent cost that varies with the amount of work required. An additional cost measuring voters' dissatisfaction with respect to the general state of the roads is also incurred. The goal of the goverment is to minimize the combination of those costs as indicated by the negative reward function below.

```
variables (a b c d e f g)


actions (patch redo inspect nothing)


observations (good ok bad)


action nothing
  a (a (0.9) (0.0))
  b (b (0.8) (0.0))
  c (c (a (0.9) (0.6))
       (0.0))
  d (d (b (1.0) (0.8))
       (0.0))
  e (e (0.9) (0.0))
  f (f (0.8) (0.0))
  g (g (f (e (1.0) (0.9))
          (0.8))
       (0.0))
endaction

action inspect
  a (a (0.9) (0.0))
  b (b (0.8) (0.0))
  c (c (a (0.9) (0.6))
       (0.0))
  d (d (b (1.0) (0.8))
       (0.0))
  e (e (0.9) (0.0))
  f (f (0.8) (0.0))
  g (g (f (e (1.0) (0.9))
          (0.8))
       (0.0))
endaction

action patch
  a (0.9)
  b (0.95)
  c (c (a (0.9) (0.6))
       (0.0))
  d (d (b (1.0) (0.8))
       (0.0))
  e (0.85)
  f (0.95)
  g (g (f (e (1.0) (0.9))
          (0.8))
       (0.0))
endaction

action redo
  a (1.0)
  b (1.0)
  c (0.95)
  d (0.9)
  e (1.0)
  f (1.0)
  g (0.95)
endaction

observation nothing
  good (1.0)
  ok (0.0)
  bad (0.0)
endobservation

observation inspect
  good (f (a (b (0.9) (0.7))
             (0.8))
          (b (0.7) (0.1)))
  ok (f (a (b (0.1) (0.3))
           (0.2))
        (0.0))
  bad (f (0.0)
         (b (0.3) (0.9)))
endobservation

observation patch
  good (g (d (c (0.7) (0.6))
             (0.4))
          (0.3))
  ok (g (0.2) (0.6))
  bad (g (d (c (0.1) (0.2))
            (0.4))
         (0.1))
endobservation

observation redo
  ok (1.0)
  cr (0.0)
  ho (0.0)
endobservation

reward
  inspect (c (d (g (-0.5) (-1.5))
                (-2.5))
             (g (-2.5) (-3.5)))
  nothing (c (d (g (0.0) (-1.0))
                (-2.0))
             (g (-2.0) (-3.0)))
  patch (c (d (g (-1.0) (-2.0))
              (-3.0))
```

```
           (g (-3.0) (-4.0)))
  redo (c (d (g (-6.0) (-7.0))
             (-8.0))
          (g (-8.0) (-9.0)))
endreward


discount 0.90
```

Bibliography

[1] K. J. Aström. Optimal control of Markov decision processes with incomplete state estimation. *J. Math. Anal. Appl.*, 10:174–205, 1965.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

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

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

[8] Arnaud Doucet, Nando de Freitas, Kevin Murphy, and Stuart Russell. Rao-Blackwellised particle filtering for dynamic Bayesian networks. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 176–183, Stanford, 2000.

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

[10] Denise Draper, Steve Hanks, and Daniel Weld. Probabilistic planning with information gathering and contingent execution. In *Proceedings of the Second International Conference on AI Planning Systems*, pages 31–36, Chicago, 1994.

[11] J. E. Handschin. Monte Carlo techniques for prediction and filtering of non-linear stochastic processes. *Automatica*, 6:555–563, 1970.

[12] J. E. Handschin and D. Q. Mayne. Monte Carlo techniques to estimate the conditional expectation in multi-stage non-linear filtering. *International Journal of Control*, 9(5):547–559, 1969.

[13] Eric A. Hansen. An improved policy iteration algorithm for partially observable MDPs. In *Proceedings of Conference on Neural Information Processing Systems*, pages 1015–1021, Denver, CO, 1997.

[14] 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.

[15] 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*, Breckenridge, CO, 2000. 130–139.

[16] Eric J. Hansen. *Finite-memory control of partially observable systems.* PhD thesis, University of Massachusetts Amherst, Amherst, 1998.

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

[18] 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.

[19] Cecil Huang and Adnan Darwiche. Inference in belief networks: A procedural guide. *Approximate Reasoning*, 11:1–158, 1994.

[20] Michael Isard and Andrew Blake. CONDENSATION—conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29(1):5–18, 1998.

[21] Michael I. Jordan, Zoubin Ghahramani, Tommi S. Jaakkola, and Lawrence K. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37:183–233, 1999.

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

[23] 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.

[24] G. Kitagawa. Monte Carlo filter and smoother for non-gaussian nonlinear state space models. *Journal of Computational and Graphical Statistics*, 5:1–25, 1996.

[25] Uffe Kjaerulff. A computational scheme for reasoning in dynamic probabilistic networks. In *Proceedings of the Eighth Conference on Uncertainty in Artificial Intelligence*, pages 121–129, Stanford, 1992.

[26] Daphne Koller and Raya Fratkina. Using learning for approximation in stochastic processes. In *Proceedings of the 15th International Conference on Machine Learning*, pages 287–295, Madison, 1998.

[27] 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.

[28] Michael L. Littman, Judy Goldsmith, and Martin Mundhenk. The computational complexity of probabilistic planning. *Journal of Artificial Intelligence Research*, 9:1–36, 1998.

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

[30] 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, 1999.

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

[32] M. L. Puterman. *Markov decision problems.* Wiley, New York, 1994.

[33] Y. Sawagari and T. Yoshikawa. Discrete-time Markovian decision processes with incomplete state observation. *Annals of Mathematical Statistics*, 41:78–86, 1970.

[34] 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.

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

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

[37] Sebastian Thrun. Monte Carlo POMDPs. In *Proceedings of Conference on Neural Information Processing Systems*, pages 1064–1070, Denver, 1999.

[38] 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.