Planning and Control in Stochastic Domains with Imperfect Information

by

Milos Hauskrecht

MIT-LCS-TR-738

This report is a modified version of the thesis submitted to the
Department of Electrical Engineering and Computer Science in August, 1997
in partial fulfillment of the requirements for the degree of Doctor of Philosophy

© Massachusetts Institute of Technology 1997. All rights reserved.

Planning and Control in Stochastic Domains with Imperfect Information

by

Milos Hauskrecht

Abstract

Partially observable Markov decision processes (POMDPs) can be used to model complex control problems that include both action outcome uncertainty and imperfect observability. A control problem within the POMDP framework is expressed as a dynamic optimization problem with a value function that combines costs or rewards from multiple steps. Although the POMDP framework is more expressive than other simpler frameworks, like Markov decision processes (MDP), its associated optimization methods are more demanding computationally and only very small problems can be solved exactly in practice. Our work focuses on two possible approaches that can be used to solve larger problems: approximation methods and exploitation of additional problem structure.

First, a number of new efficient approximation methods and improvements of existing algorithms are proposed. These include (1) the fast informed bound method based on approximate dynamic programming updates that lead to piecewise linear and convex value functions with a constant number of linear vectors, (2) a grid-based point interpolation method that supports variable grids, (3) an incremental version of the linear vector method that updates value function derivatives, as well as (4) various heuristics for selecting grid-points. The new and existing methods are experimentally tested and compared on a set of three infinite discounted horizon problems of different complexity. The experimental results show that methods that preserve the shape of the value function over updates, such as the newly designed incremental linear vector and fast informed bound methods, tend to outperform other methods on the control performance test.

Second, we present a number of techniques for exploiting additional structure in the model of complex control problems. These are studied as applied to a medical therapy planning problem—the management of patients with chronic ischemic heart disease. The new extensions proposed include factored and hierarchically structured models that combine the advantages of the POMDP and MDP frameworks and cut down the size and complexity of the information state space.

Keywords: Artificial Intelligence, partially observable Markov decision processes, planning and control under uncertainty, decision-theoretic planning, medical therapy planning, dynamic decision making, Bayesian belief networks.

Acknowledgments

I would like to thank:

Dr. Mikulas Popper for guiding my first steps in the field of Artificial Intelligence and for inspiring this in the first place.

Professor Peter Szolovits, my thesis advisor, for his guidance, encouragement and financial support over the course of my study.

Dr. William Long for being supportive and encouraging over the years, and for his patience and help while proofreading my papers and correcting my grammar mistakes.

Dr. Hamish Fraser for his time and help with the ischemic heart disease example.

Professors Leslie Kaelbling and Paul Viola, and Dr. Geoffrey Rutledge, my thesis readers, for their valuable feedback and comments on the draft of the thesis.

Anthony Cassandra, for interesting and stimulating discussions on POMDPs and for providing the exact solution for one of the test problems I used in my thesis.

Past and present members of the Clinical Decision Making Group for creating an intellectually stimulating environment, for their friendship and support. My officemates Mike Frank and Eric Jordan for making the room 421 a brighter place. Mike Frank, Heather Grove, Eric Jordan, William Long, Latanya Sweeney, and Patrick Thompson for proofreading parts of my thesis.

My parents for their unbounded love, support and belief in me.

My wife, Jana, for bearing the negative side of the study, for her patience, love, and sacrificies.

This research was supported by grant RO1 LM 04493 and grant 1T15LM07092 from the National Library of Medicine, and by DOD Advanced Research Project Agency (ARPA) under contract number N66001-95-M-1089.

Contents

1 Introduction **11**
- 1.1 Two basic control agent designs . . . 11
  - 1.1.1 Associative approach . . . 13
  - 1.1.2 Model-based approach . . . 13
  - 1.1.3 Combination of approaches . . . 14
  - 1.1.4 Approach pursued in this research . . . 14
- 1.2 Partially observable Markov decision processes . . . 14
  - 1.2.1 Partially observable Markov decision processes . . . 14
  - 1.2.2 Solving POMDP problems . . . 15
- 1.3 Solving control problems for larger POMDPs . . . 16
  - 1.3.1 Approximations . . . 16
  - 1.3.2 Exploiting additional problem structure . . . 17
- 1.4 Structure of the text . . . 18
- 1.5 Brief summary of chapters . . . 19

2 Markov decision processes **21**
- 2.1 MDP model and MDP problem . . . 21
  - 2.1.1 MDP problem . . . 22
  - 2.1.2 Control functions and policies . . . 24
  - 2.1.3 Types of MDP problems . . . 25
  - 2.1.4 Real-time control . . . 26
- 2.2 Solving the MDP problem . . . 26
  - 2.2.1 Finite horizon problem . . . 26
  - 2.2.2 Infinite discounted horizon problem . . . 28
- 2.3 Forward methods for solving MDP problems . . . 33
  - 2.3.1 Computing optimal control plans for the finite horizon model . . . 33
  - 2.3.2 Finding the optimal action for a single initial state . . . 34
- 2.4 Solving large MDP problems . . . 38
- 2.5 Summary . . . 41

3 Partially observable Markov decision process **43**
- 3.1 Partially observable Markov decision process . . . 44
- 3.2 Control in partially observable domains . . . 45
  - 3.2.1 Information state . . . 45
  - 3.2.2 Value functions in POMDP . . . 47
  - 3.2.3 Value function mappings . . . 48
- 3.3 Constructing information state MDPs for different POMDP models . . . 50

3.3.1 POMDP with standard (forward triggered) observations 50
3.3.2 POMDP with backward triggered observations 52
3.3.3 POMDP with the combination of forward and backward observations . . 53
3.3.4 POMDP with delayed observations 54
3.4 Computing optimal control policies for POMDPS 57
3.4.1 Computing optimal control policy 57
3.4.2 Computability of the optimal or ϵ optimal solutions 58
3.5 Algorithms for updating piecewise linear and convex value functions 61
3.5.1 Monahan's algorithm 62
3.5.2 Extensions of Monahan's algorithm 63
3.5.3 Cheng's linear support algorithm 65
3.5.4 Witness algorithm 68
3.5.5 Value iteration updates 69
3.6 Forward decision methods for finding optimal or near-optimal POMDP control . 71
3.6.1 Computing value function bounds 71
3.6.2 Incremental forward methods 72
3.6.3 Incremental breadth-first expansion strategy 73
3.6.4 Using heuristics to guide the decision tree expansion 73
3.6.5 Computing the decision in linear space 74
3.6.6 Combining bound improvement strategies 76
3.7 Summary 77

4 Approximation methods for solving POMDP problems 79
4.1 Types of approximation methods 79
4.1.1 Approximations of value functions and policies 79
4.1.2 Approximation (reduction) of the model 80
4.1.3 The combination of the two approaches 80
4.1.4 The structure of the chapter 80
4.2 Value function approximations 80
4.2.1 Using approximate value functions to compute control response 80
4.2.2 Incremental methods 81
4.2.3 The role of value function bounds 82
4.2.4 Convergence and stability of iterative methods 82
4.2.5 Described value function approximation methods 83
4.3 MDP-based approximations 83
4.3.1 Upper bound property 84
4.3.2 Infinite horizon solution 84
4.3.3 Summary of the method 85
4.4 Fast informed bound method 85
4.4.1 Complexity of a new update rule 86
4.4.2 Bound property of the new update strategy 86
4.4.3 Infinite discounted horizon case 87
4.4.4 Extensions of the fast informed bound method. 87
4.4.5 Summary of the fast informed bound method 88
4.5 Blind policy approximations 88
4.5.1 Blind policy 88
4.5.2 Constructing a complete blind update rule 90
4.5.3 Efficient blind policy methods 93

4.5.4 Extensions to the fixed policy method . . . 93
4.5.5 A summary of a blind policy method . . . 95
4.6 Approximation of a value function using curve fitting (least-squares fit) . . . 95
4.6.1 Versions of least-squares fit . . . 95
4.6.2 Combining value iteration and least-squares fit . . . 96
4.6.3 Parametric function models . . . 97
4.6.4 Summary of least-squares fit . . . 97
4.7 Grid-based interpolation-extrapolation strategies . . . 98
4.7.1 Properties of convex rules . . . 100
4.7.2 Constructing grids . . . 102
4.7.3 Bound property of the point-interpolation rule . . . 103
4.7.4 Extensions of the simple interpolation rule . . . 106
4.7.5 Summary of grid-based interpolation-extrapolation methods . . . 106
4.8 Grid-based linear vector method (grid-based Sondik's method) . . . 107
4.8.1 Lower bound property of the grid-based Sondik's update . . . 107
4.8.2 Infinite horizon case . . . 108
4.8.3 Summary of the grid based linear vector method . . . 110
4.9 Approximation of policies . . . 110
4.9.1 Representing control using policy (control) trees . . . 110
4.9.2 Other policy approximation methods . . . 111
4.10 Model based approximations . . . 111
4.10.1 Approximation of the information state space . . . 112
4.10.2 Truncated history . . . 114
4.10.3 POMDP model reduction . . . 116
4.11 Summary . . . 116

5 Value function approximation methods: an experimental study 119
5.1 Test problems . . . 119
5.2 Comparing quality of bounds . . . 121
5.2.1 Methods tested . . . 121
5.2.2 Experimental design . . . 123
5.2.3 Test results . . . 123
5.2.4 Evaluation of results . . . 123
5.3 Testing control performance of approximation methods . . . 130
5.3.1 Methods tested . . . 131
5.3.2 Experimental design . . . 132
5.3.3 Test results . . . 132
5.3.4 Evaluation of results . . . 141
5.3.5 Summary of test results . . . 146
5.4 Runtimes of methods . . . 147
5.5 Experimental biases . . . 147
5.6 Summary . . . 148

6 Extending POMDP framework: Management of ischemic heart disease 151
6.1 Modeling diagnostic and therapeutical processes using POMDPs . . . 151
6.1.1 Chronic ischemic heart disease . . . 152
6.1.2 A typical decision scenario . . . 153
6.2 Using POMDP to model IHD . . . 153
6.2.1 Representing states . . . 153

6.2.2 Actions . . . 155
6.2.3 Representing stochastic transition and observation models . . . 156
6.2.4 Initial state model . . . 158
6.2.5 Cost model . . . 160
6.2.6 Discretizing time . . . 160
6.2.7 Modeling the objective function . . . 161
6.3 Acquisition of model parameters . . . 161
6.3.1 Acquisition of transition and observation probabilities . . . 161
6.3.2 Acquiring cost model parameters . . . 162
6.4 Finding control policy for the IHD domain . . . 165
6.4.1 Representing information state . . . 165
6.4.2 Compilation of the model . . . 166
6.4.3 Solving control problem . . . 166
6.5 Evaluating the model . . . 168
6.5.1 Testing obtained policy for the patient follow-up case . . . 168
6.5.2 Alternative decision choices . . . 168
6.5.3 Problems with the current model . . . 170
6.6 Summary . . . 171

7 Conclusion **173**
7.1 Solving the POMDP problem . . . 173
7.1.1 POMDP exact methods . . . 173
7.1.2 POMDP approximations . . . 174
7.1.3 Extensions of the POMDP framework . . . 174
7.2 Contributions . . . 174
7.2.1 Exact POMDP methods . . . 175
7.2.2 POMDP approximation methods . . . 176
7.2.3 Comparison and tests of approximations methods . . . 177
7.2.4 Extensions of the basic POMDP framework . . . 177
7.2.5 Application of the POMDP framework . . . 178
7.3 Challenges and further research directions . . . 179
7.3.1 Attacking large problem domains . . . 179
7.3.2 Learning in partially observable stochastic control domains . . . 181

A Test problems **183**
A.1 Maze20 problem . . . 183
A.2 Maze20B problem . . . 186
A.3 Shuttle docking problem . . . 189

Chapter 1

Introduction

The construction of intelligent control agents that function in the real world has become a focus of interest for many researchers in the AI community in recent years. This line of research was triggered by an attempt to benefit from advances and results in the fields of data interpretation, diagnosis, planning, control, and learning, and combine them into more sophisticated systems, capable of solving more complex problems.

What do we expect from a control agent?
The agent is expected to live in the world. It accomplishes goals and fulfills its intentions by observing and actively changing the world. In order to do so it must exploit a combination of perceptual, acting, and reasoning capabilities. Examples of control agents include:

- robot arm controller;
- autopilot;
- medical life support device that monitors patient status and executes appropriate actions when needed.

Figure 1-1 shows the basic high level view of a control agent and its relation to the external environment. The agent interacts with the environment through actions and observations. Actions allow the agent to change the environment. On the other hand observations allow it to receive and collect the information about the environment. The control agent is designed to achieve a goal. In order to achieve the goal it coordinates its perceptual and acting capabilities: actions to change the environment in the required direction and observations to check the results of action interventions.

1.1 Two basic control agent designs

In the ideal case the control agent would perform the best possible sequence of actions leading to the goal. In order to achieve the optimal or close to optimal control sequence the agent can be designed to either:

- follow hard coded and preprogrammed control sequences;
- use the agent's model of the world's behavior and the agent's goals and try to figure out (compute) the appropriate control autonomously.

The two basic design alternatives are illustrated in figure 1-2.

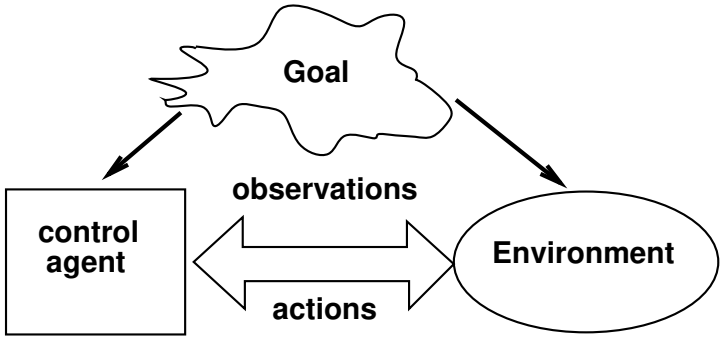


Figure 1-1: Basic control agent scheme.

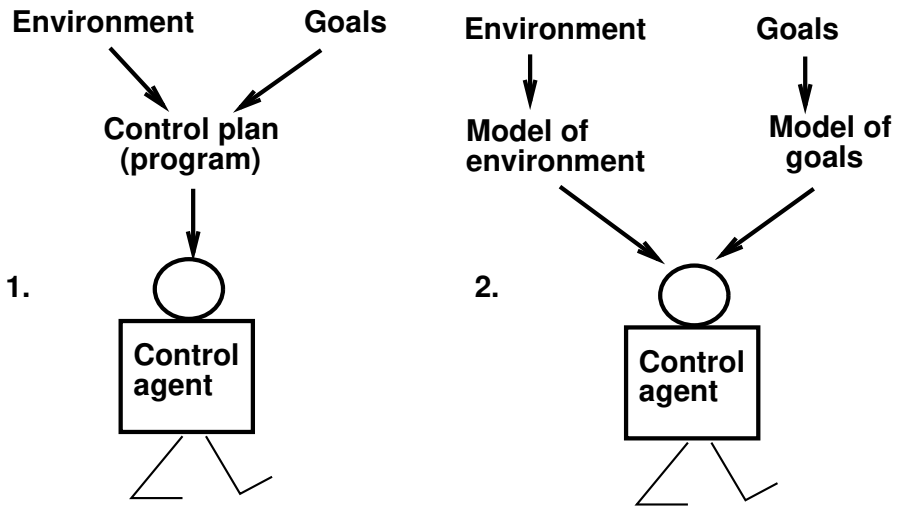


Figure 1-2: Two control agent designs: 1. Agent runs a standardized procedure (program); 2. Agent works with models of the environment and goals.

1.1.1 Associative approach

The first design alternative is based on the simple idea of knowing directly what to do or how to respond in every situation. The idea, although simple and "unintelligent," can be the basis of a high quality control agent. The examples of this kind of control system design include table-based, rule-based and protocol-based (guideline-based) architectures.

The major advantage of this approach is that it can provide rapid control responses and thus may be suitable for time critical applications. Its disadvantage stems from the fact that it relies on the external control plan source, and responsibility for the quality of the control is entirely on the shoulders of the control plan provider. This means that the external provider (usually human) must do the hardest part and "solve" the problem of how to achieve the goal by considering every situation and encoding it into the control plan. The other major disadvantage of the approach is that both the goals the controller tries to pursue and the behavior the controlled system exhibits are implicit. This causes the following:

- a control agent has no or very limited explanation capability. It has no means to justify selected control responses with regard to goals. This feature may be very important in some application areas like medicine;
- a specific control agent can be hard to update and modify, when the objectives of the controller or the description of the behavior changes.

1.1.2 Model-based approach

The second alternative assumes that control is inferred by a control agent autonomously from the description of the environment behavior and the goals to be pursued. In this case the responsibility for the quality of control is more on the side of the control agent itself and is mostly dependent on the design of its inference procedures, although providing wrong models can cause suboptimal control with regard to goals as well. The advantage of this approach is that the task of finding and selecting optimal control is performed by the controller autonomously and the external provider is required to supply only the appropriate models of goals and behavior, a task that is usually simpler than providing complete control plans. Other advantages to work with models of goals and behavior are:

- the model can be used for other tasks as well, for example prediction, diagnosis or explanation;
- the controller is easier to modify and update, that is, changes in goals or behavioral description are relatively easy to incorporate.

An obvious disadvantage of the model-based approach is that the optimal control response to be used must be found, which usually leads to longer reaction times, due to the complexity of the underlying optimization problem.

Compilation of control

The computation of optimal response, when done during the control, can cause significant delays in response times. This may be unacceptable in some time critical applications that do not tolerate large time delays. The problem with response delays due to computation can be partially or completely eliminated by performing some or all computation beforehand and storing computed results to speed up the on-line control. In the extreme case this reduces to

the *compilation process* that takes the model description and outputs corresponding control plans that in turn drive the operation of the control agent. The compilation module uses the model description to provide control plans that are much like control plans designed by a human expert.

1.1.3 Combination of approaches

The design of the real control agent does not have to fall strictly into one of the above categories and one can exploit advantages of both approaches. This can lead to the hybrid design where a control agent uses both control plans as well as models of the environment and goals or subgoals to perform the control. The two approaches can be combined easily by decomposing the original control problem into smaller subproblems and building a hierarchy of control agents (or modules) along this decomposition, where each agent is reponsible for some control task and each can be designed differently. The control sequences performed by agents on the lower level are then considered to be actions of the higher level control agent.

1.1.4 Approach pursued in this research

Although both approaches are equally important for solving complex control problems, in our research we focus on the model-based alternative and explore the problems related to modeling dynamic stochastic systems and control goals as well as to problems of computing optimal control responses for such models.

1.2 Partially observable Markov decision processes

Models of environments and goals can be of different type and complexity. The model of the environment can be deterministic or stochastic, described using discrete or continuous states, discrete or continuous time, described by simple transition relations or by differential equations. The goal can be a simple state or it can be defined over some time horizon.

The task of inferring the optimal control from models is largely dependent on the selected modeling framework and its complexity. The relation between the two is proportional: the more expressive the modeling framework, the more complex the associated computation of optimal control. Therefore one must often trade off the benefits and costs of applying different models. For example, selecting a simpler model usually leads not only to simpler computation procedures for finding optimal control but also to a cruder approximation to reality, and loss of precision. On the other hand, a more complex model and framework can approximate reality better, but finding optimal control solutions can be computationaly very expensive or even impossible. Therefore while selecting the framework one must carefully decide which features are less important and can be abstracted away and which need to be considered.

1.2.1 Partially observable Markov decision processes

There are many reasonable modeling frameworks one can explore with regard to various control problems. In our work we study the framework of partially observable Markov decision processes (POMDP) [Astrom 65] [Smallwood, Sondik 73] [Lovejoy 91a]. A POMDP describes controlled stochastic process with partially observable process states. It can be used to model dynamic environments (systems) and their partial observability by the control agent. The framework has been studied by researchers from different areas, mostly in control theory and operations

research and recently also by researchers in Artificial Intelligence (AI) [Cassandra et al. 94], [Cassandra 94], [Littman et al. 95a], [Parr, Russell 95].

The POMDP framework is closely related to the more common formalism of Markov decision processes (MDP) [Bellman 57] [Howard 60] [Puterman 94]. The main distinction is that POMDPs are more expressive and model partial observability of the controlled process, while MDPs assume that process states are always perfectly observable. Thus POMDPs allow us to represent two sources of uncertainty: uncertainty related to the behavior of the process under different interventions and uncertainty related to imperfect observability of process states. Also, POMDPs can represent investigative (perceptual) actions, that is actions that induce or trigger observations.

The main characteristics of the POMDP framework are:

- the world (environment) is described using a *finite set of states*, and the control agent can actively change them using a *finite set of actions*;
- the dynamics of the world is described using *stochastic transitions* between states that occur in *discrete time steps*;
- information about the actual world state is not available to the control agent directly but through a *finite set of observations*;
- the quality of control is modeled by means of numerical quantities representing *rewards (or costs)* associated with states or state transitions;
- the *control goal* is represented by an objective function that combines costs or rewards obtained over multiple steps.

Application areas

The main advantage of the POMDP framework is its ability to represent control and planning problems in stochastic and partially observable domains. Robot navigation [Littman et al. 95a] [Cassandra et.al 96], medical therapy planning [Hauskrecht 96a] [Hauskrecht 97a], and machine maintainance and replacement [Smallwood, Sondik 73] [Lovejoy 91b] are typical application areas.

In all these domains one faces two sources of uncertainty: action outcome uncertainty and imperfect observability. For example, with some probability a robot can move in the wrong (unintended) direction, and the information it receives from its sensors is often unreliable and subject to error. In the medical domain, a specific therapy can lead to different outcomes for a given disease, and symptoms for two or more diseases can overlap. In both examples the underlying state (a location of the robot or the disease the patient suffers from) is not known with certainty and all possible states need to be considered during planning.

The problem to solve in such domains is to determine the best sequence of actions (control plan, policy) with regard to the modeled control objectives. The control objectives to be optimized are related to multiple steps, and may correspond to the reduction of the number of steps needed to achieve the target location for the robot navigation task, or the increase in the quality and length of life of a patient suffering from a disease.

1.2.2 Solving POMDP problems

The POMDP offers a powerful theoretical framework for modeling partially observable dynamic controlled processes. However, the price paid for the increased expressivity of the framework is

that finding optimal or near-optimal control solutions for POMDP problems is computationally intractable. This is unlike control problems defined within the fully observable finite-state MDPs, because they can be solved efficiently [Puterman 94] [Bertsekas 95] [Littman et al. 95b].

In principle POMDP problems can be solved by converting POMDPs to information-state MDPs (see [Bertsekas 95]), and by using standard solution strategies developed for MDPs, like dynamic programming or value iteration. An information state summarizes all relevant information learned about the process and it is represented by a complete history of all observations and actions or by a quantity corresponding to a sufficient statistic that preserves the Markov property of the information process. The problem with using information states is that a space of all possible information states can be infinite or of expanding dimension. This makes it hard to compute complete dynamic programming and/or value iteration updates. Luckily, it can be shown that complete updates are computable for a class of POMDPs that can be reduced to belief state MDPs (a belief state assigns probability to all underlying process states). This is mostly because the objective value function for a belief space MDP is piecewise linear and convex [Smallwood, Sondik 73].

Although dynamic programming updates are computable for belief state POMDPs, the complexity of the piecewise linear value function (number of linear vectors defining it) can grow exponentialy in every update. This allows us to compute optimal solutions only for POMDPs with small state, observation and action spaces in practice. For example, no success with exact methods has been reported for POMDPs with more than 10 process states and infinite discounted horizon criteria.

Despite the modeling expressiveness, the problem of computational efficiency of exact methods leaves open the question of practical applicability of the POMDP framework, especially in solving larger and more complex control problems. The main theme of our research work was to explore various ways and propose solutions that would help us to make the framework applicable to larger size domains.

1.3 Solving control problems for larger POMDPs

The problem of computational complexity of exact optimization methods prevents us from using them for solving more complex POMDPs. In our work we focused on two solutions that allow us to attack larger problems:

- approximation methods;
- exploitation of the additional problem structure.

1.3.1 Approximations

The main idea behind approximations is to trade off the precision of the solution for speed. Thus, instead of computing the optimal solution one attempts to compute a good solution fast[1]. There are different approximation methods that can be applied in the context of POMDPs. These focus on:

- approximations of value functions (policies);
- approximations (reductions) of information-state MDPs.

[1] The term approximation as used in the MDP and POMDP literature and also here does not refer to the approximation that is guaranteed to be within some factor from the optimal solution.

In the first case the approximation targets the value function and uses a simpler value function form and simpler dynamic programming (value iteration) updates (see [Lovejoy 91b] [Littman et al. 95a]). In the second case the information-state MDP is reduced to simpler model, for example through feature extraction mappings [Bertsekas 95] [Tsitsiklis, Van Roy 96] or by using truncated histories [White, Scherer 94].

Value function approximations

There are several value function approximation methods researchers have developed to substitute hard to compute exact methods. These include methods that use MDP-based solutions (see [Lovejoy 93], [Littman et al. 95a]), grid-based updates and nonparametric value function approximations (see [Lovejoy 91b]), grid-based updates of derivatives (see [Lovejoy 91b]) and parametric value functions and least-squares techniques (see [Littman et al. 95a] [Parr, Russell 95]). However the list of methods is far from being complete and there is still a lot of room for improvements.

In our work we proposed and developed new methods and some extensions of the existing methods. These are based on different ideas and include the fast informed bound method (section 4.4) based on approximate dynamic programming updates that lead to piecewise linear and convex value functions with constant number of linear vectors (equals the number of actions), a new grid-based point interpolation method that supports variable grids (section 4.7.3), an incremental version of the linear vector method that updates value function derivatives (section 4.8.2), as well as various heuristics for selecting grid-points (see sections 4.7.3 and 4.8.2).

The lack of experimental studies

Although there is a relatively large number of approximation methods developed, there has been a lack of studies that would compare empirically their performance and that would help us to understand better the advantages and disadvantages of different approximation approaches.

To address this problem, we selected three POMDP problems of different complexities and used them to test several methods and their modifications (Chapter 5). The main purpose of testing was to get an idea about how methods compare to each other, what things matter more and which one are less important, and identify methods or modifications that are inferior or superior to others. The methods were tested from two perspectives: the quality of value function bounds for methods that are guaranteed to provide them (section 5.2) and the quality of control where methods were judged solely based on their control performance (section 5.3).

1.3.2 Exploiting additional problem structure

A complementary approach that helps us to attack larger size problems is based on the exploitation of additional problem structure, i.e. structure that cannot be expressed in the classical POMDP framework. Structural extensions and refinements can be used to reduce the complexity of the information-state space and value functions one needs to work with and thus speed-up the problem-solving routines.

To study structural extensions (Chapter 6) of the basic framework we used a medical theraphy planning problem — the management of patients with ischemic heart disease (see [Wong et al. 90]). The problem relies on both sources of uncertainty (stochastic action outcomes and partial observability) and thus fits well the POMDP framework.

Combining MDP and POMDP frameworks

The basic POMDP framework assumes that process states are always hidden and information about the state can be acquired only through observations. However this is not always true, and one often works with process states that consist of both observable and hidden components.

To address this issue we proposed new structural extensions that combine advantages of MDP and POMDP frameworks (Chapter 6). This was achieved by using factored process state models with both observable and hidden components (state variables), as well as a hierarchy of state variables that directly restricts certain combinations of state variable values (section 6.2.1). Both of the extensions reduce significantly the complexity of the information-state space and value function definitions for the ischemic heart disease problem, compared to the standard POMDP approach (section 6.4.1).

Other model extensions

To construct the model for the ischemic heart disease problem we had to deal also with issues that are not directly relevant to the control optimization problem but are very important from the viewpoint of model building. The main issue here is the size and complexity of the model, namely the number of parameters one needs to estimate. To reduce the complexity of the model definition we use factored transition and observation models represented using hierarchical version of the Bayesian belief network (section 6.2.3), and a factored cost model (section 6.2.5). Such models explicitly represent independencies and regularities that hold among the model components and reduce its complexity (section 6.3). Once defined the model can be compiled and optimized for the purpose of planning (section 6.4.2).

1.4 Structure of the text

The main objective of our work is to explore, study and propose various ideas that help to make the POMDP framework applicable to larger size domains. To do this, we focused mostly on:

- value function approximation methods;
- extensions of the basic POMDP framework that exploit additional problem structure;
- improvements of exact methods.

Closely related to approximations is an issue of experimental comparison of different approximation methods. These topics are central to the thesis and account for most of our contributions. They are presented in separate chapters:

- Chapter 3. POMDP framework and exact methods for solving control problems within it.
- Chapter 4. Approximation methods.
- Chapter 5. Experimental test, comparison and analysis of new and existing value function approximation algorithms.
- Chapter 6. Extensions of the basic framework, exploitation of the additional problem structure.

1.5 Brief summary of chapters

The text covers most of the field of Partially observable Markov decision processes. It is organized in chapters that address different topics related to the framework. An effort to present new ideas and methods in relation to the previous work has been made. This is also the reason why contributions are not presented on one place but are rather scattered throughout the thesis. However, they are pointed out and summarized at the end of each chapter, and are reviewed again in the conclusion.

The following is a brief overview of every chapter that also includes pointers to our contributions:

Chapter 2 describes the basics of the framework of the Markov decision process (MDP) that models controlled stochastic processes under the assumption of perfect observability and control problems one can solve within such a framework. The MDP framework is introduced mostly to simplify the explanation of more complex POMDP framework that is central to our work. The reason for this is that many of the solution methods developed for the MDP are directly applicable or very similar to methods used for POMDPs. The understanding of the MDP and POMDP frameworks, their differences and respective advantages will be helpful for chapter 6 in which the framework that exploits the combination of MDPs and POMDPs will be introduced.

Chapter 3 introduces the framework of Partially observable Markov decision processes and describes exact methods for computing control solutions within it. The POMDP extends the MDP framework by incorporating features of partial observability and control over observations. Our work in this chapter is centered mostly around the exploration and the developement of a number speed-up techniques for exact optimization methods. These include: new Gauss-Seidel version of the value iteration algorithm that is based on the idea of incremental lower bounds improvements (section 3.5.5); improvements of the basic Monahan's algorithm [Monahan 82] [Cassandra et al. 97] that interleave generate and pruning phases of the value function construction and prune partially constructed value functions across different actions (section 3.5.2); the design of various forward decision methods that select the best control action for a single initial state (section 3.6). Also studied are alternatives to the standard POMDP models that use different or more complex observation-state dependencies including for example a model with delayed observations (section 3.3).

The problem of finding the optimal control within the POMDP is computationally hard and exact methods are highly inefficient [Papadimitriou, Tsitsiklis 87]. This naturally leads to the exploration of methods that can acquire good solution faster, trading off the accuracy of the solution for speed. The exploration of such methods is the subject of *Chapter 4*. The chapter includes the description of a number of new and known approximation methods, and analyzes and compares their theoretical properties. The new methods and novel improvements of existing methods are: fast informed bound (section 4.4), simple variable grid point-interpolation method (section 4.7.3), incremental linear vector method (section 4.8.2), and heuristic strategies for selecting grid points (sections 4.7.3 and 4.8.2).

New and known value function approximation methods were experimentally tested and their results compared and analyzed in *Chapter 5*. The experiments were used to test two features of approximation methods and their solutions: the quality of bounds (section 5.2) and the quality of control performance (section 5.3). Tests were conducted on the set of three POMDP control problems of different complexity that include two robot navigation problems and the Shuttle docking problem due to [Chrisman 92].

In *Chapter 6* we propose and describe various extension of the basic POMDP framework that can represent additional problem structure. The extensions of the framework were explored in context of the application of the POMDP framework to medical therapy planning, more specifically on the problem of management of patients with ischemic heart disease [Wong et al. 90]. The new structural features include: a combination of MDP and POMDP frameworks using factored process states with perfectly observable and hidden components; and hierarchical state variable space that restricts possible state variable value combinations (section 6.2.1). The additional structure makes it possible to cut down the complexity of the information state (section 6.4.1) used to solve planning and control problem and thus speed-up the problem solving routines. Other new features that allowed us to construct the prototype POMDP model for the ischemic heart disease problem are: factored transition and observation model represented using hierarchical version of the Bayesian belief network (section 6.2.3), factored cost model (section 6.2.5), actions with different discounts (section 6.2.7).

Chapter 7 summarizes the preceeding text, points out main issues related to POMDP framework, describes the contributions of our work, and discusses open problems, and future research objectives.

Chapter 2

Markov decision processes

The *Markov decision process (MDP)* [Bellman 57] [Howard 60] [Puterman 94] is a basic modelling framework often used in the area of planning in stochastic domains. A Markov decision process:

- is a controlled stochastic process;
- assumes that every process state depends only on the previous process state and not a history of previous states (Markov assumption);
- assigns rewards (or costs) to state transitions.

2.1 MDP model and MDP problem

Formally the Markov decision process is a 4-tuple (S, A, T, R) where:

- S is a finite set of world states;
- A is a finite set of action;
- $T : S \times A \times S \rightarrow [0, 1]$ defines the transition probability distribution $P(s|s', a)$ that describes the effect of actions on the world state;
- $R : S \times A \times S \rightarrow \mathcal{R}$ defines a reward model that describes payoffs associated with a state transition under some action.

A Markov decision process (MDP) is a useful abstraction that represents the dynamic behavior of a process under different actions. There are different variants of the basic MDP presented above. For example very often the model uses costs instead of rewards. In general costs can be viewed as negative rewards. They measure negative aspects of transitions.

A Markov decision process can be represented graphically using the influence diagram in figure 2-1. In the influence diagrams ([Howard, Matheson 84] [Schachter 86]):

- circles represent chance nodes and correspond to states of the controlled process in two consecutive time steps;
- rectangles stand for decision nodes that represent action choices;

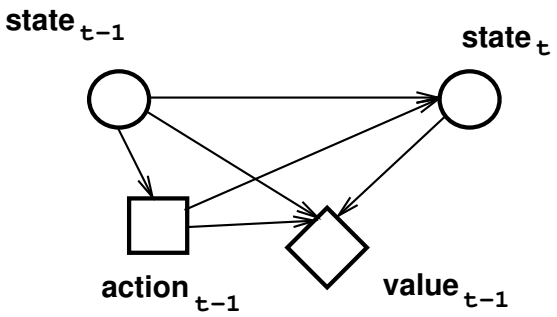


Figure 2-1: The influence diagram representing Markov decision process.

- diamonds stand for value nodes representing reward associated with transitions;
- directed links represent dependencies between individual components.

An influence diagram that represents temporal dependencies is also often called a *dynamic influence diagram*. It can be expanded over time by replicating its structure and creating a sequence (chain) of states, actions and value nodes. This is shown in figure 2-2.

2.1.1 MDP problem

A decision (control) problem within the MDP framework requires one to find an action or a sequence of actions for one or more states that optimizes some objective reward (cost) function. The objective function represents control objectives by combining the rewards incurred over time into a single quantity using various kinds of models (represented by a global value node in figure 2-2). Typically the objective function is additive and is based on expectations. The objective of control is then to find the rational choice of control actions, that is actions that lead to the maximum expected cumulative reward.

The most common kinds of models used in practice to combine rewards are:

- finite horizon models: maximize the expected reward for the next n steps:

$$\max E(\sum_{t=0}^{n-1} \gamma^t r_t)$$

  where r_t represents a reward acquired at time t and $\gamma geq 0$ corresponds to the multiplicative factor (discount factor) that scales rewards obtained in future;

- infinite horizon models:

  1. maximize expected discounted reward :

$$\max E(\sum_{t=0}^{\infty} \gamma^t r_t),$$

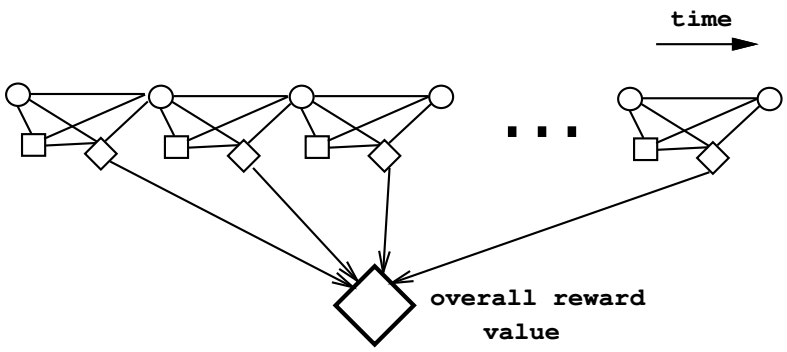


Figure 2-2: Expanded influence diagram representing MDP. The global value node represents a reward model that combines multiple one-step rewards.

where γ is a discount factor that satisfies: $0 \leq \gamma < 1$ [1];

2. maximize average expected reward per transition:

$$\max \lim_{n \to \infty} \frac{1}{n} \sum_{t=0}^{n} r_t;$$

- target state model: maximize expected reward (minimize expected cost) to some target state G.

Naturally one can imagine a whole spectrum of other models. For example one might want the control to reduce the risk of the transition to some state primarily and secondarily to decrease its expected discounted cost. This may correspond to the medical problem in which the state to be avoided is the death of the patient and where actions must be taken such that the risk of death is minimized in the first place and the well-being of the patient (represented by a lower cost) or economical cost are secondary. However our work will consider only two additive models: the n steps-to-go finite horizon model and the infinite discounted horizon model.

[1] A model very similar to the maximization of the expected discounted reward requires one to maximize expected discounted total reward, i.e. $\max \lim_{n \to \infty} E(\sum_{t=0}^{n} \gamma^t r_t)$, where $0 \leq \gamma < 1$ is a discount factor. Note that under some assumptions the limit and expectations can be exchanged and the result will be same for both models. In the following we will assume this holds. However in general the total reward model does not have to be solvable and also does not need to be equal to the expected reward.

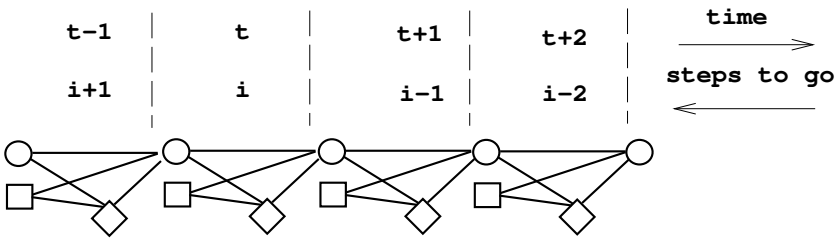


Figure 2-3: The relationship between the two schemes used to index control policies. The t index follows the time flow, while the i index goes in the opposite direction and represents steps to go.

2.1.2 Control functions and policies

Let μ denotes a *control function* that maps states to actions (i.e. $\mu : S \rightarrow A$) and let $\pi = \{\mu_0, \mu_1, \cdots\}$ be a *policy* that corresponds to a sequence of control functions. A sequence of control functions is also often referred to in the literature as a *strategy* or a *control plan*. Control functions and policies in the MDP framework describe a specific reactive behavior of the agent in various circumstances. The policies can be stored in tables that enumerate all possible situations the agent can wind up in.

Stationary and non-stationary policies

A policy $\pi = \{\mu, \mu, \cdots\}$ that has a fixed control function over time is called *stationary*. If control functions within the policy $\pi = \{\mu_0, \mu_1, \mu_2, \cdots, \mu_t, \cdots\}$ are allowed to vary over time steps, the policy is *non-stationary*. The optimal control policy is (see [Puterman 94]):

- non-stationary for the finite horizon model with n steps-to-go;
- stationary for the infinite discounted horizon model.

A control strategy for the finite horizon problem can be fully described by a finite n-step policy $\pi = \{\mu_n, \mu_{n-1}, \cdots, \mu_i, \cdots, \mu_1\}$, where μ_i represents the control action to use when i control steps remain to be done. Note that there are two indexing schemes one can use to describe finite horizon problem policies: one that follows the time flow and indexes control functions starting from time 0 and one that indexes control functions by counting steps to go and starts from n (steps to go). These two schemes are opposite of each other and the choice is simply a matter of convenience. The relation between the time and cost-to-go indexes is shown in figure 2-3. The basic relationship between the two is that if:

$$\mu_i = \mu_t \qquad \text{then} \qquad \mu_{i-1} = \mu_{t+1}.$$

In order to avoid confusion and differentiate between the two schemes we will always use t when referring to time indexing and use other indexes for the steps-to-go indexing.

The optimal policy for the inifinite horizon problem is stationary because in any state at any point in time the control agent faces an infinite number of steps to go and thus the optimal control function must be the same for any state.

Deterministic and stochastic policies

We have assumed so far that control functions are of the form: $\mu : S \rightarrow A$, that is that they assign actions to states deterministically. When the policy consists of such control functions it is called *deterministic*. However, in general a control function can assign actions to states nondeterministically according to some probability distribution. In such a case the policy is called *stochastic* and can be realized by a coin flipping machine. Note that a deterministic policy is a special case of a stochastic one.

In the following we will work only with deterministic policies. In fact it is possible to show that for discrete state MDPs with perfectly observable process states the optimal policy is always deterministic (see [Puterman 94]). This may not be the case in situations when process states are only partially observable and policy is constructed using observations or features that are different from process states (see [Singh et al. 94]).

2.1.3 Types of MDP problems

The basic MDP control problem requires one to determine the optimal policy. However, in many cases we are not always interested in finding the complete description of all optimal responses for all possible contingencies. Then, based on the scope and detail of the required solution, the MDP problem can consist of:

- finding the optimal control policy for all possible states;
- finding the sequence of optimal control functions for a specific initial state;
- finding the best control action (decision) for a specific initial state.

The problem formulation that requires one to find the optimal complete policy can be valuable in situations in which a control agent can be asked to solve the same problem with the same objective function repeatedly but from different initial situations.

On the other hand the problem that requires one to find the optimal control sequence only for a specific initial state is important in cases in which the agent always starts from the same initial state. The difference in this case is that control functions in the optimal solution do not need to be defined completely for every step. For example for the initial state s_{init}, μ_n^* only needs to contain the mapping from the initial state s_{init} to the optimal action a, μ_{n-1}^* only needs mappings from states that can be reached from the initial state by performing optimal action a in s_{init} , and μ_i^* only needs mappings from states reachable from s_{init} through a sequence of $n - i$ optimal actions. This can in some cases significantly reduce the amount of computation needed to find the optimal control.

Both of the above more general problems subsume the problem in which one is interested in finding the best control response (decision) for a single initial state. Although finding the optimal action may require the complete plan to be found, it is often the case that the decision about the best action can be made without evaluating all possible future situations and thus carried out in a more efficient way.

There are other variants of control problems that can be solved within the MDP framework, for example finding a partial k-step policy for some initial state. However the three types of MDP problem listed above are used most often, so they will be also the focus of our attention.

2.1.4 Real-time control

The objective of a control agent that acts in the world described by some MDP model is to repeatedly choose the action that is expected to result in the best overall performance, that is the action that maximizes the expected overall reward. Such an agent can be implemented using the decision problem solving procedure (finds the optimal action for a single state) over and over again. This approach has both its advantages and disadvantages. The main advantage of the approach is that it can be combined with various routines for adapting the underlying MDP model. Its disadvantage is that the time spent on computing the optimal action can lead to unacceptable delays, for example when the agent acts in some time critical environments.

The natural solution for the time critical application is to avoid the expensive on-line computation of the optimal response and try to precompute possible control responses off-line before they are used by a control agent. The off-line computation finds policies for one or more states. Once computed these can be stored in various forms: as lookup tables, as protocol like structures with conditional action sequences, or using various auxiliary structures, for example one that stores precomputed values of objective functions (so called value functions). In general the idea is that the precomputed result and the structure used to represent the policy should allow the agent to extract the control response sufficiently quickly.

In the following text we will focus our attention on issues related to the problem of finding optimal control solutions and we will not consider technical issues related to the choice of data structures used to store policies and the efficiency of such representations. However, when building a real-time agent one must also consider also the delays and the efficiency due to the policy representation.

2.2 Solving the MDP problem

There are numerous methods one can apply to solve control problems formulated within the MDP framework. The focus of the following is to describe the basic methods for solving complete policy problem for both finite and infinite discounted horizon criteria, and to explore some of their extensions and modifications. A good in-depth analysis of such methods can be found in [Puterman 94] or [Bertsekas 95]. Later in the chapter, methods that compute simpler or more restricted MDP problems more efficiently will be discussed.

2.2.1 Finite horizon problem

The objective of the n step horizon control problem is to find a policy that optimizes the additive reward model: $\max E(\sum_{t=0}^{n-1} \gamma^t r_t)$. A nice property of the additive model is that the overall expected reward for some control plan can be decomposed into the expected reward associated with the first control step and the expected reward for the remaining plan steps. Let V denote a *value function* $V : S \to \mathcal{R}$ representing the expected reward of some complete policy. Then because of the decomposability of the value function for an n steps-to-go policy $\pi_n = \{\mu_n, \mu_{n-1}, \cdots, \mu_1\}$ we can write:

$$V_n^{\pi_n}(s) = \rho(s, \mu_n(s)) + \gamma \sum_{s' \in S} P(s'|s, \mu_n(s)) V_{n-1}^{\pi_n - 1}(s') \tag{2.1}$$

where $\rho(s, \mu_n(s))$ corresponds to the expected reward incurred by performing first action $\mu_n(s)$ of the plan π_n in state s and $V_{n-1}^{\pi_n - 1}(s')$ corresponds to the expected reward associated with the

remaining $(n-1)$ steps of the plan π_n. $\rho(s,a)$ for a state s and an action a is computed as:

$$\rho(s,a) = \sum_{s' \in S} P(s'|s,a))R(s,a,s').$$

Our objective is to find a policy that optimizes the overall expected reward. This can be done using Bellman's principle of optimality [Bellman 57][2]. Using Bellman's principle, the optimal value function V^* for an n steps-to-go plan starting at state s is:

$$V_n^*(s) = \max_{a \in A} \rho(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V_{n-1}^*(s') \tag{2.2}$$

where $V_{n-1}^*(s')$ is the optimal value function for the $n-1$ step optimal plan. This implies that the optimal control function μ_n^* must be:

$$\mu_n^*(s) = \text{argmax}_{a \in A} \rho(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V_{n-1}^*(s'). \tag{2.3}$$

Q functions

The optimality equations can also be written using *action-value functions* or so called Q-functions. The action value function $Q^* : S \times A \rightarrow \mathcal{R}$ represents the expected reward associated with taking a fixed action from a specific state first and proceeding optimally afterwards. The relation between value and action value functions is:

$$V_n^*(s) = \max_{a \in A} Q_n^*(s,a)$$

$$Q_n^*(s,a) = \rho(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V_{n-1}^*(s')$$

where the last formula can be rewritten in pure Q form as:

$$Q_n^*(s,a) = \rho(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \max_{a' \in A} Q_{n-1}^*(s',a').$$

The introduction of an action value function has no special meaning on this place. However it will be used in the upcoming sections in some of the algorithms and therefore it was introduced here.

H mappings

In many cases it is easier to rewrite recursive equations 2.1 and 2.2 into a value function mapping form. Let B be a set of bounded real-valued functions V on S, $V : S \rightarrow \mathcal{R}$ and let h be a mapping $h : S \times A \times B \rightarrow \mathcal{R}$ such that:

$$h(s,a,V) = \rho(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V(s')$$

[2]Bellman's principle of optimality says that any tail subplan of the optimal plan must be also optimal. The proof of this is straightforward and is based on the fact that a plan with a suboptimal tail subplan cannot be optimal.

Let μ be an arbitrary control function. Then we can define a mapping $H^{\mu} : B \rightarrow B$ such that:

$$H^{\mu}V(s) = h(s, \mu(s), V),$$

and a mapping $H : B \rightarrow B$ such that:

$$HV(s) = \max_{a \in A} h(s, a, V).$$

Then using the value function mappings one can represent equation 2.1 as:

$$V_n^{\pi_n} = H^{\mu_n} V_{n-1}^{\pi_{n-1}},$$

and equation 2.2 as:

$$V_n^* = HV_{n-1}^*.$$

Finding the optimal n step policy

The n step control plan can be computed easily in a backward fashion using the dynamic programming approach. The dynamic program computes the optimal value and control functions for i steps-to-go from the optimal value function for $i-1$ steps-to-go:

$$V_i^*(s) = \max_{a \in A} \rho(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V_{i-1}^*(s')$$

$$\mu_i^*(s) = \text{argmax}_{a \in A} \rho(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V_{i-1}^*(s').$$

Using the above formulas repeatedly one can construct the complete solution policy backwards. That is, starting with a value function for 0 steps to go, one can compute the optimal value and control functions for 1 step to go, and then the optimal functions for 2 steps to go, and so on, up to n steps to go. The simple version of the dynamic programming computes a complete n steps-to-go policy in $O(n|A||S|^2)$ time.

2.2.2 Infinite discounted horizon problem

The objective of the infinite discounted horizon problem is to find a stationary policy that optimizes $\max E(\sum_{t=0}^{\infty} \gamma^t r_t)$ with γ being restricted to $0 \leq \gamma < 1$.

The optimal value and control function for an infinite discounted horizon must satisfy the fixed point equation:

$$V^*(s) = \max_{a \in A} Q^*(s, a) = \max_{a \in A} \rho(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^*(s'). \tag{2.4}$$

The equation can also be written using H mapping as $V^* = HV^*$. Once the optimal value function is known the optimal control (policy) can then be acquired:

$$\mu^*(s) = \text{argmax}_{a \in A} Q^*(s, a) = \text{argmax}_{a \in A} \rho(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^*(s'). \tag{2.5}$$

There are three basic approaches to find the optimal function for the infinite discounted horizon problems:

- value iteration
- policy iteration
- linear programming

The first two methods are iterative. They allow one to approximate the control policy. They also guarantee convergence to the optimal solution after a sufficient number of iterations. On the other hand, the linear programming approach converts the planning problem directly to a linear programming optimization problem.

Value iteration

The value iteration method [Bellman 57] finds the optimal or ϵ-optimal value function. The method builds on the fact that there is a unique fixed point value function V^* satisfying Bellman's equation:

$$V^*(s) = \max_{a \in A} \rho(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^*(s')$$

and that a simple value iteration method allows us to find it. Both of these results follow directly from properties of H mappings.

Let B is a set of real valued bounded functions on S, i.e. for $V \in B$, $V : S \to \mathcal{R}$. Let $\| V \| = \max_{s \in S} |V(s)|$ be a max norm. Then B together with the max (supremum) norm is a complete, normed linear space or *Banach space* (see [Puterman 94]). Assuming the discount factor $0 \leq \gamma < 1$, value function mappings H and H^μ correspond to isotone contraction mappings on B with a contraction factor γ.

Definition 1 *(contraction mapping) The mapping $H : B \to B$ is a contraction when for any two functions $U, V \in B$ the following holds:*

$$\| HV - HU \| \leq \beta \| V - U \|$$

with $0 \leq \beta < 1$ being the contraction factor.

Definition 2 *(isotone mapping) The mapping H is isotone when for any two functions $U, V \in B$ that satisfy $V(s) \leq U(s)$ for all $s \in S$, denoted $V \leq U$, holds: $HV \leq HU$.*

The proof that H and H^μ are isotone contractions is straightforward and can be found in [Puterman 94]. Knowing that H and H^μ are contraction mappings, one can directly apply the results of the Banach theorem.

Theorem 1 *(Banach theorem). Let B be a Banach space, $F : B \to B$ be a contraction mapping and let $(x_k)_k$ be a sequence with arbitrary initial point $x_0 \in B$, such that $x_k = F x_{k-1}$. Then:*
1. F has a unique fixed point solution x^* such that $Fx^* = x^*$
2. the sequence $(x_k)_k$ converges to x^*
3. for all k the following estimates hold:

$$\| x_k - x^* \| \leq \frac{\gamma^k}{1-\gamma} \| x_1 - x_0 \|$$

$$\| x_k - x^* \| \leq \frac{\gamma}{1-\gamma} \| x_k - x_{k-1} \|$$

$$\| x_k - x^* \| \leq \| x_{k-1} - x^* \|$$

The immediate consequences of the Banach theorem for H are:

- H has a unique fixed point value function solution, denoted V^*, i.e. $HV^* = V^*$.
- One can construct a sequence of value functions using a simple iteration method $V_k = HV_{k-1}$ that starts from an arbitrary value function V_0 and converges to the fixed point solution V^*.
- The precision of the value function approximation using the kth member of the sequence is given by simple error bounds provided by the theorem.

The same holds for H^μ and V^μ.

Therefore one can always guarantee the existence of the unique optimal value function solution $V^* = HV^*$ as well as the existence of a simple value iteration method that can be used to find it. Based on the provided error bounds one can also compute the minimum number of iteration steps to make in order to guarantee the required precision of the value function solution.

Theorem 2 *Let M be the maximum per step cost, let $0 \leq \gamma < 1$ be the discount factor and let ϵ be the required precision. Then the simple value iteration method, starting from $V_0(s) = 0$ is guaranteed to achieve required precision ϵ after k steps, where:*

$$k \geq \frac{\ln \epsilon(1-\gamma) - \ln M}{\ln \gamma}.$$

Proof. The proof exploits the fact that under the max norm, two consecutive value functions acquired by the iteration method are guaranteed to be lower than M, that is:

$$\| x_k - x^* \| \leq \frac{\gamma^k}{1-\gamma} \| x_1 - x_0 \| \leq \frac{\gamma^k}{1-\gamma} M$$

Then by setting:

$$\frac{\gamma^k}{1-\gamma} M \leq \epsilon$$

we can derive the minimum number of iterations needed to achieve the required precision. □

The minimum number of iterations computed using the above formula is usually very rough and not tight. In general case ϵ optimality can be reached sooner by examining the difference between the value functions computed for two consecutive steps. This is expressed in the following Bellman residual theorem [Puterman 94].

Theorem 3 *(Bellman residual) Suppose $V_k(s)$ and $V_{k-1}(s)$ differ by at most δ for every $s \in S$. Then both $V_k(s)$ and $V_{k-1}(s)$ never differ from V^* by more than $\frac{2\delta\gamma}{1-\gamma}$.*

The Bellman residual theorem provides a nice stopping criterion that iterative algorithms can use to compute ϵ optimal value function solutions. Such an algorithm is shown below. It outputs the value function V, such that:

$$| V(s) - V^*(s) | \leq \epsilon$$

holds for every state s. An ϵ-optimal value function can be then used to compute control function as:

$$\mu(s) = \text{argmax}_{a \in A} \rho(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)V(s').$$

Value iteration (MDP, γ, ϵ)
initialize $V(s)$ for all $s \in S$;
repeat
set $V'(s) \leftarrow V(s)$ for all $s \in S$;
set $V(s) \leftarrow \max_{a \in A}[\rho(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)V'(s')]$;
until $\mid V(s) - V'(s) \mid \leq \frac{\epsilon(1-\gamma)}{2\gamma}$ for all $s \in S$
return V;

The value iteration algorithm can come in different flavors. One obvious modification to the described basic version is to update the value function used in the iteration immediately with a new result and not to wait until the value function for all states is available. This modification is often referred to as the *Gauss-Seidel version of value iteration* and usually leads to faster convergence of the algorithm.

Policy iteration

An alternate approach to the computation of the optimal policy for the infinite discounted horizon problem is policy iteration. This method was suggested by Howard [Howard 60] and is based on the two computation steps performed iteratively:

- *value determination*: computes expected return for current (initially random) fixed policy;
- *policy improvement*: improves the current policy.

The method relies on the fact that for a fixed stationary policy it is easy to:

- compute the value function corresponding to such a policy (simply by solving a set of linear equations);
- improve the policy if it is suboptimal;
- decide if the policy is optimal.

This is based on two theorems, which are presented without proof (proofs can be found in [Bellman, Dreyfus 62]).

Theorem 4 *(Improvement theorem) Let μ and η be two control functions defining two stationary policies and let μ be chosen such that:*

$$V_\eta(s) \leq Q_\eta(s, \mu(s)) \text{ for all } s \in S.$$

Then it follows that μ is uniformly better than η, i.e.

$$V_\eta(s) \leq V_\mu(s) \text{ for all } s \in S.$$

Theorem 5 *(Optimality theorem) Let μ be a control function (policy), with associated value function $V_\mu(s)$ and action-value function $Q_\mu(s,a)$. If policy μ cannot be further improved using the policy improvement theorem, that is if*

$$V_\mu(s) = \max_{a \in A} Q_\mu(s,a) \qquad \textit{for all } s \in S,$$

then $V_\mu(s)$ and $Q_\mu(s,a)$ are unique optimal value and action value functions and μ is an optimal control function defining the optimal stationary policy.

An immediate consequence of the improvement theorem is that a policy constructed from the current policy by replacing all actions in the current policy with actions with better $Q_\mu(s,a)$ guarantees better results. This defines the improvement step. The consequence of the optimality theorem is that if the policy cannot be improved using the improvement step then it is optimal. This represents an optimality test. The following algorithm incorporates these steps, and represents the policy iteration method.

Policy iteration(MDP, γ)
  set μ to be an arbitrary control function defining policy π;
  repeat
    compute value function $V_\mu(s)$;
    compute action values $Q_\mu(s,a)$ for all $s \in S, a \in A$;
    set $\mu(s) \leftarrow \text{argmax}_{a \in A} Q_\mu(s,a)$ for all $s \in S$;
  until no change in μ is observed
  return control function μ;

The value determination phase of the algorithm computes the value function for a fixed stationary policy. The value function can be obtained by solving the set of linear equations of the form:

$$V_\mu(s) = \rho(s, \mu(s)) + \gamma \sum_{s' \in S} P(s'|s, \mu(s)) V_\mu(s'),$$

which can be solved by any of the available methods. The system of linear equations can become computationally expensive for larger state spaces. However, in order to improve the policy it is not necessary to compute the exact value function, and the improvement can be made based on a value function approximation. This idea is used in the version of the policy iteration procedure called *modified policy iteration* [Puterman 94]. Modified policy iteration uses value iteration techniques to approximate the value function in the policy evaluation step. This can be done because H^μ for any μ (that defines a policy) is a contraction mapping as shown above.

There are other possible modifications of the basic policy iteration procedure, for example policy iteration that eliminates suboptimal actions considered during the policy improvement using bounding techniques (see the discussion later in the chapter). A nice survey of policy iteration algorithms can be found in [Puterman 94].

Linear programming

The problem of finding the optimal control value function can be also reformulated as a linear programming task. The linear program can be solved in time polynomial in the number of variables and constraints (and precision), using either ellipsoid or Karmarkar's algorithms (see [Strang 86]). The basic linear program used in the infinite discounted horizon and the reward maximization is (see [Puterman 94] [Bertsekas 95]):

$$\text{minimize: } \sum_{s \in S} v_s$$

under the constraint:

$$v_s \geq \rho(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) v_{s'}$$

for all $s \in S$ and $a \in A$. Similarly one can construct the linear program for the problem with the minimization of costs (see [Puterman 94] [Bertsekas 95]).

Variable(s) v_s represent value function values associated with process states and the linear program attempts to find their optimal values $V^*(s)$. This is because: a value function for every state is no smaller than the immediate one step expected reward plus the expected reward for any possible process state continuation; minimizing the sum of value functions for all process states guarantees that values for the fixed point solution are found. Once the optimal value functions are found the optimal policy can be easily computed by selecting the action that minimizes the value function.

The above linear program consists of $|S|$ variables and $|S||A|$ constraints. It is also possible to construct a dual linear program that allows one to find the control function and that consists of $|S||A|$ variables and $|S|$ constraints (see [Puterman 94] [Littman et al. 95b]).

2.3 Forward methods for solving MDP problems

Methods we have discussed so far are suitable for computing value functions or control policies for all states. However, when one only needs to find the optimal control plan for a single initial state or to select the best control action for a single state, more efficient forward methods can often be used.

The main idea of forward methods is to identify states reachable from the initial state by unwinding the optimality formula in the forward fashion first (identification phase) and perform the computation backwards using only states reached in the identification phase. The effectiveness of forward methods depends mostly on the sparseness of the transition matrices. Thus, the more sparse the transitions, the better the chance forward methods improve the efficiency.

2.3.1 Computing optimal control plans for the finite horizon model

Forward methods can be applied to compute the n steps-to-go policy for a single initial state. In this context one can use an extension of the backward dynamic programming method, that:

- identifies all process states that need to be considered at every stage (forward phase);
- computes value and control functions backward only for those states that were reached in the forward phase.

States that need to be considered can be found in the forward fashion by simply tracking and marking all states reachable from the initial state. Then the computation of the optimal value and control functions is performed only for these states, leaving all others undefined.

The main advantage of the method is that it eliminates the computation of value functions at states that cannot be reached. The savings from it might be significant when one deals with a large model with a large number of states and sparse transition matrix.

2.3.2 Finding the optimal action for a single initial state

The decision problem, that seeks the optimal control action for a single initial state is the other problem for which forward methods are suitable. The problem is simpler than the above problem that requires us to find optimal choices for all reacheable states. This often allows us to construct simpler and faster problem-solving methods that focus on finding the single state control. The algorithms are best described using stochastic decision trees (see e.g. [Pearl 89]).

Decision tree

The decision tree depicts in the chronological order actions a control agent can make and subsequent outcomes of these actions that are governed by chance. An example of a decision tree is in figure 2-4. It consists of two types of nodes:

- decision nodes (rectangles);
- chance nodes (circles).

In the decision tree, decision nodes stand for process states, branches starting in decision nodes represent actions the agent might select, chance nodes represent states after the selection and branches emanating from the chance nodes represents possible stochastic outcomes following the action in the state. A complete decision tree represents: states that are reacheable from the initial state and action choices that lead to them. For MDPs, the decision tree structure can be used to compute optimal value function for some state as described in the basic value function formula:

$$V^*(s_t) = \max_{a \in A} Q^*(s_t, a) = \rho(s_t, a) + \gamma \sum_{s_{t+1} \in S} P(s_{t+1}|s_t, a) V^*(s_{t+1}) \tag{2.6}$$

where $V^*(s_t)$ and $Q^*(s_t, a)$ are values that can be associated with decison node s_t and chance node $[s_t, a]$ respectively. Thus the tree is best viewed as being constructed by a repeated unfolding of the value function formula.

Note the difference between the two graphical representations: dynamic influence diagrams and decision trees. The former serves to represent the model and its components, while the latter one represents how the solution for some specific state is computed.

The goal of the decision task is to select the optimal control action for the initial state that corresponds to the root of the tree. The best action choice is computed simply as:

$$\text{argmax}_{a \in A} Q(s_0, a).$$

The problem with a decision tree method that blindly unfolds the recursive formula is that the size of the tree can grow exponentially. This can lead to a significant inefficiency due to repeated or redundant computation. Therefore one needs a mechanism to restrict the size of the tree. In the following we will present two mechanisms that keep the size of the tree from growing large.

Using bounds for pruning suboptimal branches

The idea of pruning is simple and is based on the ability to compute bounds for the expected reward of any encountered state of the partially constructed decision tree. Then assuming that bounds are known for the leaves of the partially constructed decision tree, one can compute

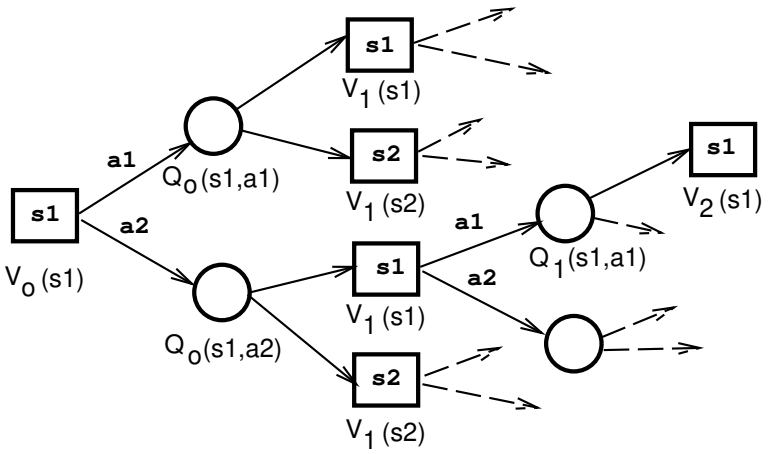


Figure 2-4: An example of a decision tree. Decision nodes (rectangles) correspond to process states and chance nodes (circles) represent process states with fixed action choices and possible stochastic outcomes. States are associated with value functions (V) and state-action choices with action-value functions (Q).

bounds for inner nodes of the decision tree simply by computing the expected reward for the best and worst case scenarios. The bounds at leaves of the decision tree can be computed easily. Assuming that:

$$M_u = \max_{a \in A} \max_{s \in S} \rho(s, a)$$
$$M_l = \min_{a \in A} \min_{s \in S} \rho(s, a)$$

stand for the maximum and minimum expected one step cost rewards respectively, bounds for the infinite discounted horizon problem and for any state s are:

$$ubound(s) = \frac{M_u}{1 - \gamma} \tag{2.7}$$
$$lbound(s) = \frac{M_l}{1 - \gamma}. \tag{2.8}$$

The computation at every decision node assumes that the action leading to the maximum expected reward is selected. But this means that when computed bounds for any two decision nodes do not overlap, one of them is guaranteed to be suboptimal and can be pruned from the decision tree. That is, whenever:

$$lbound(s_t, a_1) > ubound(s_t, a_2)$$

holds, we know that a_2 leads to a suboptimal solution and can be pruned from the decision tree.

The above criterion allows one to prune tree branches that are clearly suboptimal. However

one can also develop soft criteria that allow one to prune the decision tree branches based on the precision with which the decision at certain points needs to be made. The pruning rule in this case is :
Let ϵ be a precision with which the action at state s_t needs to be selected. Then whenever:

$$lbound(s_t, a_1) + \epsilon > ubound(s_t, a_2)$$

holds the decision tree branch corresponding to action a_2 can be pruned. The major problem in applying the soft pruning method is in allocating a precision factor to different branches of the tree and allowing soft pruning throughout the decision tree. This is because one is usually given only the precision error that is related to the decision at the root of the tree.

Basic method for the dynamic construction of the decision tree

It has been shown how one can use bounds to prune suboptimal branches of a decision tree that is only partially expanded. To exploit this feature, a strategy that incrementally expands a decision tree can be constructed. The strategy starts with a small initial decision tree, which is gradually expanded whenever the required decision cannot be made. Such a strategy allows one to avoid the unecessary exploration of large parts of the decision tree, and to prune suboptimal branches as soon as possible. We will refer to this strategy and its modifications as the *incremental expansion strategy* or *incremental decision tree strategy*. The simple breadth first version of this strategy is shown in the following algorithm.

Incremental expansion(MDP, γ, s_I, ϵ, V_L, V_U)
    initialize tree T with s_I and $ubound(s_I), lbound(s_I)$ using V_L, V_U;
    repeat until (single action remains for s_I or $ubound(s_I) - lbound(s_I) \leq \epsilon$)
        call Improve-tree(T, MDP, γ, V_L, V_U);
    return action with greatest lower bound as a result;

Improve tree(T, MDP, γ, V_L, V_U)
    if $root(T)$ is a leaf
        then expand root(T)
            set bounds $lbound$, $ubound$ of new leaves using V_L, V_U;
        else for all decision subtrees T' of T
            do call Improve-tree($T', MDP, \gamma, V_L, V_U$);
    recompute bounds $lbound(root(T)), ubound(root(T))$ for $root(T)$;
    when root(T) is a decision node
        prune suboptimal action branches from T;
    return;

The algorithm takes an MDP model, a discount factor γ, an initial state s_I, a precision parameter ϵ and value function bounds V_L and V_U used to initialize leaf nodes of the partially built decision tree. It returns an action that is guaranteed to be ϵ-optimal. The algorithm builds a decision tree T and improves bounds $ubound, lbound$ associated with nodes of the tree incrementally by calling subroutine *Improve tree*. It stops when ϵ-optimal action can be selected. This is when the bound difference for the root of the tree is less than ϵ or when only single action remains possible (all others were pruned). Bounds at leaves of the tree are always initialized using V_L and V_U that are computed e.g. using equations 2.7 and 2.8 for the infinite discounted horizon problem.

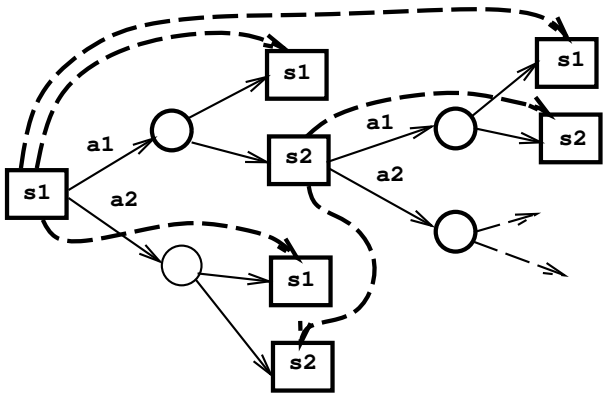


Figure 2-5: The elimination of recomputation by a result sharing.

Computing decisions using bound iteration

Although pruning can help eliminate some parts of the decision tree, this usually does not prevent one from the exploration of a large part of a decision tree. The major source of inefficiency is that the same tree substructure can occur repeatedly in two or more decision tree branches. The solution to this is to compute the result once. The idea of result sharing for the infinite discounted problem is shown on figure 2-5. Here decision nodes associated with common process states (e.g. $s1$) share substructures.

In the following we will describe one method that eliminates redundant recomputations. This method can be used to compute decisions for infinite discounted horizon problems. It is based on the same idea as value iteration, and uses the incremental expansion strategy with pruning. One difference between ordinary value iteration and this method is that the new method tries to iteratively improve value function bounds, and not the value function itself (hence bound iteration). Another difference is that value iteration works purely in a backward fashion for all possible states, while the decision tree method tries to iterate only over the states that are needed for the decision, that is, states reacheable by forward expansion.

The simplest version of bound iteration that uses a breadth-first expansion of the decision tree with pruning and repeated substructure elimination is described below.

Bound iteration(MDP, γ, s_I, ϵ, V_L, V_U)
    initialize tree T with s_I and $ubound(s_I), lbound(s_I)$ using V_L, V_U;
    repeat until (single action remains for s_I or $ubound(s_I) - lbound(s_I) \leq \epsilon$)
        set visited-set $VS = \{[s_I, lbound, ubound]\}$;
        call Improve-tree($T, VS, MDP, \gamma, V_L, V_U$);
    return action with greatest lower bound as a result;

Improve tree($T, VS, MDP, \gamma, V_L, V_U$)
    case
        $root(T) \in VS$: **set** new bounds for $root(T)$ from values in VS;
        $root(T)$ is a leaf: **expand** root(T);
                set bounds $lbound, ubound$ of new leaves to values from VS
                    or (if not there) from V_L and V_U;

otherwise: **set** $VS \leftarrow VS \cup \{[root(T), lbound, ubound]\}$;
for all decision subtrees T' of T
do call Improve-tree$(T', VS, MDP, \gamma, V_L, V_U)$;
recompute bounds $lbound(root(T)), ubound(root(T))$ for $root(T)$;
update record for $root(T)$ in VS;
prune suboptimal action branches from T;
return;

The bound iteration algorithm implements a gradual breadth-first expansion of the decision tree, and reuses bound results for shared substructures using the data structure visited-set VS. The way results are reused in this algorithm is illustrated in figure 2-5, assuming that the branch corresponding to an action $a1$ is expanded first. The algorithm stops when the solution is guaranteed to be ϵ-optimal or when the root of the decision tree has only one remaining action (all others were pruned).

2.4 Solving large MDP problems

We have pointed out that one is able to solve the planning problems in time polynomial in the size of the state space $|S|$ and action space $|A|$. This means that one can solve the planning problems efficiently with regard to the component space sizes. However, for many real world problems the state space size can become very large, and is itself subject to exponential growth.

The notion of state in many real world problems is defined usually through a set of state variables, each with a specific number of values it can take. Using such factored state representation, the total state space consists of all possible combinations of assignments of values to state variables and is exponential in the number of variables used. For example, for a simple case with n boolean state variables, the complete state space has 2^n states. Similarly, when an action space is defined through a set of m possible elementary actions that may or may not be performed simultaneously by an agent, the total number of different actions the agent can perform is 2^m.

Reducing the complexity of MDP definitions

Having large state and action spaces increases computational time and forces the designer of the model to provide huge transition matrices and reward models (an entry is needed for every possible combination of two states and an action). This problem is reminiscent of the problem in the 70s, where methods for handling uncertainty based on probabilities were considered inadequate because one was expected to define huge probability tables.

The complexity of an MDP definition can be reduced by exploiting additional structure, such as independence and conditional independence, or various regularities and restrictions that hold among the components of a factored MDP model. One might be able to define larger models using significantly fewer parameters by using factored model instead of a complete one.

Graphical models, like belief networks [Pearl 89] or dynamic influence diagrams [Schachter, Peot 92], let us represent dependencies between components of an MDP model in more detail. A simple example is shown in figure 2-6. Here a process state is represented using state variables A, B, and C and both transition and cost models are described in the factored form.

The dynamic influence diagram example does not cover all possible ways one might express structural properties and regularities of an MDP model. For example, parameters corresponding

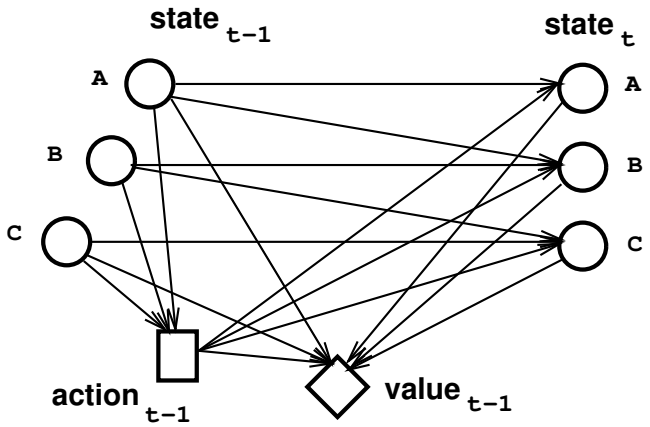


Figure 2-6: The influence diagram representing a factored MDP. A, B and C correspond to state variables. Dependencies between two consecutive states ($state_{t-1}$ and $state_t$) are now described using state variable dependencies.

to transition probabilities in the factored representaton can be expressed using decision trees or rules that map propositions constructed from state variable values to state variable values (with associated probabilities and possibly also costs). The advantage of such representations is that they reduce the size of the model description by representing relevant dependencies and excluding irrelevant ones. They can be viewed like compression techniques for sparse or high regularity parameter matrices.

Solving problems with large MDPs

There are two approaches one can use to simplify the computation of complex MDP models:

- exploitation of the additional model structure that makes independences or regularities among model components explicit;
- approximation of the model, where irrelevant features of the model are abstracted away.

Solving problems by exploiting MDP structure

The first approach is based on the exploitation of additional model structure, for example a graphical model explicitly represents dependencies and independences that hold among state variables that describe the process state. This approach does not change the content of the model, so the solution obtained is the same as one would achieve using the classical MDP model with a flat state space. Moreover, the solution plan for a structured model can often be described in a more compact way compared to the complete description that enumerates all possible state variable value combinations.

The solution policy for a factored MDP can be represented more compactly using a set of control rules. Every rule consists of a proposition part that lists a set of state variable values and an action part that specifies a control choice to be performed whenever the proposition is

satisfied:

$$\mu_i = \{< \varphi_i^k \rightarrow a_i^k >\}$$

where φ_i^k stands for the rule proposition and a_i^k is the action associated with it. The complexity of the rule set definition can usually be reduced by representing them through classification (decision) trees or decision lists with actions associated with their leaves.

Factored MDPs with additional structure can be solved using specialized procedures that take advantage of the structure and output structured policies and/or value functions (see [Puterman 94]). A method for computing infinite horizon problems that uses structured control and value functions is called *structured policy iteration*. This method was applied by [Boutillier et al. 95] for example. The main features of the approach are:

- MDP model is represented in a factored form and with additional structure (independences, regularities);
- value and control functions are expressed compactly using decision trees;
- value determination and policy improvement stages work directly with structured policies and structured value functions.

Approximations using model simplifications

One can compute control policies while avoiding the need to work with complete state space by exploiting regularities in the MDP definitions. Unfortunately, many problems do not exhibit perfect regularities that allow the problem to be solved and represented efficiently. However, in a large number of control problems, there are usually features that are less relevant, and that do not influence the quality of the final solution dramatically. Then, one would expect to get a good solution when such features are ignored and only relevant features are accounted for in the computation and in the resulting solution. This idea is the basis of approximation algorithms.

In general there are two methods researchers suggest for the purpose of approximation:

- model reduction (e.g. [Bertsekas 95], [Boutillier, Dearden 94]);
- decomposition [Dean, Lin 95].

The first approach is based on creating a new simpler MDP model that simplifies the original model by reducing the size of the state and/or action spaces. The reduction in the complexity of the model then allows for faster approximate solutions by trading off accuracy for speed. Alternatively, one can try to combine computation steps performed with complete and reduced models as suggested by [Bertsekas 95].

The reduced MDP model can be supplied completely or partially by the designer of the system or can be computed automatically by dropping the least relevant parts of the model. The MDP can be defined by the designer using *feature* or *aggregate* states and probability distributions mapping the new aggregate states to original model states $P(s|s^{Agg})$ [Bertsekas 95]. Using the conditional probability one can compute components of the new transition probability matrix as:

$$P(s_1^{Agg}|s_2^{Agg}, a) = \sum_{s \in S} P(s|s_2^{Agg}) \sum_{s' \in s_1^{Agg}} P(s'|s, a).$$

Alternatively one might construct a simpler MDP model with aggregate states that uses upper and lower bounds on transition probabilities and that does not require priors on states $P(s|s^{Agg})$

to be defined. Such an approach was pursued by [Dean, Givan 97] [Dean et al. 97] who also devised techniques to extract simpler models for factored MDPs.

Note that the computation of the new simpler model from the old one may require a significant amount of time. If the model reduction is performed during problem-solving, the overhead time spent on the reduction itself needs to be added to the overall running time. Then, if the complexity of the computation associated with the transformation of the model is comparable to the computation of the complete MDP the use of model reduction to solve the problem is completely unjustified.

The approximation through decomposition method [Dean, Lin 95] divides the complete state space into a collection of smaller state space regions with stronger links between intraregion states and weaker or limited links between interregion states. Regions are expected to consist of a small number of state variables that are assumed to be relevant only within the region and can define local policies. Different regions are then treated as a states of the higher level process, with actions corresponding to the lower level local policies. The approximate solution is then acquired by applying the divide and conquer strategy that breaks down the large MDP problem to smaller problems on both higher and lower levels. These are subsequently solved, combined and iteratively improved.

2.5 Summary

The Markov decision process (MDP) framework is a framework commonly used for representing and modelling control problems in stochastic dynamic domains. The basic MDP model assumes a process with a finite state space. Various problem-solving methods can be used to obtain optimal control solutions for such a model. The problem solving methods are: dynamic programming for the finite horizon case; value iteration, policy iteration, and linear programming for the infinite dicounted horizon case. Whenever the optimal decision for a single initial state is sought and transitions in the MDP are sparse the problem-solving can often be sped up using forward decision tree methods.

The main challenge for future reasearch in MDPs is to model and solve MDPs with large or continuous state spaces. The advances and new results in the neuro-dynamic programming (see [Bertsekas, Tsitsiklis 96]), and graphical modelling and associated probabilistic reasoning methods (see [Lauritzen 96]) that take advantage of independences and regularities between model components are of high importance in this respect.

The objective of this chapter was to summarize the MDP framework, basic methods for solving control and decision problems within it. The MDP framework is introduced mostly to simplify the explanation of more complex POMDPs that are the central topic, as many of the solution methods developed for the MDP are directly applicable or very similar to methods used to solve POMDP problems.

Chapter 3

Partially observable Markov decision process

The Markov decision process framework models a controlled stochastic process with perfectly observable states. This represents the situation in which a control agent can be uncertain about possible outcomes of its actions, but still able to verify the resulting state once the action is completed. That is, there is no uncertainty with regard to what state the agent currently is, though there is an uncertainty with regard to where it can be after the next action is taken.

One can easily imagine the situation in which the agent cannot observe the process state directly, but only indirectly through a set of noisy or imperfect observations. The feature of partial observability can be important in many real world problems. For example, a robot planning its route or deciding about what action to take usually works with noisy sensory information; in the medical area, the physician often needs to decide about the treatment based on available findings and symptoms while being uncertain about an underlying disease. In all such cases the perceptual information need not align with and imply the actual world state with certainty. Then the agent that acts in environments with imperfect state information may face uncertainty from the two sources:

- uncertainty about the action outcome;
- uncertainty about the world state due to imperfect (or partial) information.

Observations may not be costless. Often they can require a special action to be taken before they are enabled and this action might have both cost or transitional effect. The actions that enable observations are called *investigative actions*. The main purpose of performing investigative actions is to narrow the uncertainty about the world state, for example by performing a special test revealing more information about the ongoing patient's disease process, or using camera surveillance in order to detect the current position of the robot. Therefore when making the decision about an investigative action one needs to carefully consider both benefits and costs associated with performing it. For example, some investigative actions in medicine although very helpful in diagnosing underlying problems can be very risky and costly due to their invasivenes.

The presence of partial observability in the environment, as well as the capability of an agent to perform investigative actions have a major impact on how planning procedure must work. The reason for this is that:

- in order to find an optimal control one should account for imperfect observability now and in future steps;
- during planning, one must consider the cost and benefits of both control and investigative actions.

In the following we will focus on the modelling framework that represents action outcome nondeterminism, imperfect observability as well as investigative actions. The modelling framework is called *Partially observable Markov decison process (POMDP)* [Astrom 65] and it is best viewed as a further extension of the MDP framework.

3.1 Partially observable Markov decision process

More formally, partially observable Markov decision process is defined as (S, A, Θ, T, O, R) where:

- S corresponds to a finite set of world states;
- A is a finite set of actions;
- Θ is a finite set of observations;
- $T : S \times A \times S \rightarrow [0, 1]$ defines the transition probability distribution $P(s|s', a)$ that describes the effect of actions on the state of the world;
- $O : \Theta \times S \times A \rightarrow [0, 1]$ defines the observation probability distribution $P(o|s, a)$ that models the effect of actions and states on observations;
- R corresponds to the reward model $S \times A \times S \rightarrow \mathcal{R}$ that models payoffs incurred by state transitions under specific actions (alternate formulation may include costs that correspond to negative rewards).

The influence diagram describing the partially observable Markov decision process is shown in figure 3-1. The main distinction between fully observable MDPs and POMDPs is in the information one uses to select an action. In the MDP case actions are selected using process states that are always known with certainty, while for the POMDP, actions are based only on the available information that consists of previous observations and actions. Note that the observation model as defined makes it possible to condition observations on both actions and process states. This allows one to model investigative actions in the same way as other control actions.

The standard observation model (figure 3-1) assumes that observations depend on a previous action and a current process state, that is, O always defines $P(o_t|s_t, a_{t-1})$ relative to t. However, while modeling some decision and control problems one often needs to use different observation models that fit better the real world, for example one may need to model observation delays. These models can be very important in medical decisions in which test results are often not available immediately and are delayed (thus they refer to past patient states). One of the topics of our work is to explore some of these more complex observation models.

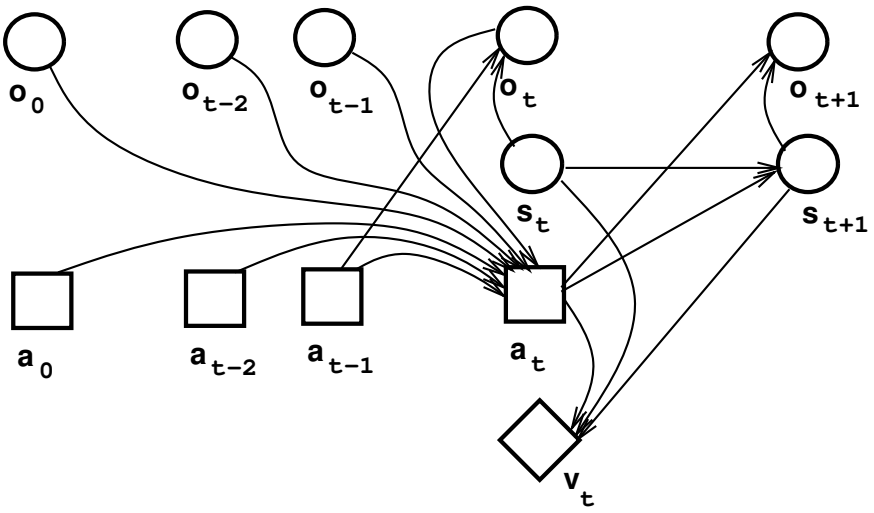


Figure 3-1: Influence diagram describing the POMDP model.

3.2 Control in partially observable domains

The major difference between MDP and POMDP is that in the POMDP the underlying process state is not known with certainty and can be only guessed based on past observations, actions and any prior information available. Therefore we need to differentiate between the true process state and the *information (or perceived) state* that captures all things important and known about the process.

3.2.1 Information state

An *information state* represents all information available to the agent at the decision time that is relevant for the selection of the optimal action. The information state consists of either a complete history of actions and observations or corresponding sufficient statistic. A sequence of information states defines a Markov controlled process in which every new information state is computed as a function of the previous information state, the previous step action and new observations seen:

$$I_t = \tau(I_{t-1}, o_t, a_{t-1})$$

where I_t and I_{t-1} denote new and previous information states. The process defined over information states is also called the *information-state Markov decision process* or *information-state MDP*. In priciple one can always reduce the original POMDP into the information-state MDP. The relation between the components of the POMDP model and its information state as well as a reduction of the model to information-state MDP is shown in figure 3-2.

Complete information state

The easiest way to represent an information state is to use all information available to the agent since the beginning (time $t = 0$) as shown in figure 3-1. Then information consists of

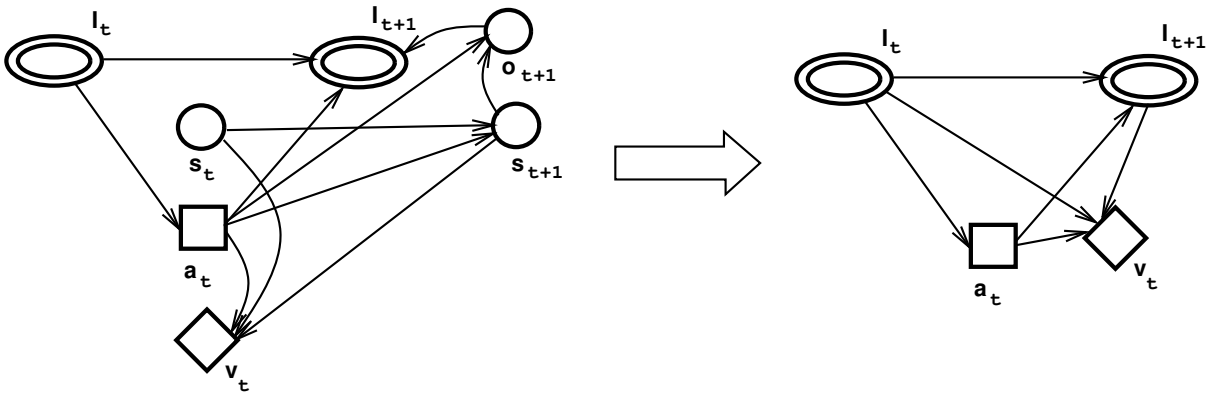


Figure 3-2: Influence diagram for the POMDP model with information states and corresponding information-state MDP.

a complete history of actions and observations made; in other words it corresponds to the *complete information state (vector).*

Definition 3 *(Complete information state (vector)) The information state I_t for time t is called complete (denoted I_t^C) when it consists of all information available to the agent before the action at time t is made. The complete information state consists of:*

- *prior belief on states at time* 0*;*
- *all observations available up to time t;*
- *all actions performed before time t.*

Note that the complete information state process satisfies trivially a Markov property. That is, any new information state can be expressed as a function of the previous information state, the previous action and the new observation. The update function is then simply implemented by adding the action and the new observation to the previous step information state.

Representing information states with sufficient statistics

The main problem with the complete information state is that it is expanding its size with elapsed time. This may be a major drawback, especially in the case where we are interested in computing and representing solutions to infinite horizon planning problems. A slightly different problem of control solution representability using the complete information vector with regard to planning can be due to the existence of the infinite size subspace corresponding to the prior belief at time 0.

The expanding dimension of complete information vectors is one of the major hindrances to both the computation of the value function as well as representation of control plans (policies). This problem can be resolved by replacing complete information states with quantities that represent *sufficient statistics* with regard to control (see for example [Bertsekas 95]). These quantities satisfy the Markov property and preserve the information content of the complete state that is relevant for finding the optimal control.

Definition 4 *(Sufficient information state process) Let $\mathcal{P} = \{I_0, I_1, \cdots, I_t, \cdots\}$ be a sequence of information vectors describing the information process. Then $\mathcal{P}$ is a sufficient information process with regard to the optimal control when for every component I_t in $\mathcal{P}$ holds:*

$$I_t = \tau(I_{t-1}, o_t, a_{t-1}),$$

$$P(s_t|I_t^C) = P(s_t|I_t),$$

$$P(o_t|I_{t-1}^C, a_{t-1}) = P(o_t|I_{t-1}, a_{t-1})$$

where I_{t-1} and I_t are sufficient information states, I_t^C and I_{t-1}^C are complete information states, o_t is an observation that became available at time t, and a_{t-1} is an action made at time $t-1$.

The main reason to use sufficient information states is that they can be significantly smaller and of non-expanding dimension and still allow one to compute optimal value and control functions. On the other hand the update of information states is usually more complex compared to the updating of complete histories. The sufficient information state can be used not only for optimization but also to encode a control plan (policy). Such a plan then requires the plan executor to update sufficient statistics at every step, which may cause a slight delay in the overall response time compared to the case when one works with complete histories, encoded, for example, as control trees [Cassandra 94] [1]. However, in many applications the delay due to information state update should not play a major role.

Belief states as sufficient information states

The quantity often used as a sufficient statistic for planning and control in POMDPs is the *belief state (or belief vector)*. The belief state assigns probability to every process state and reflects the extent to which states are believed to be present. The belief vector b_t at time t corresponds to:

$$b_t(s) = P(s|I_t^C)$$

where I_t^C is a complete information vector at time t.

Although one cannot guarantee that a belief state corresponds to the sufficient information vector for an arbitrary POMDP model, a large number of POMDP models used in practice (including standard POMDPs) falls into the class of *belief space POMDPs*. The major advantages of a belief information state are that it is defined over a finite number of process states and that it is relatively easy to work with. This is mostly due to nice properties satisfied by value functions defined for belief state MDPs. We will be discuss them later in this chapter.

3.2.2 Value functions in POMDP

Value function formulas we derived for the fully observable Markov model can be applied directly to the information-state MDP. For example n steps-to-go value function for some fixed plan $\pi_n = \{\mu_n, \mu_{n-1}, \cdots, \mu_i, \cdots, \mu_1\}$ corresponds to:

$$V_n^{\pi_n}(I_n) = \rho(I_n, \mu_n(I_n)) + \gamma \sum_{I_{n-1}} P(I_{n-1}|I_n, \mu_n(I_n)) V_{n-1}^{\pi_{n-1}}(I_{n-1}) \tag{3.1}$$

[1] The control (policy) tree [Cassandra 94] is best viewed as a collapsed decision tree with fixed action choices that the agent follows under different observations.

where μ_n is a control function defined over the complete information vector space, I_n and I_{n-1} are information states for n and $n-1$ steps-to-go, $\rho(I_n, \mu_n(I_n))$ is an expected one step reward from performing action $\mu_n(I_n)$ in I_n and $V_{n-1}^{\pi_{n-1}}(I_{n-1})$ is an expected reward associated with the remaining steps of the plan. Expected one step cost for an information state I_n and an action a is equal to:

$$\rho(I_n, a) = \sum_{s \in S} \rho(s, a) P(s|I_n).$$

A next step information state I_{n-1} is acquired from the current state using the Markov update function τ:

$$I_{n-1} = \tau(I_n, o, a).$$

This means that there are at most $|\Theta|$ following information states for every action and initial information state. The restricted number of observations allows us to rewrite the value function equation 3.1 more compactly by summing over all possible observations:

$$V_n^{\pi_n}(I_n) = \sum_{s \in S} \rho(s, \mu_n(I_n)) P(s|I_n) + \gamma \sum_{o \in \Theta_{next}} P(o|I_n, \mu_n(I_n)) V_{n-1}^{\pi_{n-1}}(\tau(I_n, o, \mu_n)) \tag{3.2}$$

where Θ_{next} stands for all possible observations following $\mu_n(I_n)$ in I_n. Note that for a general POMDP (which can include observation delays), Θ_{next} represents a set of observations available at $n-1$ steps to go and does not need to correspond to Θ. Θ_{next} is thus best viewed as a function of I_n and a: $Next(I_n, a)$.

Based on the fixed policy result, we can construct the optimal value function for the finite n steps-to-go problem as:

$$V_n^*(I_n) = \max_{a \in A} \sum_{s \in S} \rho(s, a) P(s|I_n) + \gamma \sum_{o \in \Theta_{next}} P(o|I_n, a) V_{n-1}^*(\tau(I_n, o, a)). \tag{3.3}$$

That is, the maximum expected reward for the information state I_n is computed recursively by summing an expected one step reward and an expected reward associated with the rest of the plan. The optimal control function μ_n is then:

$$\mu_n^*(I_n) = \text{argmax}_{a \in A} \sum_{s \in S} \rho(s, a) P(s|I_n) + \gamma \sum_{o \in \Theta_{next}} P(o|I_n, a) V_{n-1}^*(\tau(I_n, o, a)).$$

Similarly, the fixed point formula for the infinite discounted horizon problem is:

$$V^*(I) = \max_{a \in A} \sum_{s \in S} \rho(s, a) P(s|I) + \gamma \sum_{o \in \Theta_{next}} P(o|I, a) V^*(\tau(I, o, a)) \tag{3.4}$$

and the optimal control function is:

$$\mu^*(I) = \text{argmax}_{a \in A} \sum_{s \in S} \rho(s, a) P(s|I) + \gamma \sum_{o \in \Theta_{next}} P(o|I, a) V^*(\tau(I, o, a)).$$

3.2.3 Value function mappings

Basic value function equations can be written also in the value function mapping form. Let B be a set of real valued bounded functions $V : \mathcal{I} \to \mathcal{R}$ defined on the information vector space

$\mathcal{I}$, and let $h : \mathcal{I} \times A \times B \to \mathcal{R}$ be defined as:

$$h(I, a, V) = \sum_{s \in S} \rho(s, a) p(s|I) + \gamma \sum_{o \in \Theta_{next}} P(o|I, a) V(\tau(I, o, a)).$$

Then we can define the value function mapping $H^{\mu_i} : B \to B$ such that:

$$H^{\mu_i} V(I) = h(I, \mu_i(I), V),$$

and the value function mapping H such that:

$$HV(I) = \max_{a \in A} h(I, a, V).$$

Equation 3.1 can be expressed using the value function mapping as:

$$V_n^{\pi_n} = H^{\mu_n} V_{n-1}^{\pi_{n-1}}$$

and equations 3.3 and 3.4 as:

$$V_n^* = HV_{n-1}^* \qquad \text{and} \qquad V^* = HV^*.$$

The important property of H and H^μ mappings is that they are isotone. That is, for any two functions U, V satisfying $V \leq U$ holds: $HV \leq HU$. For the infinite discounted horizon (discount factor $0 \leq \gamma < 1$) mappings H^μ and H are contraction mappings under the max (or supremum) norm $\| V \| = \max_{\mathcal{I}} |V(I)|$. More specifically it holds that:

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

The proofs are shown below and are based on [Heyman, Sobel 84] and [Puterman 94].

Theorem 6 *(Isotonicity of H mapping) H mapping for $\gamma \geq 0$ is isotone. That is for any two functions U, V satisfying $U \leq V$ holds: $HU \leq HV$.*

Proof. Let I be an arbitrary information state. Then we can write:

$$\begin{aligned}
HU(I) &= \max_{a \in A} \rho(I, a) + \gamma \sum_{o \in \Theta_{next}} P(o|I, a) U(\tau(I, o, a)) \\
&= \rho(I, a^*) + \gamma \sum_{o \in \Theta_{next}} P(o|I, a^*) U(\tau(I, o, a^*)) \\
&\leq \rho(I, a^*) + \gamma \sum_{o \in \Theta_{next}} P(o|I, a^*) V(\tau(I, o, a^*)) \\
&\leq \max_{a \in A} \rho(I, a) + \gamma \sum_{o \in \Theta_{next}} P(o|I, a) V(\tau(I, o, a)) \\
&= HV(I).
\end{aligned}$$

As the above inequality holds for any state I, $HU \leq HV$ follows. □

Theorem 7 *(Contraction property) H with a discount factor $0 \leq \gamma < 1$ is a contraction under the max norm.*

Proof. Assume two value functions U, V. Let I be an arbitrary information state, and assume that $HU(I) \leq HV(I)$ holds. Also assume that a^* is an action that optimizes $HV(I)$, i.e.:

$$a^* = \text{argmax}_{a \in A} \rho(I,a) + \gamma \sum_{o \in \Theta_{next}} P(o|I,a) V(\tau(I,o,a)).$$

Then we can write:

$$\begin{aligned}
0 &\leq HV(I) - HU(I) \\
&\leq \rho(I,a^*) + \gamma \sum_{o \in \Theta_{next}} P(o|I,a^*) V(\tau(I,o,a^*)) - \rho(I,a^*) - \gamma \sum_{o \in \Theta_{next}} P(o|I,a^*) U(\tau(I,o,a^*)) \\
&= \gamma \sum_{o \in \Theta_{next}} P(o|I,a^*)[V(\tau(I,o,a^*)) - U(\tau(I,o,a^*))] \\
&\leq \gamma \sum_{o \in \Theta_{next}} P(o|I,a^*) \parallel V - U \parallel \\
&= \gamma \parallel V - U \parallel .
\end{aligned}$$

As max norm is symmetrical, the same result can be derived for the case when $HU(I) \geq HV(I)$. But then taking the maximum over all information states I we can write:

$$\parallel HV - HU \parallel \leq \gamma \parallel V - U \parallel,$$

that is H is a contraction mapping under the max norm. □

Isotonicity and contraction will be extremely important for the design of exact and approximation methods. For example, the contraction property guarantees the unique optimal solution (fixed point) for infinite discounted horizon problem and convergence of exact value iteration algorithm to it.

3.3 Constructing information state MDPs for different POMDP models

A POMDP model can be converted into an information state MDP. Information states can be represented trivially by complete histories or appropriate sufficient statistics. The focus of this section is to explore how one can construct appropriate sufficient information states for different observations models.

3.3.1 POMDP with standard (forward triggered) observations

A model used frequently in the POMDP literature (hence standard) assumes that an observation depends solely on the current process state and the previous action. This situation is illustrated in figure 3-3. The observation model O then in fact describes $P(o_t|s_t, a_{t-1})$ for time t. Since an observation is related to the state that results from the action that also triggered (induced) the observation, we will refer to this model as to the model with *forward triggered observations.*

The important feature of POMDPs with standard observation models is that information state MDP is sufficiently represented using belief states. The sufficient information state process by definition should satisfy the following:

1. Belief states satisfy the Markov property, that is, the next belief state can be computed from the previous belief, previous action and new observation as $b_t = \tau(b_{t-1}, o_t, a_{t-1})$.

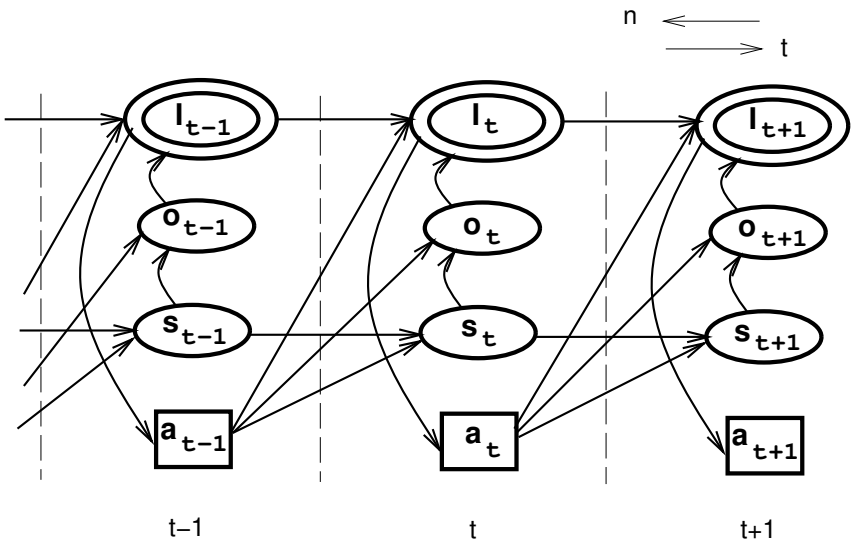


Figure 3-3: POMDP with standard (forward triggered) observation model.

2. An information state at time t should be sufficient to compute the belief state at time t, $P(s_t|I_t) = P(s_t|I_t^C) = b_t(s_t)$.

3. $P(o_t|I_{t-1}^C, a_{t-1}) = P(o_t|b_{t-1}, a_{t-1})$.

The Markov property of the belief state process holds because a belief state b_t can be computed from the belief state b_{t-1}, action a_{t-1} and observation o_t. The belief update that implements the transition function τ is:

$$\begin{aligned} b_t(s) &= \tau(b_{t-1}, o_t, a_{t-1})(s) \\ &= P(s|o_t, a_{t-1}, b_{t-1}) \\ &= \beta P(o_t|s, a_{t-1}) P(s|a_{t-1}, b_{t-1}) \\ &= \beta P(o_t|s, a_{t-1}) \sum_{s' \in S} P(s|a_{t-1}, s') b_{t-1}(s') \end{aligned} \tag{3.5}$$

where β is a normalizing constant and is equal to:

$$\beta = 1/P(o_t|a_{t-1}, b_{t-1}) = 1/\sum_{s \in S} P(o_t|s, a_{t-1}) \sum_{s' \in S} P(s|a_{t-1}, s') b_{t-1}(s').$$

The next conditions hold as well: $P(s_t|I_t^C) = b_t(s_t)$ trivially, and $P(o_t|I_{t-1}^C, a_{t-1}) = P(o_t|b_{t-1}, a_{t-1})$ follows because observations made at time t depend solely on the process state at time t and the action a_{t-1}.

This shows that belief states are sufficient to represent information states for the standard POMDP models. Thus standard POMDP models belong to the class of belief space POMDPs

and the optimal value function equation can be directly rewritten using belief states:

$$V_n^*(b_n) = \max_{a \in A} \sum_{s \in S} \rho(s,a) b_n(s) + \gamma \sum_{o \in \Theta_{next}} P(o|b_n, a) V_{n-1}^*(\tau(b_n, o, a)). \tag{3.6}$$

The computation of a new belief state always depends on the preceeding belief state, new observation and previous action. To bottom out the updating machinery we start with a prior belief over all initial process states, that is, a probability distribution over process states at time $t = 0$. Once we know the prior belief, we can compute subsequent belief states easily using the belief update formula.

3.3.2 POMDP with backward triggered observations

In the standard (forward triggered) POMDP model (figure 3-3) an observation at time t is triggered by an action a_{t-1} at time $t-1$, and is related to the process state s_t at time t. However this model may not be the best for all real world domains and we can consider other observation models as well.

One possible model corresponds to the observation model in which an action a_t performed at time t causes an observation about the process state s_t to be made (see figure 3-4). That is, the action performed at time t enables the observation that refers to the "before action" state. We will refer to such an observation model as to the model with *backward triggered observations*. Although the model seems to defy laws of causality and time, it may be more suitable for some domains than the model with forward triggering. This is because the forward model may suffer from the complementary problem: when action is actually responsible for the observation, then after the action is finished the observation made does not have to refer to the "after action" state. The whole problem is caused by modelling continuous domains by time discretization. Then the choice of the model boils down to the question of which state is better approximated by a new observation: the state that occured after or before the action.

Assuming that actions always delimit discrete time steps, observations in the backward observation model are always delayed one time step. Despite this feature that makes the model different from the standard observation model, one can show that also now the information state MDP can be constructed using belief information states.

The belief update for an action a_{t-1} and an observation o_{t-1}^t that is related to the state at time $t-1$ but observed (made available) at time t is:

$$b_t(s) = \beta \sum_{s' \in S} P(s|s', a_{t-1}) P(o_{t-1}^t|s', a_{t-1}) b_{t-1}(s')$$

where β is a normalizing constant and is equal to:

$$\beta = 1/\sum_{s' \in S} P(o_{t-1}^t|s', a_{t-1}) b_{t-1}(s').$$

The other two prerequisites of the information state process are satisfied as well. The second one is trivial again and the third prerequsite ($P(o^t|I_{t-1}^C, a_{t-1}) = P(o^t|b_{t-1}, a_{t-1})$) holds since the observation made at time t depends solely on the state at time $t-1$ and an action a_{t-1}.

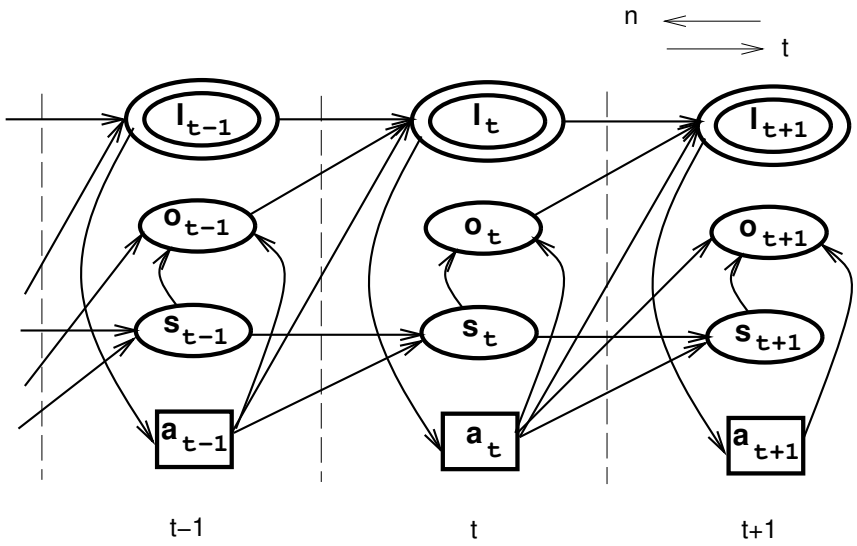


Figure 3-4: POMDP with simple (backward triggered) observation model.

3.3.3 POMDP with the combination of forward and backward observations

Two previous models can be combined into the POMDP with forward and backward observations. This model's basic structure is shown in the figure 3-5. The observation model does not consist of one monolithic set of observations but rather of the two groups of observations. One group is triggered in the forward and the other in the backward fashion. Using the similar notation to that introduced above, observations at time t are split into those related to the state at time t, o_t^t, and those related to the previous state, o_{t-1}^t. Futher, we assume the observations associated with the same state are independent given that state.

Interestingly, this model can be also converted to the information state MDP with belief states. To show that a belief state at time t must be Markov updateable. Let b_{t-1} stand for the belief state at time $t-1$, a_{t-1} be an action performed at time $t-1$, and o_{t-1}^t and o_t^t be observations made at time t that are related respectively to a state at $t-1$ and t. Then a new belief vector b_t at time t is computed as:

$$b_t(s) = \beta P(o_t^t|s, a_{t-1}) \sum_{s' \in S} P(o_{t-1}^t|s', a_{t-1}) P(s|s', a_{t-1}) b_{t-1}(s') \tag{3.7}$$

where β is a normalizing constant equal to:

$$\beta = 1/\sum_{s \in S} P(o_t^t|s, a_{t-1}) \sum_{s' \in S} P(o_{t-1}^t|s', a) P(s|s', a_{t-1}) b_{t-1}(s').$$

The derivation of the update formula (not shown here) exploits the independence between forward and backward observations given the underlying process state. Similar to both forward and backward observation models, the combination of the two satisfies the third condition as

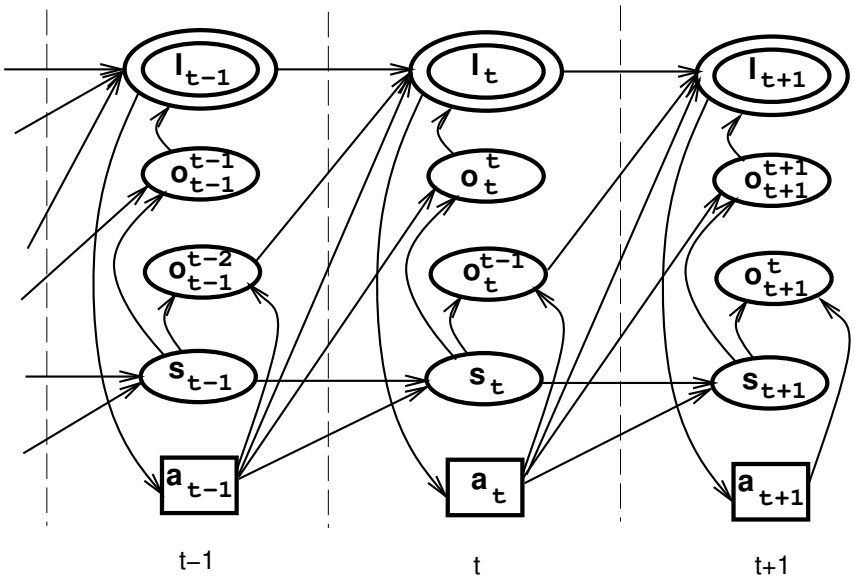


Figure 3-5: POMDP model with the combination of forward and backward observations.

observations made at time t depends either on the state at time $t-1$ or the state at time t, and an action a_{t-1}. Therefore a POMDP model with the combination of forward and backward triggered observations falls also into the category of belief space POMDPs.

3.3.4 POMDP with delayed observations

The class of belief space POMDP models covers only a small part of possible POMDP models used to represent real-world control problems. The important feature of many domains is the need to model time lags in the information (perception) and control (action) channels. In general:

- an action issued by an agent at time t will be performed at time $t+k$;
- an observation made at time t will become available to the agent at time $t+k$.

In this section we will focus on a POMDP model with delayed observations. The model with delayed actions can be treated in a similar way.

The basic motivation for introducing the model with observation lags is that the time at which the observation is made and the time at which it is seen by an agent can differ. If time is discretized the delay may span one or more time steps. The delayed model can be very important, for example, in the medical domain in which some test results are not available immediately, but only after some delay.

In the following we will show how one can go about constructing a suitable information state MDP for a k-step delayed observation model (see figure 3-6). The important features of the model in figure 3-6 are:

- observations are triggered backwards;

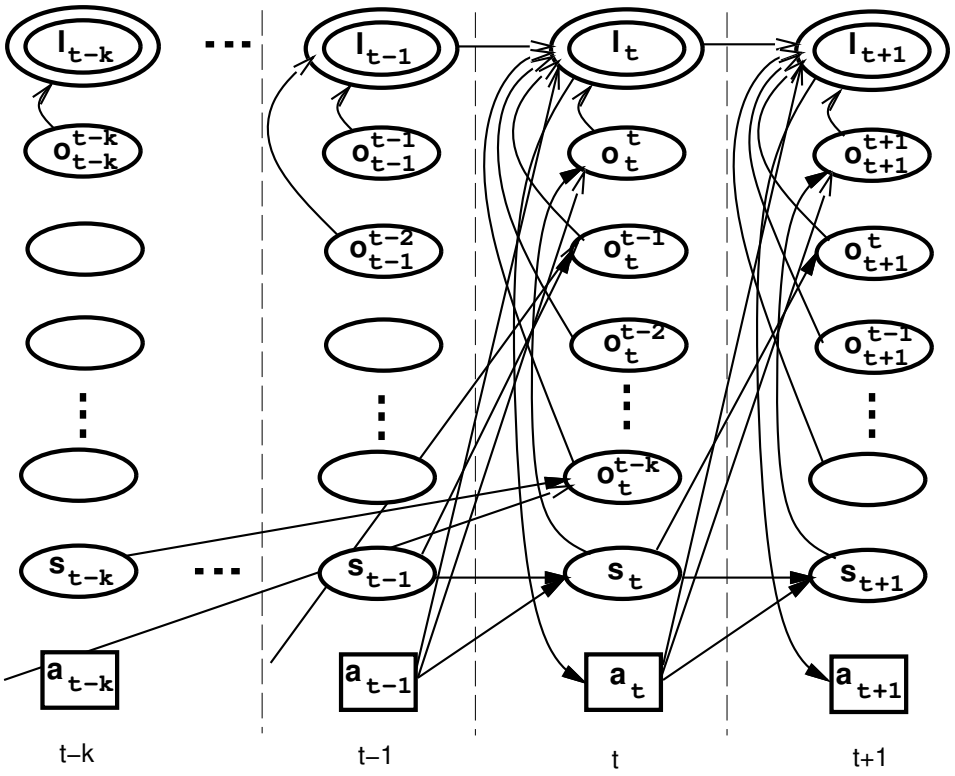


Figure 3-6: POMDP with k-step delayed observation model.

- observations with different time lags are assumed to be independent given the process state;
- at every time t the agent can expect to receive results related to at most k past process states.

An observation model with k step delays can be formalized as:

$$O : S \times A \times \Theta \times D \rightarrow [0, 1]$$

where $D = \{0, 1, 2, \cdots k\}$, denotes the delay with which the observation becomes available to the agent.

Based on different time lags, observations in Θ can be distributed into groups: $\Theta_0, \Theta_1, \cdots, \Theta_k$, where members of every group are observed with the delay corresponding to the index. Then, one can describe the observation model alternatively as:

$$O : S \times A \times \Theta_0 \times \Theta_1 \times \cdots \times \Theta_k \rightarrow [0, 1]$$

or using independence between observations with different lags as:

$$O = \{O_0, O_1, \cdots O_k\}$$

where

$$O_i : S \times A \times \Theta_i \rightarrow [0,1]$$

for all $0 \leq i \leq k$.

Contrary to other models, the computation of a new belief state for the k-step delayed model cannot be done solely from the previous belief state, previous action and new observations. This is because delayed observations influence the belief about the past state, that in turn affects the current belief. This violates the third prerequisite of the sufficient information state process and one cannot use a belief state as a sufficient replacement of the complete information vector.

A suitable sufficient information state process can be built using basic principles of probabilistic inference in graphical models (see [Pearl 89] [Jensen 96] [Castillo et al. 97]): Let λ^t_{t-i} be a contribution to the belief state at time $t-i$ that comes from observations related to that state and that were made up to time t:

$$\lambda^t_{t-i}(s) = \prod_{j=t-i}^{t} P(o^j_{t-i}|s, a_{t-i}).$$

Let us call λ an *observation vector*.

Let ω^t_{t-i} be a contribution to the belief state at time $t-i$ from all actions made prior to that time, related observations made up to time t, and prior belief at time $t=0$:

$$\omega^t_{t-i}(s) = P(s|o^t_{t-i-1}, \cdots, o^{t-i-1}_{t-i-1}, \cdots, o^t_0, \cdots, o^0_0, a_{t-i-1}, a_{t-i-2}, \cdots, a_0, \omega^0_0)$$

where ω^0_0 stands for the prior belief at time $t=0$. As ω^t_{t-i} captures the contribution to the belief state from previous observations, we will call it the *prior belief state (or vector)*. Note, that ω in fact corresponds to π messages in Markov trees in Pearl's notation [Pearl 89]. However, we already use the π symbol to denote a control policy and thus in order to avoid the confusion we have chosen the new symbol ω.

The belief in state s at time t can be expressed Using λ and ω vectors as:

$$b^t_t(s) = \beta \omega^t_t(s) \lambda^t_t(s)$$

where β is a normalizing constant equal to:

$$\beta = 1/\sum_{s \in S} \omega^t_t(s) \lambda^t_t(s).$$

The value of a prior belief vector ω^t_t is computed recursively from the past state contributions:

$$\omega^t_{t-i}(s) = \beta \sum_{s' \in S} P(s|s', a_{t-i}) \lambda^t_{t-i-1}(s) \omega^t_{t-i-1}(s)$$

for $0 \leq i \leq k-1$, and

$$\omega^t_{t-i}(s) = \omega^{t-1}_{t-i}(s)$$

for $k \leq i$.

This means that in order to compute the new belief state properly one needs to know not only new observations, but also observations related to the past k steps, past k actions and prior belief for process state k-steps in the past. Therefore, one can construct an information

state MDP using information states I_t:

$$I_t = \{a_{t-1}, \cdots, a_{t-k}, O_t^t, O_{t-1}^{t-1}, \cdots, O_{t-k}^{t-k}, \omega_{t-k}^t\}$$

where O_i^j stands for all observations related to time i and observed up to time j. It is easy to show that I_t is sufficient to compute $P(s_t|I_t^C)$. Similarly we can show that an information state is Markov updateable and that it allows one to correctly compute the probability of observations seen in the next step. This can be seen since we can always:

- compute a prior belief ω_{t-k}^t at time $t-k$ from observations related to the state at that time and previous state prior belief vector;
- update observation sets, by excluding observations related to a state at time $t-k-1$ and including all new observations;
- compute the probability $P(o^t|I_t^C, a_{t-1})$ using I_t as the maximum observation delay is limited to k steps.

Therefore the original POMDP model with k-step delays can be converted to the information state MDP with process states corresponding to I_t.

3.4 Computing optimal control policies for POMDPS

The policy problem computes optimal control for all information states. This problem was shown to be of polynomial complexity for the MDP framework and for both finite and infinite horizon problems (see Chapter 2). Unfortunately the computation of optimal control decisions in the partially observable case turns out to be far more complex and computationally demanding. This is illustrated by the fact that a POMDP decision problem with a single initial state, finite horizon and no observation delays was shown to be PSPACE-hard [Papadimitriou, Tsitsiklis 87], thus making the planning problem intractable and algorithms providing exact solutions inefficient.

3.4.1 Computing optimal control policy

Finite horizon problem

The finite horizon problem could be solved theoretically using the dynamic programming paradigm. That is, assuming we know the optimal value function for $i-1$ steps-to-go we can compute the optimal value function for any information state with i steps-to-go as:

$$V_i^*(I_i) = \max_{a \in A} \sum_{s \in S} \rho(s,a) P(s|I_i) + \gamma \sum_{o \in \Theta_{next}} P(o|I_i, a) V_{i-1}^*(\tau(I_i, o, a)),$$

described also as $V_i^* = HV_{i-1}^*$. Then the optimal control action is:

$$\mu_i^*(I_i) = \text{argmax}_{a \in A} \sum_{s \in S} \rho(s,a) P(s|I_i) + \gamma \sum_{o \in \Theta_{next}} P(o|I_i, a) V_{i-1}^*(\tau(I_i, o, a)).$$

Then starting from the 0 steps-to-go value function (expressing the expected cost associated with information states at the end) one could theoretically compute optimal value and control functions for all possible information states for 1 step to go, then use the 1 step-to-go optimal

value function to compute optimal actions and value functions for all information states at 2 steps-to-go and so on up to n steps-to-go.

Computing ϵ optimal control for the infinite discounted horizon

Finding an ϵ-optimal value function for the infinite discounted horizon could be approached similarly using the value iteration strategy. Knowing that the value function mapping H is an isotone contraction, we could construct a simple value iteration method with step:

$$V_{i+1} = HV_i$$

that converges to the unique fixed point solution V^* (using the result of the Banach theorem). Therefore, after a sufficient number of iterations we could obtain any ϵ-optimal solution. Using the optimal or ϵ optimal substitute, the optimal control is:

$$\mu^*(I) = \text{argmax}_{a \in A} \sum_{s \in S} \rho(s,a) P(s|I) + \gamma \sum_{o \in \Theta_{next}} P(o|I,a) V^*(\tau(I,o,a)).$$

Note that the value iteration update step is equal to the dynamic programming update step.

3.4.2 Computability of the optimal or ϵ optimal solutions

There is a serious problem in applying both of the above computational schemes in practice. The problem stems from the fact that in the POMDP the information state space is infinite (for example, there is an infinite number of belief states in the sufficient belief state space). Then having a continuous component in the state description poses the following threats:

- a value function for the complete information state space may not be representable by finite means and/or computable in a finite number of steps;
- a control function that maps the information state space may not be computable in a finite number of steps.

Luckily the above threats do not always materialize and one can guarantee in some cases the computability of value and control functions using a finite number of dynamic programming or value iteration updates as well as the their finite description. In the following we will narrow our attention to the problem of finding optimal value functions for a class of belief space POMDPs. This class, as discussed above, covers POMDPs with standard (forward triggered), backward triggered observations models, as well as their combinations.

Computing optimal value functions for belief space POMDPs

A nice and important feature of POMDP models with sufficient belief states is that their optimal or ϵ-optimal value functions are piecewise linear and convex. That is, V_i^* (V_i for the infinite discounted horizon)) can be expressed as:

$$V_i^*(b) = \max_{\alpha_i^k \in \Gamma_i} \sum_{s \in S} b(s) \alpha_i^k(s)$$

where b denotes a belief state and Γ_i is a set of linear vectors α_i^k defining the value function. A piecewise linear and convex value function for a two state POMDP is illustrated in figure 3-7.

Figure 3-7: An example of a piecewise linear and convex value function for a POMDP with two process states $\{s_1, s_2\}$. Note that for the components of the belief state hold: $b(s_1) = 1 - b(s_2)$.

The piecewise linearity and convexity of the value function will be shown in the following theorem. It is based on the theorem proven by Smallwood and Sondik for standard observation models [Smallwood, Sondik 73]. The theorem presented here can be viewed as a generalization of the result that covers a class of belief space POMDPs.

Theorem 8 *(Piecewise linear and convex value functions) Let V_{init} be an initial value function that is piecewise linear and convex. Then a value function obtained after a finite number of update steps for a belief space POMDP is also finite, piecewise linear and convex, that is,*

$$V_i^*(b) = \max_{\alpha_i^k \in \Gamma_i} \sum_{s \in S} b(s)\alpha_i^k(s),$$

where b and α_i^k are vectors of size $|S|$, and Γ_i is a finite set of linear α_i vectors.

Proof. In the proof a notation for the finite horizon case and the dynamic programming update is used. However, the proof holds also for the value iteration update and the infinite discounted horizon criterion. Let us assume that the optimal value function for any b_{i-1} at $i-1$ steps-to-go is expressed using a finite set of vectors $\Gamma_{i-1} = \{\alpha_{i-1}^0, \alpha_{i-1}^1, \cdots, \alpha_{i-1}^l\}$ as:

$$V_{i-1}^*(b_{i-1}) = \max_{\alpha_{i-1}^k \in \Gamma_{i-1}} \sum_{s \in S} b_{i-1}(s)\alpha_{i-1}^k(s).$$

We will show that for i steps the optimal value function is also piecewise linear and convex. Knowing that a belief state is a sufficient information vector, we can write the belief of being in state s at $i-1$ steps-to-go after performing action a in the belief state b_i and subsequently observing o as:

$$b_{i-1}(s) = P(s|b_i, a, o).$$

Using this in the value function we get:

$$V_{i-1}^*(b_{i-1}) = \max_{\alpha_{i-1}^k \in \Gamma_{i-1}} \sum_{s \in S} P(s|b_i, a, o)\alpha_{i-1}^k(s).$$

Substituting the value function in to the equation 3.3 we get:

$$V_i^*(b_i) = \max_{a \in A} \sum_{s' \in S} \rho(s', a) b_i(s') + \gamma \sum_{o \in \Theta_{next}} P(o|b_i, a) \max_{\alpha_{i-1}^k \in \Gamma_{i-1}} \sum_{s \in S} P(s|b_i, a, o) \alpha_{i-1}^k(s)$$

This can be further rewritten as:

$$\begin{aligned}
V_i^*(b_i) = & \\
&= \max_{a \in A} \sum_{s' \in S} \rho(s', a) b_i(s') + \gamma \sum_{o \in \Theta_{next}} P(o|b_i, a) \max_{\alpha_{i-1}^k \in \Gamma_{i-1}} \sum_{s \in S} P(s|b_i, a, o) \alpha_{i-1}^k(s) \\
&= \max_{a \in A} \sum_{s' \in S} \rho(s', a) b_i(s') + \gamma \sum_{o \in \Theta_{next}} \max_{\alpha_{i-1}^k \in \Gamma_{i-1}} \sum_{s \in S} P(o|b_i, a) P(s|b_i, a, o) \alpha_{i-1}^k(s) \\
&= \max_{a \in A} \sum_{s' \in S} \rho(s', a) b_i(s') + \gamma \sum_{o \in \Theta_{next}} \max_{\alpha_{i-1}^k \in \Gamma_{i-1}} \sum_{s \in S} P(s, o|b_i, a) \alpha_{i-1}^k(s) \\
&= \max_{a \in A} \sum_{s' \in S} \rho(s', a) b_i(s') + \gamma \sum_{o \in \Theta_{next}} \max_{\alpha_{i-1}^k \in \Gamma_{i-1}} \sum_{s \in S} \left[\sum_{s' \in S} P(s, o|s', a) b_i(s') \right] \alpha_{i-1}^k(s).
\end{aligned}$$

Let $\alpha_{i-1}^{b,a,o} \in \Gamma_{i-1}$ denotes the optimal selection of α (the one that maximizes the value function) for fixed b, a, o. Then we can write:

$$\begin{aligned}
V_i^*(b_i) = & \\
&= \max_{a \in A} \sum_{s' \in S} \rho(s', a) b_i(s') + \gamma \sum_{o \in \Theta_{next}} \sum_{s \in S} \left[\sum_{s' \in S} P(s, o|s', a) b_i(s') \right] \alpha_{i-1}^{b_i,a,o}(s) \\
&= \max_{a \in A} \sum_{s' \in S} b_i(s') \left[\rho(s', a) + \gamma \sum_{o \in \Theta_{next}} \sum_{s \in S} P(s, o|s', a) \alpha_{i-1}^{b_i,a,o}(s) \right].
\end{aligned}$$

Assuming the complete belief space, the expression in brackets can evaluate to $|A||\Gamma_{i-1}|^{|\Theta_{next}|}$ different vectors: one for every combination of actions and permutations of α_{i-1} vectors of size $|\Theta_{next}|$. Assuming that each vector equals some $\alpha_i^k \in \Gamma_i$, we can rewrite the $V_i^*(b_i)$ as:

$$V_i^*(b_i) = \max_{\alpha_i^{a,j} \in \Gamma_i} \sum_{s \in S} b_i(s) \alpha_i^{a,j}(s)$$

where

$$\alpha_i^{a,j}(s') = \rho(s', a) + \gamma \sum_{o \in \Theta_{next}} \sum_{s \in S} P(s, o|s', a) \alpha_{i-1}^{a,o,j}(s)$$

corresponds to a linear vector for an action a and the j-th permutation of α_{i-1} vectors of size $|\Theta_{next}|$. But that means that $V_i^*(b_i)$ is also piecewise linear, convex and is defined by a finite collection of α vectors Γ_i.
As an initial function V_{init} is piecewise linear and convex, the value function acquired after a finite number of update steps must be also piecewise linear and convex, which concludes the proof. □

The major consequences of the above theorem are that:

- starting from a finite, piecewise linear and convex function one can always compute the value function for a finite number of update steps in finite time;

- the value function acquired after a finite number of update steps can be represented by finite means, using a finite number of linear α vectors;
- the control function is computable.

Useful linear α vectors

A value function V_i consists of a finite number of linear segments (α vectors). This was shown in the theorem proof by constructing a linear vector set Γ_i that consisted of linear vectors corresponding to all possible combinations of actions and pairs of observations and α_{i-1} vectors. The total number of all possible linear vectors is $|A||\Gamma_{i-1}|^{|\Theta_{next}|}$. However, in practice the complete set of linear vectors is rarely used. This is because some of the linear vectors are completely dominated by other vectors and their omission does not influence or change the resulting piecewise linear and convex function. A linear vector that can be eliminated without changing the resulting value function solution is called a *redundant linear vector*. Conversely, a linear vector that singlehandedly achieves optimal value for at least one point of the information vector space is called a *useful linear vector*[2].

For the sake of computational efficiency it is important to keep the size of the linear vector set as small as possible (keep only useful linear vectors) over dynamic programming or value iteration steps. This is because finding the value function V_i^* requires one to check and try all linear vectors in Γ_{i-1} and including redundant ones. The effect of not removing redundant linear vectors after every update would then lead to the growth of the number of redundant vectors and can be a source of major inefficiency.

Unfortunately, it has also turned out that the problem of finding useful linear vector sets cannot be solved efficiently with regard to $|S|, |A|, |\Theta_{next}|, |\Gamma_{i-1}|, |\Gamma_i|$. This was proved in [Littman et al. 95c], who showed that the problem can be solvable efficiently only when $RP = NP$. This means that one does not only face the potential exponential growth of the number of useful linear vectors, but also inefficiencies related to the identification of such vectors. In the following we will explore several methods for computing value function updates that output piecewise linear value functions described only by useful linear vectors. Such updates are then repeatedly used within the main dynamic programming or value iteration procedures.

3.5 Algorithms for updating piecewise linear and convex value functions

In the following we will briefly review some of the existing algorithms for computing piecewise linear and convex value function updates. Unfortunately, as mentioned above, neither these nor other algorithms are guaranteed to run in time polynomial in $|S|, |A|, |\Theta_{next}|, |\Gamma_{i-1}|, |\Gamma_i|$.

The first group of methods fall into the category of generate and test algorithms. We start with a simple generate and test algorithm, called Monahan's algorithm [Monahan 82], and then proceed with its more complex extensions. These algorithms try to construct a useful linear vector set by combining linear vectors in Γ_{i-1} and testing them for redundancy using either intermediate or final redundancy tests.

The alternate methods for computing useful linear vector updates are based on Sondik's idea of computing an optimizing linear vector for a single belief point [Smallwood, Sondik 73].

[2] In defining the redunant and useful linear vectors we assume that there are no linear vector duplicates, i.e. only one copy of the same linear vector is kept in the set Γ_i.

These methods try to locate belief points that can seed new useful vectors. The search for "seed" points must be complete in the sense that belief points examined must guarantee that none of the useful vectors will be missed. We will describe and analyze two algorithms from this group. The first is called the linear support algorithm and is due to Cheng [Cheng 88] (see also [Cassandra 94]). The second is the Wittness algorithm and is due to [Cassandra 94] and [Littman 94]. Other methods that fall into this category are Sondik's method [Cassandra 94] or Cheng's relaxed region algorithm[Cheng 88] [Cassandra 94]. A nice description of several exact algorithms is provided in [Cassandra 94].

Finally, at the end we will propose a new Gauss-Seidel speedup of the value iteration method for infinite discounted horizon problems.

3.5.1 Monahan's algorithm

Monahan's algorithm uses a simple generate and test approach [Monahan 82] [Cassandra 94]. The generation phase of the algorithm corresponds to the enumeration of a complete and possibly redundant set of α_i vectors. Every α_i vector corresponds to one possible combination of an action and a permutation of previous step vectors α_{i-1} of size $|\Theta_{next}|$. A linear vector obtained for an action a and j-th permutation of size $|\Theta_{next}|$ of vectors in Γ_{i-1} is computed as:

$$\alpha_i^{a,j}(s') = \rho(s',a) + \gamma \sum_{o \in \Theta_{next}} \sum_{s \in S} P(s,o|s',a)\alpha_{i-1}^{a,o,j}(s).$$

This gives a total of $|A||\Gamma_{i-1}|^{|\Theta_{next}|}$ vectors α_i^k in Γ_i.

In the testing phase all redundant vectors in Γ_i are tested and removed. A redundant vector is a vector that does not singlehandedly optimize the value function on at least one point of the belief space. Assuming that α_i^k is a vector to be tested for redundancy, the test can be accomplished by setting up the following linear program (see [Monahan 82] or [Cassandra 94]):

$$\text{maximize: } \delta$$
$$\text{using the following constraints:}$$

$$\sum_{s \in S} b(s) \left[\alpha_i^j(s) - \alpha_i^k(s)\right] + \delta \leq 0 \qquad \text{for all} \quad \alpha_i^j(s) \in \Gamma_i \qquad \text{such that} \quad \alpha_i^j(s) \neq \alpha_i^k(s)$$

$$\sum_{s \in S} b(s) = 1$$

$$b(s) \geq 0 \text{ for all } s \in S.$$

The elements of b ($b(s)$) and a parameter δ are treated as linear program variables. If it is found that the maximum possible δ is less than or equal to 0 ($\delta \leq 0$), it must be the case that α_i^k is not singlehandedly best at some point of the belief space. Then, it is either dominated or covered by other α vectors. Therefore, testing the resulting δ makes it possible to exclude a specific redundant vector from Γ_i.

In principle one can test all possible vectors using the above linear program. However, this can be quite expensive, especially when large linear programs need to be solved. The testing process can be sped up to some extent by excluding some of the redundant αs through a cheaper pure dominance test. In the pure dominance test, a vector α_i^k can be excluded (is redundant)

whenever there is a vector α_i^j such that:

$$\text{for all} \quad s = 1 \cdots |S| \quad \alpha_i^j(s) \geq \alpha_i^k(s) \quad \text{holds.}$$

A simple dominance test can cut the size of the linear vector set before more expensive linear programming test is used. This modification was suggested in [Eagle 84].

3.5.2 Extensions of Monahan's algorithm

The main problem with Monahan's algorithm is that it tries to generate blindly all possible vectors first and only then to remove the redundant ones. However, it is also possible to test a partially built solution [Cassandra et al. 97] [Zhang, Liu 96]. This feature makes it possible to interleave the generate and test phases and save some time by recognizing and pruning partial components that are suboptimal earlier. The idea of interleaving the generation and test phases can be used to construct new versions of Monahan's approach.

Interleaving processes of linear vector generation and testing

Let us assume that a set of observations Θ_{next} is partitioned into M disjoint subsets $\{\theta_{next}^1, \cdots \theta_{next}^k \cdots \theta_{next}^M\}$. Then we can rewrite the expression for computing a new linear vector using the partitioning as:

$$\begin{aligned}
\alpha_i^{a,j}(s') &= \rho(s',a) + \gamma \sum_{o \in \Theta_{next}} \sum_{s \in S} P(s,o|s',a)\alpha_{i-1}^{a,o,j}(s) \\
&= \rho(s',a) + \gamma \sum_{o \in \Theta_{next}^1} \sum_{s \in S} P(s,o|s',a)\alpha_{i-1}^{a,o,j}(s) + \cdots + \\
&\quad + \gamma \sum_{o \in \Theta_{next}^k} \sum_{s \in S} P(s,o|s',a)\alpha_{i-1}^{a,o,j}(s) + \cdots + \\
&\quad + \gamma \sum_{o \in \Theta_{next}^M} \sum_{s \in S} P(s,o|s',a)\alpha_{i-1}^{a,o,j}(s).
\end{aligned}$$

Now assume two vectors, $\alpha_i^{a,l}$ and $\alpha_i^{a,m}$, with identical action a and with linear vector choices $\alpha_{i-1}^{a,o,j}$ that differ only in the partition Θ_{next}^k. But then, whenever:

$$\sum_{o \in \Theta_{next}^k} \sum_{s \in S} P(s,o|s',a)\alpha_{i-1}^{a,o,l}(s) \geq \sum_{o \in \Theta_{next}^k} \sum_{s \in S} P(s,o|s',a)\alpha_{i-1}^{a,o,m}(s) \quad \text{for all} \quad s' \in S,$$

the linear vector $\alpha_i^{a,m}$ must be redundant and can be excluded from the useful vector set. This represents a redundancy test for two partially constructed linear vectors and can be applied within any partition. The test can be extended to handle a set of linear vectors by using the same linear program as used for complete linear vector sets. The main advantage of the partial test is that the number of linear vectors to be compared and tested is usually smaller, and therefore cheaper.

One can construct various methods that employ different partitioning schemes and generate linear vectors from components that have passed partial (lower level) redundancy tests. For example one can create a hierarchical scheme that uses a fixed ordering of observations Θ_{next} and that constructs the solution gradually by computing and testing partial linear vectors for

the first two observations, then partial linear vectors for the first three observations, and so on, up to all observations. The advantage of such an approach is that only partial linear vectors found to be nonredundant on the lower level are combined and used on the next level. This leads to the incremental scheme that interleaves generation and test phases. The incremental approach was proposed and its performance tested in [Cassandra et al. 97]. It can result in significant speedups for problems with a large number of redundant linear vectors.

Pruning redundant partial linear vectors across different actions

The idea of partitioning allows one to do early redundancy tests and pruning for linear vectors created for the same action. However, the question is whether we can apply early pruning and use a similar approach also accross different actions. The idea for doing this is proposed and described below and is based on the upper bound linear vector estimates.

Let $\Gamma_i^{A_m}$ be a set of useful linear vectors built for actions $A_m \subset A$. Let $\alpha_i^{a',j}$ be a linear vector obtained for action $a' \notin A_m$ and the j-th permutation of $|\Theta_{next}|$ linear vectors in Γ_{i-1}. Let $\widehat{\alpha}_i^{a',j}$ be an upper bound estimate of $\alpha_i^{a',j}$. Then if $\widehat{\alpha}_i^{a',j}$ is found redundant with regard to $\Gamma_i^{A_m}$ then it must hold that $\alpha_i^{a',j}$ is redundant as well and can be excluded.

The question now is how to compute an upper bound estimate of the complete linear vector for some partially built linear vector. Let $\alpha_i^{a',j,k}$ be a partial linear vector built for the partition Θ_{next}^k:

$$\alpha_i^{a',j,k}(s') = \sum_{o \in \Theta_{next}^k} \sum_{s \in S} P(s,o|s',a') \alpha_{i-1}^{a',o,j}(s).$$

Then we can construct an upper bound estimate $\widehat{\alpha}_i^{a',j,k}$ for it as:

$$\widehat{\alpha}_i^{a',j,k}(s') = \sum_{o \in \Theta_{next}^k} \sum_{s \in S} P(s,o|s',a') \max_{\alpha_{i-1} \in \Gamma_{i-1}} \alpha_{i-1}(s),$$

which can be computed very easily. Then combining together either exact partial vectors or their upper bound estimates for different partitions we can compute an upper bound $\widehat{\alpha}_i^{a',j}$. For example, using the exact partial solution for the first partition and upper bound estimates for all other partitions we get :

$$\widehat{\alpha}_i^{a',j}(s') = \rho(s',a') + \gamma \left[\left[\sum_{o \in \Theta_{next}^1} \sum_{s \in S} P(s,o|s',a') \alpha_{i-1}^{a',o,j}(s) \right] + \cdots + \widehat{\alpha}_i^{a',j,k}(s') + \cdots + \widehat{\alpha}_i^{a',j,M}(s') \right].$$

The fact that one can relatively easily compute the upper bound estimates of partial linear vectors for every partition (one needs to compute $\max_{\alpha_{i-1} \in \Gamma_{i-1}} \alpha_{i-1}(s)$ only once) can be used to do the redundancy check of partially built linear vectors across actions. This test can be combined with the redundancy test for partial linear vector sets and fixed actions, discussed above. In general the early elimination of redundant partial linear vectors can speed up the construction of the useful linear vector set Γ_i and can be very useful in cases in which the number of useful linear vectors is relatively small compared to the size of the maximum linear vector set.

3.5.3 Cheng's linear support algorithm

Cheng's linear support approach [Cheng 88] [Cassandra 94] exploits piecewise linearity and convexity of the value function to construct a set of useful linear vectors from scratch. The algorithm is based on two key features:

- It is possible to find a useful linear vector(s) for any point of the belief space (using Sondik's point update method [Smallwood, Sondik 73]).
- Any subset $\widehat{\Gamma}_i$ of useful vectors Γ_i defines a piecewise linear and convex approximation that is worst at intersections of vectors in $\widehat{\Gamma}_i$, and/or at intersections of such vectors with belief space boundaries [Cheng 88].

The above two features give rise the following idea for finding the useful vector set: starting from the initial incomplete useful set, find all useful vectors gradually by checking points created by intersections of already known α vectors. This idea is embodied in the following algorithm which is the modified version of Cheng's algorithm.

Chengs's algorithm (Γ_{i-1})
select arbitrary point b of the belief space;
initialize $\widehat{\Gamma}_i$ with a useful vectors built for b and mark them;
while there exists a marked vector in $\widehat{\Gamma}_i$
do select marked vector α from $\widehat{\Gamma}_i$;
find all extreme points of the region for which α gives the optimal value
(using other vectors in $\widehat{\Gamma}_i$ and simplex constraints);
for each extreme point b of region α
compute useful vector for b;
if the new useful vector is not in $\widehat{\Gamma}_i$
add it to $\widehat{\Gamma}_i$ and mark it;
otherwise ignore it;
unmark α;
return $\widehat{\Gamma}_i$ as Γ_i;

The algorithm relies on the ability to compute:

- all extreme points of the belief space region defined by some useful linear vector $\alpha_i^k \in \widehat{\Gamma}_i$, i.e. α_i^k is optimal on the region;
- useful vectors for an arbitrary belief state.

Let us look more closely at these tasks.

Computing all extreme points of the belief region

Let α_i^k be a useful vector in $\widehat{\Gamma}_i$. Then a belief space region for which it is optimal satisfies the following constraints:

$$\sum_{s \in S} b(s) \left[\alpha_i^j(s) - \alpha_i^k(s) \right] \leq 0 \qquad \text{for all} \quad \alpha_i^j(s) \in \widehat{\Gamma}_i \qquad \text{such that} \quad \alpha_i^j(s) \neq \alpha_i^k(s)$$

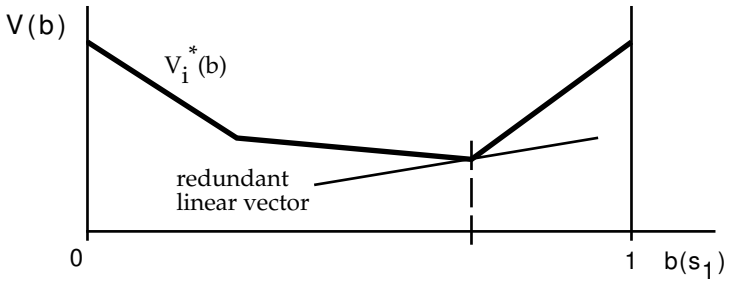


Figure 3-8: A situation in which a linear vector that is optimal for some belief point can become redundant. The vector is dominated and fully covered by other linear vectors.

$$\sum_{s \in S} b(s) = 1$$

$$b(s) \geq 0 \text{ for all } s \in S$$

The problem with computing all extreme points for the α_i^k region is that the number of different extreme points can be exponential in $\widehat{\Gamma_i}$ or $|S|$. But this means that computing and checking all extreme points of the α_i^k region can lead to inefficiency, as in general there is no guarantee that every extreme vertex seeds a new useful α vector. Ideally, what we would like is to compute and check only those vertices of the region that seed new α vectors. This is the crucial point of the approach and any efficient (polynomial time) solution to it will lead to the efficient running time of the overall algorithm.

Computing useful vectors for a single belief point

The other open spot in the algorithm description is related to the task of finding useful vectors for a specific belief point. The main problem here is that there can be an α vector that is optimal at some point but despite that it is redundant. This corresponds to the situation in which the vector is covered and dominated by other linear vectors. The situation is illustrated in figure 3-8. Therefore one cannot simply select a vector that gives the optimal value for the target point without a guarantee that there is some belief point that is singlehandedly optimized by the vector. In the following it will be shown how such a vector can be found efficiently.

Let us first consider the problem of computing the optimal value function for some belief point b. This can be achieved using the value function equation:

$$V_i^*(b) = \max_{a \in A} \left[\sum_{s' \in S} \rho(s', a) b(s') + \gamma \sum_{o \in \Theta_{next}} \left[\max_{\alpha_{i-1}^k \in \Gamma_{i-1}} \sum_{s \in S} \left[\sum_{s' \in S} P(s, o | s', a) b(s') \right] \alpha_{i-1}^k(s) \right] \right].$$

Following the shown parenthesization we can compute the optimal value function from inside out using the following steps:

1. For a fixed a and o try all α_{i-1}^k and select the result that maximizes:

$$\sum_{s \in S} \left[\sum_{s' \in S} P(s, o|s', a) b(s') \right] \alpha_{i-1}^k(s).$$

2. For a fixed a do step 1 for all possible o, and sum the maximal results.

3. Compute $\sum_{s' \in S} \rho(s', a) b(s')$ and add it to the result achieved in step 2.

4. For all possible a select the best overall result.

This task can be accomplished in time $O(|A||\Theta_{next}||\Gamma_{i-1}||S|^2)$.

As seen above, the task of computing the value function at point b is relatively easy. However our task is to find a linear vector that optimizes the value at b and is also useful.

Sondik's linear vector update method

The optimal linear vector for a belief point b and an action a can be computed using Sondik's approach [Smallwood, Sondik 73]:

$$\alpha_i^{b,a}(s) = \rho(s, a) + \gamma \sum_{o \in \Theta_{next}} \sum_{s' \in S} P(s', o|s, a) \alpha_{i-1}^{\iota(b,a,o)}(s') \tag{3.8}$$

where $\iota(b, a, o)$ indexes a linear vector α_{i-1} in a set of linear vectors Γ_{i-1} (defines V_{i-1}) that maximizes:

$$\sum_{s' \in S} \left[\sum_{s \in S} P(s', o|s, a) b(s) \right] \alpha_{i-1}(s')$$

for a fixed combination of b, a, o. The optimizing linear vector for a point b is then obtained by choosing the one giving the best value function result from among candidate vectors computed for all actions. That is, assuming Γ_i^b is a set of all candidate vectors, the resulting vector must satisfy:

$$\alpha_i^b = \text{argmax}_{\alpha_i^{b,a} \in \Gamma_i^b} \sum_{s \in S} \alpha_i^{b,a}(s) b(s).$$

Sondik's method for computing the linear vector that optimizes the value function for point b can be accomplished also in $O(|A||\Theta_{next}||\Gamma_{i-1}||S|^2)$ time. Unfortunately, there can be more than one α_i^b vector that optimizes $V_i(b)$. Then the problem is to select a linear vector that is also guaranteed to be useful.

The existence of more linear vectors that optimize the value function at some point b can be caused by having:

- more than one linear vector $\alpha_{i-1}^{\iota(b,a,o)}(s')$ that optimizes

$$\sum_{s' \in S} \left[\sum_{s \in S} P(s', o|s, a) b(s) \right] \alpha_{i-1}(s');$$

- more than one optimizing $\alpha_i^{b,a}$.

The problem of multiple choices can be resolved by constructing a linear vector that is guaranteed to be useful. This can be achieved in both cases by using a procedure that selects

the optimizing vector from among contenders by comparing linear vector values on a fixed sequence of critical belief points (dimensions). The procedure selects first the vector with the largest component in the first dimension and in the case of ties the vector with the largest component in the second dimension and so on. Such a choice guarantees that the selected vector will lead to the optimal value not only for a belief point b but also for some belief point in b's close neighbourhood (see also [Littman 96]). Moreover the vector is guaranteed to singlehandedly achieve the optimal value for such a point. Therefore the selected vector must be useful. Note that in order to guarantee usefulness, a sequence of fixed belief points (dimensions) needs to be the same for both sets of linear vectors.

3.5.4 Witness algorithm

The Witness algorithm [Cassandra 94] [Littman 94] adopts in principle the same idea as Cheng's linear support algorithm and tries to build the useful vector set gradually by identifying points that seed useful linear vectors. However, the major distinction between the two is that the Witness algorithm applies the idea to find useful α vectors Γ_i^a that describe the action-value function $Q_i(.,a)$. The resulting value function is constructed by combining results for different action-value functions using the redundancy test from Monahan's procedure to enforce usefulness. Contrary to this, Cheng's algorithm builds the value function V_i directly.

The fact that the Witness algorithm identifies action-values first and only then it combines them can be again a source of major inefficiency. This is because the number of useful α vectors generated for some action can be exponential with regard to the useful set of vectors of the resulting value function. However the main advantage and most important feature of the Witness algorithm is that it can construct the action value function efficiently. Then the overhead from finding useful vectors that define all action-value functions and their subsequent combinations is outweighed by the efficiency of the procedure.

The efficiency of the action-value computation stems from the fact that for every useful α_i^k vector in $\widehat{\Gamma_i^a}$ (partially built linear vector set) it is always possible to find a belief point (if it exists) for which there is a different optimal α_i^j not included in $\widehat{\Gamma_i^a}$. Such a point is called a witness point (hence Witness algorithm). This feature makes it different form Cheng's algorithm in which for every new useful vector found, the vertices of the region associated with it need to be enumerated first and then checked, with no guarantee that they will seed new useful vectors. The blind enumeration of all possible vertices is thus the major source of inefficiency, as the number of vertices to be checked can be exponential in $|\Gamma_{i-1}|$ or $|S|$. The following is a basic description of the Witness algorithm.

Witness (Γ_{i-1}, a)
  select arbitrary point b of the belief space;
  initialize $\widehat{\Gamma_i^a}$ using a useful vector defining action-value function for b and a, and mark it;
  while there exists a marked vector in $\widehat{\Gamma_i^a}$
    do select marked vector α from $\widehat{\Gamma_i^a}$;
      while there is a witness point b^* for α and $\widehat{\Gamma_i^a}$
        do
          compute a useful vector for b^*, mark it and add it to $\widehat{\Gamma_i^a}$;
      unmark α;
  return $\widehat{\Gamma_i^a}$ as Γ_i^a;

The key part of the algorithm is the problem of finding the witness point, that is a point of

the belief space that is optimized by α_i^k with regard to $\widehat{\Gamma_i^a}$, but not with regard to the complete set Γ_i^a. This problem can be solved by constructing a special linear program for every possible observation $o \in \Theta_{next}$ and every vector $\alpha_{i-1}^l \in \Gamma_{i-1}$ such that:

$$\alpha_{i-1}^l \neq \alpha_{i-1}^{k,a,o}$$

where $\alpha_{i-1}^{k,a,o}$ is an α_{i-1} vector from Γ_{i-1} used to construct α_i^k. The linear program then corresponds to [Cassandra 94]):

maximize: δ
using the following constraints:

$$\sum_{s_inS} b(s) \left[\alpha_i^j(s) - \alpha_i^k(s)\right] \leq 0 \qquad \text{for all} \quad \alpha_i^j(s) \in \widehat{\Gamma_i^a} \qquad \text{such that} \quad \alpha_i^j(s) \neq \alpha_i^k(s)$$

$$\sum_{s' \in S} \sum_{s \in S} P(s, o|s', a) b(s') \left[\alpha_{i-1}^{k,a,o}(s) - \alpha_{i-1}^l(s)\right] + \delta \leq 0$$

$$\sum_{s \in S} b(s) = 1$$

$$b(s) \geq 0 \text{ for all } s \in S.$$

Components of b as well as δ represent linear program variables. Assume that α_{i-1}^l is a better choice than $\alpha_{i-1}^{k,a,o}$ for at least one point within the region that is currently optimized by $\alpha_i^{k,a}$. Then a variable δ is larger than 0, $\delta > 0$, and components of b represent the point for which there is an α vector with better value. Thus solving the linear program for every possible observation $o \in \Theta_{next}$ and vector $\alpha_{i-1}^k \in \Gamma_{i-1}$, and checking the resulting δ, allows one to identify a witness point associated with a useful vector α_i.

3.5.5 Value iteration updates

All of the discussed methods can be without change applied to compute updates in the value iteration method and infinite discounted horizon problem. The value iteration method runs until some required precision of the solution value function is reached. The precision can be guaranteed using one of the two stopping criteria: absolute or relative (see section 2.2.2). The absolute criterion uses a minimum number of value iterations one has to perform (they are derived directly from the Banach theorem), while the relative stopping criterion is based on Bellman's residuals.

A slight problem with the relative stopping approach is that the value function is defined over an infinite belief space, compared to the MDP case that works with a finite state space. However, value functions for belief state POMDP are piecewise linear and convex, thus one is always able to compute the maximum difference between two such functions in a finite number of steps. Methods for doing this are discussed for example in [Littman 94].

Incremental (Gauss-Seidel) method

A simple value iteration method is rarely used in the Markov decision framework. Instead, a Gauss-Seidel modification that incorporates immediately any change in value function values

is commonly used. This speeds up the convergence rate and effectively replaces parallel value updates with a continuous update scheme.

The question is if it would be possible to construct a Gauss-Seidel version of the value iteration method also for POMDPs. The prerequisite to this is to find a method that allows one to gradually change the value function so that in the limit the optimal value function is reached. The main idea that makes the construction of such a method possible is a new one and is based on piecewise linear lower bounds [Hauskrecht 96b] [Hauskrecht 97b].

Gauss-Seidel updates

Let V_{i-1} be a piecewise linear lower bound on the optimal value function $V^* = HV^*$ and let Γ_{i-1} be a set of linear vectors describing it. Then a new linear vector α_i computed for an arbitrary belief point b using Sondik's update formula satisfies:

$$\sum_{s \in S} \alpha_i(s) b(s) \geq \max_{\alpha_{i-1} \in \Gamma_{i-1}} \sum_{s \in S} \alpha_{i-1}(s) b(s).$$

This inequality holds because the update formula implements a value function mapping H and H is an isotone contraction. But then we can construct a new piecewise linear convex function V_i such that $V_{i-1} \leq V_i \leq V^*$, simply by updating a linear vector set:

$$\Gamma_i = \Gamma_{i-1} \cup \alpha_i.$$

Note that by computing new Γ_i, some of the previously useful linear vectors can become redundant. One can apply redundancy tests discussed above to elimate such vectors.

The new update method updates and improves the lower bound value function gradually, point by point, and makes results of previous linear vector updates immediately available. In general the update rule can be combined with any point selection strategy, that guarantees the convergence to the optimal solution. That is the strategy is able to eventually locate all necessary points. Systematic and complete point selection strategies can be built by modifying exact methods discussed above, or using simple random strategy that converges to the optimal solution in the limit.

Problem of precision

The major problem with the incremental update rule is that it makes impossible the determination of the boundary of a value iteration step. That is, starting from an arbitrary piecewise linear lower bound value function, one cannot say or detect when the improvement worthy of at least one parallel value iteration step has been made. Thus one can implement neither fixed step nor Bellman residual stopping criteria to guarantee the required precision of the actual solution. Contrary to the incremental Gauss-Seidel method, it is easy to detect the precision of the obtained solution when parallel value function updates are used. Thus the difference between parallel and incremental Gauss-Seidel methods boils down to the ability to check ϵ-optimality of the current solution versus speed and better convergence. One promising avenue of research would be to explore the combination of the two methods that exploits positives of each one and that interleaves exact value iteration steps with incremental Gauss-Seidel updates. We believe that this will allow us to acquire solutions with guaranteed precision for more complex problems than are solvable with currently available exact methods.

3.6 Forward decision methods for finding optimal or near-optimal POMDP control

A policy task that produces complete optimal or ϵ-optimal control functions is computationally very expensive and very hard to accomplish in practice for POMDPs with larger state, action and observation spaces. However, when one expects to find the optimal control or value function only for a single information state, forward decision methods often represent the best choice.

The most appealing property of forward methods is that after a finite number of steps they can reach only a finite number of information states. Information states that can be reached correspond to different action-observation sequences one can generate from the initial information state. Forward methods and strategies used for POMDPs are similar to those for MDPs and are based on the forward decision tree expansion. However partial observability introduces a new dimension of complexity that makes the optimization task harder.

The basic computational structure used for finding the best decision is a decision tree. The main difference between MDPs and POMDPs is that in the POMDP framework the decision nodes are associated with information states while in the MDP framework they are associated with true process states. The fact that sufficient information space can be of infinite size causes an infinite number of different decision subtrees to be present for the infinite horizon problem. This makes it impossible to:

- bound the size of the tree needed for the exact computation;
- cut the computational time through result sharing;

as used in the MDP framework. However, we can still use pruning strategies and eliminate those branches of the tree that are provably suboptimal. Assuming we can show that action-value functions for two actions a and a' satisfy:

$$lbound(Q(a, I)) \geq ubound(Q(a', I))$$

we can eliminate action a'. The effectiveness of pruning then depends strongly on the quality of value function bounds provided.

3.6.1 Computing value function bounds

The bounds can be computed by using the minimum and maximum expected one step rewards. Bounds for the n-step finite horizon problem and information state I_n are:

$$lbound(I_n) = \left[(\gamma^{n+1} - 1)/(\gamma - 1)\right] M_l + \gamma^n M_l^0$$

$$ubound(I_n) = \left[(\gamma^{n+1} - 1)/(\gamma - 1)\right] M_u + \gamma^n M_u^0$$

where M_l, M_u are the minimal and maximal expected one step rewards and M_l^0, M_u^0 are minimal and maximal zero steps-to-go rewards. Bounds for the infinite discounted horizon problem are computed similarly:

$$lbound(I) = \frac{M_l}{1 - \gamma}$$

$$ubound(I) = \frac{M_u}{1 - \gamma}.$$

The above bounds are not very tight. In general far better bounds can be found using other more complex bound strategies. These will be proposed, described and analyzed in the Chapter 4.

3.6.2 Incremental forward methods

Forward decision methods compute the optimal control by forward unfolding of the value function equation. The unfolding steps correspond to dynamic programming updates (steps) for the finite horizon problem and to value iteration updates for the infinite discounted horizon problem. The simplest decision methods can be based on the blind expansion of update formulas. This causes the decision tree to grow exponentially with the number of steps and it can become infinite for the infinite horizon problem. The basic idea of more intelligent methods is to eliminate the full expansion of the decision tree and still compute the same control response. This can be done by devising methods that interleave bound improvement and pruning stages.

Improving internal node bounds

Bounds associated with an internal node of the decision tree can be computed from bounds provided at leaves of the partially expanded tree by performing update backups. This means that the quality of bound values at internal nodes depends both on the bound values supplied to leaves of the partially expanded tree, as well as on the number of updates (backups) one must perform to propagate the bound effect from leaves to the internal node. In other words there are two possible strategies that can lead to the improvement of the bound at any internal decision tree node: either improve the leaf bound function or further expand the partially built tree.

Any improvement in the bound used at leaves translates directly to an improvement of bounds at internal nodes. The reason for this is that H mapping is isotone and thus any change in leaf bounds propagates also to internal node bounds.

The effect of the number of update steps (backups) on the quality of internal node bounds can be direct or indirect, depending on the reward model used. The effect is direct for an infinite discounted horizon model, indirect for the finite horizon case.

Assume we have fixed an initial value function bound for a partial tree built for an infinite discounted horizon problem. By increasing the size of the tree the number of backups increases as well. Then using the same initial bound at new leaves translates to an improvement of the bound. This is because H is an isotone contraction and bounds are guaranteed to improve with more backup updates (correspond to iteration steps).

A finite horizon problem must use different leaf bounds (value functions) for decision trees of different depths. This is because nodes at different levels are associated with different steps-to-go value functions. Expanding the tree by one more level requires that a different leaf value function is used. However, it is often reasonable to assume that both the previous and the new leaf value function bounds are produced by the same procedure that monotonically degrades the bound precision for more steps to go, i.e. bounds for two consecutive steps satisfy:

$$|V_i^* - \widehat{V}_i| \leq |H(V_{i-1}^* - \widehat{V}_{i-1})|,$$

where V^* stands for the optimal value function and $\widehat{V}$ stands for an upper or lower bound. The condition guarantees that the expansion of the decision tree by one level always leads to the improvement of the internal node bounds.

3.6.3 Incremental breadth-first expansion strategy

The simplest incremental decision tree method uses breadth-first expansion strategy. The idea of the method is the following: If the decision about the optimal or ϵ-optimal action cannot be made based on the current tree and bounds, then the decision tree is expanded in a breadth-first manner, that is, all leaf nodes are expanded one level and bounds for all nodes are updated using bounds at new leaves. The algorithm implementing breadth-first strategy is shown below and it is a POMDP modification of the breadth-first algorithm we constructed for the fully observable case (see Chapter 2). The algorithm uses leaf value function bounds V_L and V_U and for the initial information state I_0 computes the action that is guaranteed to be ϵ-optimal.

Incremental expansion - breadth-first($POMDP$, γ, I_0, ϵ, V_U, V_L)
  initialize tree T with I_0 and $ubound(I_0)$, $lbound(I_0)$ using V_U, V_L;
  repeat until (single action remains for I_0 or $ubound(I_0) - lbound(I_0) \leq \epsilon$)
    call Improve-tree(T, $POMDP$, γ, V_U, V_L);
  return an action with the largest lower bound as a result;

Improve-tree(T, $POMDP$, γ, V_U, V_L)
  if $root(T)$ is a leaf
    then expand $root(T)$
      and set bounds $lbound, ubound$ of new leaves using V_L, V_U;
    else for all decision subtrees T' of T
      do call Improve-tree(T', $POMDP$, γ, V_U, V_L);
  update bounds $lbound(root(T)), ubound(root(T))$ for $root(T)$;
  when $root(T)$ is a decision node
    prune suboptimal action branches from T;
  return;

The major problem with the breadth-first approach is that it expands all leaf nodes at once. However, in practice not all subtrees help to discriminate between actions evenly, thus the refinement of bounds is usually influenced more by some subtrees and less by the others. Breadth-first expansion strategy expands leaf nodes blindly and it results in expansions that are unneccessary or not very helpful for the correct decision.

3.6.4 Using heuristics to guide the decision tree expansion

The problem with the breadth-first expansion can be partially remedied by using various heuristics that try to locate branches with larger bound refinement potential and to expand them first. A simple heuristic that seems to work quite well is to promote the expansion of the decision tree based on bound differences. The heuristic is based on the assumption that a larger bound span has a larger potential to be improved (shrunk) and thus has a large chance to result in pruning. The incremental expansion algorithm shown below expands and subsequently recomputes (improves) the branch of the decision tree with the largest bound span. The branch to be expanded (improved) is found in the top-down fashion using the following rules:

- at the decision node corresponding to I_t, choose a succesor chance node with the largest bound difference: $ubound([I_t, a]) - lbound([I_t, a])$;
- at the chance node corresponding to $[I_t, a]$ choose a successor decision node I_{t+1} with the largest weighted bound difference: $P(o|I_t, a)[ubound(I_{t+1}) - lbound(I_{t+1})]$.

Incremental expansion - heuristic($POMDP$, γ, I_0, ϵ, V_U, V_L)
    initialize tree T with I_0 and $ubound(I_0)$, $lbound(I_0)$ using V_U, V_L;
    repeat until (single action remains for I_0 or $ubound(I_0) - lbound(I_0) \leq \epsilon$)
        call Improve-tree(T, $POMDP$, γ, V_U, V_L);
    return action with the largest lower bound as a result;

Improve-tree(T, $POMDP$, γ, V_U, V_L)
    case
        $root$ (T) is a leaf:
            expand root(T) and set bounds $lbound, ubound$ of new leaves using V_L, V_U;
        $root(T)$ is a decision node:
            select subtree T' corresponding to the chance
                node with the largest bound span;
            call Improve-tree(T', $POMDP$, γ, V_U, V_L);
        $root(T)$ is a chance node:
            select subtree T' corresponding to the decision
                node with the largest weighted bound span;
            call Improve-tree(T', $POMDP$, γ, V_U, V_L);
    update bounds $lbound(root(T)), ubound(root(T))$ for $root(T)$;
    when root(T) is a decision node
        prune suboptimal action branches from T;
    return;

The main problem with the above algorithm is that it starts to perform backups (updates) after a single leaf node is expanded. As one node expansion can often lead to a bound improvement that is small, frequent backups with small changes can cause a significant slowdown of the algorithm. This deficiency may be remedied by expanding more then one successor node in one bound improvement cycle. In order to find a good compromise between the slow one-node heuristic expansion and the large scale all node breadth-first expansion we propose a simple randomized strategy that selects branches to be expanded in proportion to their bound difference. The strategy can be implemented by modifying the breadth-first algorithm, such that nodes corresponding to possible branches are expanded with probability:

$$\exp^{[M_{diff} - (ubound(x) - lbound(x))]/T}$$

where M_{diff} is the largest bound span from among the candidates and T is a temperature constant. The randomized strategy usually leads to the expansion of the decision tree at more leaf nodes in one improvement cycle. Note that at least one branch of the tree is always expanded.

3.6.5 Computing the decision in linear space

Though good heuristics can speed up the computation, the optimal decision method still needs to explore trees of extreme sizes. Although time is almost always the issue in evaluating the decision procedures, the computational process can be affected also by another limited resource: memory needed to store the decision tree. In the following we will focus on the memory issue and propose the algorithm that computes the required decision in a linear space. The method does not have any immediate benefit with regard to runtime efficiency and in general makes

the running time worse. Its only benefit is in saving the memory needed to store the tree.

The basic idea of the linear space algorithm is to exploit the heuristic expansion strategy with the capability to cut off and recover branches not currently targeted by an expansion process. The method works in space linear in $|A|, |\Theta|$ and d where d is the maximum depth of the decision tree that needs to be constructed. The selection of the node to be expanded is governed by the following rules:

- at the decision node corresponding to I_t, choose a successor chance node with maximum $ubound([I_t, a])$;
- at the chance node corresponding to $[I_t, a]$, select a decision node I_{t+1} with the largest bound difference: $P(o|I_t, a)[ubound(I_{t+1}) - lbound(I_{t+1})]$.

The linear space algorithm is shown below. It dynamically cuts and recovers previously cut decision tree branches by repeated computation, similar to the iterative deepening procedure (see [Korf 85]). A branch expansion is done in two steps: recovering of the best result first and improving it afterwards. Only after this happens is control returned to the predecessor node.

Incremental expansion - linear space($POMDP$, γ, I_0, ϵ, V_U, V_L)
  initialize tree T with I_0 and $ubound(I_0)$, $lbound(I_0)$ using V_U, V_L;
  set $ubound'(I_0) = ubound(I_0)$ and $lbound'(I_0) = lbound(I_0)$;
  repeat until (single action remains for I_0 or $ubound'(I_0) - lbound'(I_0) \leq \epsilon$)
    call Improve-tree(T, $POMDP$, γ, V_U, V_L);
  return action with the largest lower bound as a result;

Improve-tree(T, $POMDP$, γ, V_U, V_L)
  set $b \leftarrow root(T)$;
  when b has no successors (either leaf node or successors were cut):
    expand b;
    compute bounds $ubound', lbound', ubound, lbound$ of new leaves from V_U, V_L;
    recompute $ubound(b), lbound(b)$ using $ubound', lbound'$ bounds of its successors;
  while $ubound(b) - lbound(b) > ubound'(b) - lbound'(b)$
    case
      b is a decision node:
        select successor c of b corresponding to chance node with largest $ubound'(c)$;
        prune subtrees of all other successors of b;
        call Improve-tree($tree(c)$, $POMDP$, γ, V_U, V_L);
      b is a chance node:
        select successor d of b corresponding to a decision node
          with largest $P(o|I_t, a)[ubound'(c) - lbound'(c)]$;
        prune subtrees of all other successors of b;
        call Improve-tree($tree(d)$, $POMDP$, γ, V_U, V_L);
    recompute $ubound(b), lbound(b)$ using $ubound', lbound'$ bounds of its successors;
    when b is a decision node
      prune suboptimal actions branches from T;
  set $ubound'(b) = ubound(b), lbound'(b) = lbound(b)$
  return;

The algorithm works with two sets of bounds:

- $ubound, lbound$ that denote bounds computed in the current improvement cycle;

- $ubound', lbound'$ that refer to bounds computed in the previous improvement cycles.

$ubound', lbound'$ thus refer to the bounds computed before the current improvement cycle was initiated, and allow us to test if improvement in the computed bounds was achieved. The improvement is guaranteed when:

$$ubound(b) - lbound(b) < ubound'(b) - lbound'(b)$$

holds, as $ubound, lbound$ are bound values computed in the current cycle.

Note also that the active decision tree at any node allows only one successor node to be expanded to the depth of more than 1. All other branches are temporarily pruned and are rebuilt whenever needed. Every temporarily pruned branch has the next node that stores the $ubound', lbound'$ values it can achieve. This means that for any decision node there are at most $|A|$ successor chance nodes and only one of them can be expanded to the greater depth. Similarly every chance node has at most $|\Theta|$ successors with at most one successor expanded to a depth of more than 1. As the maximum depth of the tree explored is d, the number of nodes one needs to keep is linear in $d, |A|, |\Theta|$.

3.6.6 Combining bound improvement strategies

Forward methods, as discussed so far, assume that the value function bounds used to prune the decision tree are improved only through the decision tree expansion. That is, initial bounds used at leaves are given a priori and are fixed during the problem solving. However, we have already pointed out that a change in the leaf bounds can improve internal node bounds as well. Therefore we may also construct methods that improve incrementally the value function bound at leaves and keep the size of the decision tree fixed. Incremental methods capable to improve bound value functions are discussed in more detail in the next chapter.

The two improvement strategies can also be combined. The basic problem we face is the following: There are two methods to improve bounds. These have different time complexity and improve different things. The point is to find an appropriate cost-benefit tradeoff between the two and answer the question of when one method is better then the other. Costs are associated with the computation time and benefits are associated with bound improvements.

The decision tree method is usually better for smaller size decision trees. The reason for this is that it does not require too much effort to expand the tree and backup the solution. On the other hand, when a decision tree becomes very large, the computation of improved bounds using the decision tree can become very expensive. Also, for the infinite discounted problem, the potential of a large step improvement in bounds diminishes with the depth of the decision tree (due to discounting), thus lowering the chance of reaching the required bound precision.

When one faces a large decision tree the improvement of the leaf bound value functions often becomes more appropriate. This is because an improvement of the complete bound can become computationally cheaper than any improvement accomplished through the expansion and backups. Methods capable of improving the value function bounds incrementally for the complete information vector space will be described in the next chapter.

A strategy that combines advantages of both methods can be constructed using a metalevel decision procedure that selects the most promising improvement procedure to be tried next based on available cost-benefit profiles. The profiles can be either static and provided at the beginning or can change (adapt) with regard to the actual cost-benefit results acquired for the problem. An adaptive procedure then monitors costs and benefits of a decision tree expansion and leaf bound improvement, and adjusts the profile accordingly.

3.7 Summary

The framework of partially observable Markov decision processes (POMDPs) models two sources of uncertainty: action outcome uncertainty and partial observability. To find the optimal control the POMDP is converted to an information-state MDP, that uses information states corresponding to complete histories of actions and observations or appropriate sufficient statistics that preserve the Markov property of the information process. The common problem with information state MDPs is that they use states that are continuous or states that expand the dimension with elapsed time. This feature makes the computation of optimal value functions and optimal control policies very hard. In fact, optimal or ϵ-optimal solutions are possible only for a class of POMDPs that can be converted into belief-state MDPs. These are solved using dynamic programming or value iteration methods and rely on piecewise-linearity and convexity of value functions [Smallwood, Sondik 73]. Alternatively, when the optimal decision for a single initial state is sought, forward decision tree methods based on bounds can be applied.

Contributions

The chapter describes the POMDP framework, and summarizes exact methods for solving control problems within the framework. New contributions presented in this chapter are related to various improvements and speed-ups of exact methods. These include:

- A speed-up of the incremental version of the Monahan's algorithm. The incremental version interleaves generate and test phases of the basic Monahan's algorithm [Monahan 82], and is based on early pruning of redundant partially built linear vectors. The pruning for Q-functions has been investigated and proposed in [Cassandra et al. 97]. We have proposed a modification that allows to do early pruning of partially built linear vectors also accross different actions, based on upper bound estimates.

- New Gauss-Seidel improvement of the exact value iteration algorithm for the infinite discounted horizon problems. The method improves incrementally a piecewise linear and convex lower bound function by computing new linear vectors for selected points of the belief space and adding them to the previous step function. Thus a new linear vector obtained can be immediately used to compute next updates. This makes it possible to propagate improvements more rapidly. Also, it is not necessary to recompute the complete value function from scratch for every update step.

- Forward decision methods that find optimal or ϵ optimal control for a single initial information state. The methods work with bounds and incrementally expand and prune the decision tree. The methods proposed here include: breadth-first, bound-span heuristic, randomized heuristic and linear space algorithms. Also suggested is a new method that combines incremental decision tree expansion and incremental bounds improvement strategies using a metalevel decision procedure.

We have also explored and studied modifications of the standard POMDP model that use different state-observation dependencies. We have showed that some of the models (models with backward triggered observations and combination of backward and standard models) can be converted to belief-state MDPs with piecewise-linear and convex value functions that are computable, similarly to the standard model. Unfortunately, this no longer holds for models with observation delays.

Chapter 4

Approximation methods for solving POMDP problems

The major problem with the optimal and ϵ-optimal POMDP control solutions is that procedures for finding them are computationally very expensive. This makes exact methods practical only for POMDP models of very small size. A typical approach to such a problem is to relax the requirement of the solution accuracy and accept a "good" solution whenever it can be acquired fast. This reflects the common tradeoff between accuracy and speed. The exploration of more efficent approximation methods for POMDPs is the focus of this chapter.

4.1 Types of approximation methods

Approximation strategies for POMDPs include:

- approximations of value functions and policies;
- approximations (reductions) of the information-state MDP.

4.1.1 Approximations of value functions and policies

The main idea of the first approach is to approximate optimal value or control functions using simpler functions $\widehat{V} : \mathcal{I} \rightarrow \mathcal{R}$ or $\widehat{\mu} : \mathcal{I} \rightarrow A$. These functions are defined over the same information space, and are computed using simpler update rules.

The output of methods can be either a value function approximation or an approximation of the optimal policy. In the first case, the target approximate control is obtained from the approximate value functions in a standard way. For example, the control for the infinite discounted horizon is computed as:

$$\widehat{\mu}(I) = \text{argmax}_{a \in A} \sum_{s \in S} \rho(s,a) P(s|I) + \gamma \sum_{o \in \Theta_{next}} P(o|I,a) \widehat{V}(\tau(I,o,a)).$$

In the second case, the control policy is returned directly by an approximation routine. Both cases are usually closely related, and the computation of an approximate control policy often builds on approximate value function solutions.

4.1.2 Approximation (reduction) of the model

The second approach reduces the information-state MDP constructed for the POMDP model. The primary target of reduction strategies is the information state space. The information space is approximated by a feature space $\widehat{\mathcal{I}}$, which is usually of smaller size and summarizes the important characteristics of the state with regard to control. The resulting approximate model is then used to compute value or control functions defined over the feature space $\widehat{V} : \widehat{\mathcal{I}} \to \mathcal{R}$ and $\widehat{\mu} : \widehat{\mathcal{I}} \to A$. The approximate value and/or control functions for the original information space are then computed by mapping the information state to a feature state and using associated feature-based value and control functions.

4.1.3 The combination of the two approaches

The two approximation approaches are not exclusive and can be combined when needed. This leads to the approximation on the level of the model, as well as on the level of functions defined over the new feature space.

4.1.4 The structure of the chapter

The objective of this chapter is to describe and analyze various new and known approximation methods. The primary focus will be on methods that approximate optimal value functions. These are based on approximate versions of exact dynamic programming (value iteration) updates described in the previous chapter. Such updates can then be applied to compute both finite and infinite discounted horizon problems. At the end of the chapter the main ideas of alternative approximation strategies that include policy approximation (section 4.9) and model reduction (section 4.10) will be described.

All methods designed and described here can be applied to belief space POMDP models with sufficient belief information space. However, some of them are more general and suitable for other POMDP models as well, for example models with time lags. The description of value function approximation methods includes also proofs of their properties, namely bound and convergence properties. Some of the proofs are new, but some are originally due to other researchers and are reproved here. The reason for doing this is to provide a uniform view in which methods and their properties are described with regard to the approximate updates they implement. This in turn simplifies their theoretical comparison. The performance of the value function approximation methods discussed in this chapter will be experimentally tested and compared in the Chapter 5.

4.2 Value function approximations

4.2.1 Using approximate value functions to compute control response

Let $\widehat{V}$ and $\widehat{Q}$ denote approximations of value and action value functions and let $\widehat{\mu}$ stand for the approximate control function resulting from it. Then the control function $\widehat{\mu}$ for the infinite discounted horizon problem and a belief space POMDP can be defined using the approximate value function as:

$$\widehat{\mu}(b) = \text{argmax}_{a \in A} \sum_{s \in S} \rho(s,a) b(s) + \gamma \sum_{o \in \Theta_{next}} P(o|b,a) \widehat{V}(\tau(b,o,a))$$

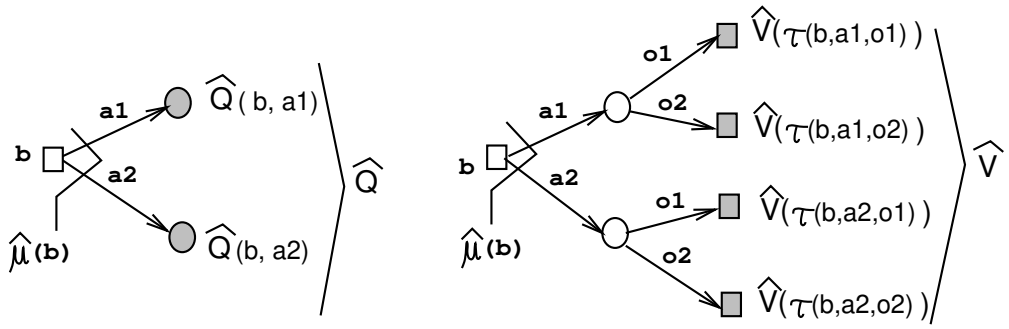


Figure 4-1: Computing control response using action-value and value function approximations.

or using $\widehat{Q}$ function as:

$$\widehat{\mu}(b) = \text{argmax}_{a \in A} \widehat{Q}(b, a).$$

Similar formulas can be written and used for the finite horizon case.

The computation associated with selecting the control action is best viewed as the one step (partial or complete) expansion of the decision tree with an approximate value function used at the leaves of the decision tree. The complete one step expansion occurs when the approximation of the value function $\widehat{V}$ is used, while a partial one step decision tree expansion corresponds to the approximation with $\widehat{Q}$ functions. This is illustrated in figure 4-1. Assuming that the computation of a value for value and action value functions is comparable, the control response with action-value approximation should be faster. This is because we need to work with a tree with $|A|$ leaves, compared to the tree with $|A||\Theta_{next}|$ leaves. On the other hand one can expect that the memory requirement for storing $|A|$ approximate action-value functions will exceed requirements for remembering single value function. This is just another example of how speed can be traded for memory.

The idea of selecting a control action through a one step decision tree expansion can be pushed further. A control action can be selected using a larger size decision tree, with more expanded levels and an approximate value function used at its leaves. The reason for an expansion of the decision tree to more levels is similar to that in the exact forward methods, and one expects the value function to improve more with more expanded steps. However exact forward methods use and compute bounds, while in this case an arbitrary value function approximation can be plugged in.

4.2.2 Incremental methods

The value function approximation can be used by a control agent to select action responses in the on-line mode. When time permits, the control response can be further improved in two ways: using a larger decision tree, assuming that the larger tree (which represents more iterations or recursions) makes it possible to obtain a better value function at the root of the tree; or directly by improving the value function $\widehat{V}$ that is used at the leaves of the tree.

The selection of the control response can be implemented using simple or more complex anytime algorithms. The algorithm performs the decision tree expansion, and subsequent control decision improvement, continuously up to the occurence of some critical event. Similarly

the algorithm can be designed to incrementally improve the approximate value function it uses, or combine both improvement strategies.

4.2.3 The role of value function bounds

The complete optimal or ϵ-optimal value function is hard to compute. However, one can often benefit from knowing the approximate range in which the optimal value function can be found. Such a range can be identified using various approximate methods that are guaranteed to produce upper and lower bounds of the optimal value function.

Bound methods can be used within the POMDP framework in several ways. They can provide a good initial value function for the exact version of the value iteration algorithm, or can be combined and interleaved with steps of exact methods. For example bounds can be used to prune early suboptimal actions and thus reduce the complexity of the exact problem solving routines. The important thing in this context is that bounds can often be improved and further tightened with exact iteration (dynamic programming) steps. For the value iteration case, this is because the mapping H is an isotone contraction, and an exact update step applied to a bound always preserves the bound and improves the approximation.

A typical problem with selecting actions is that an approximate value function $\widehat{V}$ can be significantly different from the optimal value function V^*. Then a control function $\widehat{\mu}$ computed from $\widehat{V}$ using standard equation may in fact lead to an expected reward that is far lower than the expected reward suggested by $\widehat{V}$. The important question is whether we can guarantee that the action found does not fall below treshold value. This can be achieved by selecting the action using the value function that lower-bounds the optimal value function. That is, when $\widehat{V} \leq V^*$ holds for the infinite discounted problem, a control function:

$$\widehat{\mu}(b) = \text{argmax}_{a \in A} \sum_{s \in S} \rho(s,a) b(s) + \gamma \sum_{o \in \Theta_{next}} P(o|b,a) \widehat{V}(\tau(b,o,a))$$

satisfies:

$$\begin{aligned} \widehat{V}^{\widehat{\mu}}(b) &= \sum_{s \in S} \rho(b, \widehat{\mu}(b)) b(s) + \gamma \sum_{o \in \Theta_{next}} P(o|b, \widehat{\mu}(b)) \widehat{V}(\tau(b,o,\widehat{\mu}(b))) \\ &= H\widehat{V}(b) \geq \widehat{V}(b) \end{aligned}$$

Altough a control function based on this lower bound does not fall below the expected reward suggested by an associated value function $\widehat{V}$ it can be too conservative. Then using a value function that is not a lower bound of the optimal function can still lead to better expected rewards.

4.2.4 Convergence and stability of iterative methods

Approximation methods for the infinite discounted horizon problem are usually built on the idea of approximate value iteration. These try to replicate exact value iteration using its approximate form:

$$\widehat{V}_{i+1} = \widehat{H}\widehat{V}_i$$

where $\widehat{V}$ stands for approximate value functions of various forms and $\widehat{H}$ defines a function mapping derived in some way from H that is used to compute updates of approximate value functions. The fixed point solution $\widehat{V^*} = \widehat{H}\widehat{V^*}$ or its close approximation would then represent

the intended output of the approximation routine.

The main problem with the iteration method is that in general it can converge to unique or multiple solutions, diverge or oscillate depending on the function form, value function mapping and initial values. Therefore, unique convergence cannot be guaranteed for an arbitrary mapping $\widehat{H}$. The convergence of a specific approximation method needs to be proved.

4.2.5 Described value function approximation methods

Value function approximations that will be described and discussed in the following include:

- MDP-based approximation (section 4.3);
- fast-informed bound method (section 4.4);
- blind (fixed) policy approximations (section 4.5);
- curve fitting methods (least square error) (section 4.6);
- grid-based interpolation-extrapolation methods (section 4.7);
- grid-based linear vector methods with Sondik's updates (section 4.8).

4.3 MDP-based approximations

The optimal value function V^* for both infinite discounted and finite horizon problems can be approximated by the MDP-based approximation method [Lovejoy 93] [Littman et al. 95a]. The method approximates the optimal value function for the POMDP using the optimal value function V^*_{MDP} for the fully observable case:

$$\widehat{V}(b) = \sum_{s \in S} b(s) V^*_{MDP}(s)$$

The basic idea of the MDP-based approximation is illustrated in figure 4-2. The approximate value function is described by a single linear function that is fully defined by values at critical points of the belief space. These correspond to optimal MDP values.

An MDP-based method can be described also by means of the value function updates that one would repeatedly apply over multiple steps of dynamic programming or value iteration procedures. Expresing the method using value function updates often simplifies the comparison of approximation and exact methods.

Let $\widehat{V}_i$ be a value function described by a single linear vector $\alpha_i^{MDP} = V_i^{MDP}$. Then a new value function $\widehat{V}_{i+1}$ that is obtained through MDP-based update is:

$$\begin{aligned} \widehat{V}_{i+1}(b) &= \sum_{s' \in S} b(s') \max_{a \in A} [\rho(s,a) + \gamma \sum_{s' \in S} p(s|s',a) V_i^{MDP}(s)] \\ &= H_{MDP} \widehat{V}_i(b). \end{aligned}$$

$\widehat{V}_{i+1}$ is described by a single linear vector with components:

$$V_{i+1}^{MDP}(s) = \rho(s,a) + \gamma \sum_{s' \in S} p(s|s',a) V_i^{MDP}(s)$$

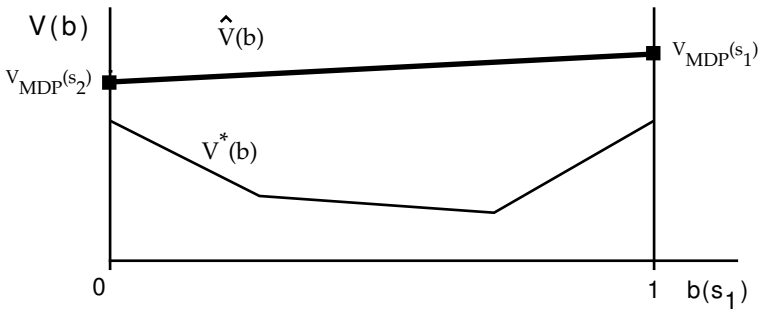


Figure 4-2: MDP-based approximation. Values at critical points of the belief space are obtained from the optimal MDP solution.

which matches exactly the update rule for the perfectly observable MDP. Therefore the MDP-based update always leads to the value function $\widehat{V}_{i+1}$ that is desribed by a single linear vector.

4.3.1 Upper bound property

The important property of this update rule is that it upper bounds the exact update rule. That is, $H\widehat{V}_i \leq H_{MDP}\widehat{V}_i$ holds. This property is trivial and follows from the fact that one cannot get a better solution with less information. The proof is shown bellow.

Theorem 9 *(Upper bound property of the MDP based update rule) Let $\widehat{V}_i$ be a value function described by a single linear vector $\alpha_i^{MDP} = V_i^{MDP}$. Then it holds that $H\widehat{V}_i \leq H_{MDP}\widehat{V}_i$.*

Proof.

$$\begin{aligned}
H\widehat{V}_i(b) &= \max_{a\in A}\sum_{s'\in S}\rho(s',a)b(s') + \gamma\sum_{o\in\Theta_{next}}\sum_{s\in S}\sum_{s'\in S}P(s,o|s',a)b(s')\alpha_i^{MDP}(s) \\
&= \max_{a\in A}\sum_{s'\in S}b(s')[\rho(s',a) + \gamma\sum_{s\in S}P(s|s',a)\widehat{V}_i^{MDP}(s)] \\
&\leq \sum_{s'\in S}b(s')\max_{a\in A}[\rho(s',a) + \gamma\sum_{s\in S}P(s|s',a)\widehat{V}_i^{MDP}(s)] \\
&= H_{MDP}\widehat{V}_i(b)
\end{aligned}$$

□

4.3.2 Infinite horizon solution

For the infinite discounted horizon case, the value function mapping H_{MDP} is an isotone contraction. Thus it leads to the unique fixed point solution $\widehat{V}^* = H_{MDP}\widehat{V}^*$. Such a solution upper-bounds the optimal value function.

Theorem 10 *Let V^*_{MDP} be an optimal value function for the associated fully observable MDP problem. Then $\widehat{V}^*(b) = \sum_{s\in S}b(s)V^*_{MDP}(s)$ is an upper bound on the optimal value function V^*, that is $V^* \leq \widehat{V}^*$.*

Proof. The proof is based on showing that $H\widehat{V}^* \leq \widehat{V}^*$ holds. Let V^*_{MDP} be an optimal solution to the perfectly observable case. Then it holds that:

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

Then for any b it holds that:

$$\begin{aligned}
H\widehat{V}^*(b) &= \max_{a \in A} \sum_{s' \in S} \rho(s', a) b(s') + \gamma \sum_{o \in \Theta_{next}} \sum_{s \in S} \sum_{s' \in S} P(s, o|s', a) b(s') V^*_{MDP}(s) \\
&= \max_{a \in A} \sum_{s' \in S} \rho(s', a) b(s') + \gamma \sum_{s \in S} \sum_{s' \in S} P(s|s', a) b(s') V^*_{MDP}(s) \\
&= \max_{a \in A} \sum_{s' \in S} b(s') [\rho(s', a) + \gamma \sum_{s \in S} P(s|s', a) V^*_{MDP}(s)] \\
&\leq \sum_{s' \in S} b(s') [\max_{a \in A} \rho(s', a) + \gamma \sum_{s \in S} P(s|s', a) V^*_{MDP}(s)] \\
&= \sum_{s' \in S} b(s') V^*_{MDP}(s) = \widehat{V}^*(b)
\end{aligned}$$

Knowing that H is isotone and that $H\widehat{V}^* \leq \widehat{V}^*$ holds it follows that $H^2\widehat{V}^* \leq H\widehat{V}^* \leq \widehat{V}^*$ must be satisfied as well. Using the isotonicity argument recursively, $V^* = HV^* \leq \cdots H^2\widehat{V}^* \leq H\widehat{V}^* \leq \widehat{V}^*$ must also hold. But this means that $V^* \leq \widehat{V}^*$ is true, which concludes the proof. □

4.3.3 Summary of the method

The main advantage of this method is that it is fast, as the MDP problem can be solved in time polynomial in the number of states and actions (see Chapter 2). As the MDP solution assumes perfect observability, the resulting value function is overly optimistic, and provides an upper bound on the optimal value function. The important property of the MDP-based solution is that it can be used to compute upper bounds also for POMDP models with observation delays. The idea here is the following: an upper bound on the value function for a model with no observation delays (standard model) should upper-bound also the value function constructed for the delayed model. Or in other words, one cannot do worse with information that is revealed ahead of time, than without it.

A disadvantage of the MDP based method is that it tends to ignore "investigative" actions, that is actions that can help to narrow the uncertainty about the true state of the process by enabling observations. This causes the Q_{MDP} based control to never choose such an action. This feature was noticed and pointed out by [Littman et al. 95a]. However, this does not hold, when control actions are selected based on value function V_{MDP}.

4.4 Fast informed bound method

The approximation obtained by the MDP-based approach can be improved by a new method – the fast informed bound method. The method uses a newly designed update rule that upper bounds the exact update rule similarly to the MDP-based method.

Let $\widehat{V}_i$ be a piecewise linear and convex value function represented by a set of linear vectors Γ_i. Then the new fast informed update rule corresponds to:

$$\begin{aligned}
\widehat{V}_{i+1}(b) &= \max_{a \in A} \sum_{s' \in S} \rho(s', a) b(s') + \gamma \sum_{o \in \Theta_{next}} \sum_{s' \in S} \max_{\alpha_i^k \in \Gamma_i} \sum_{s \in S} P(s, o|s', a) b(s') \alpha_i^k(s) \\
&= \max_{a \in A} \sum_{s' \in S} b(s') \left[\rho(s', a) + \gamma \sum_{o \in \Theta_{next}} \max_{\alpha_i^k \in \Gamma_i} \sum_{s \in S} P(s, o|s', a) \alpha_i^k(s) \right] \\
&= \widehat{H}_{FIB} \widehat{V}_i(b)
\end{aligned}$$

4.4.1 Complexity of a new update rule

An important feature of the new method is that it preserves piecewise linearity and convexness of the value function. That is, a new value function obtained from a piecewise linear and convex function is again piecewise linear and convex. Moreover, the resulting value function consists of at most $|A|$ different linear vectors, each corresponding to one action. This can be seen from the update formula, where a linear vector for an action a corresponds to:

$$\alpha_{i+1}^a(s') = \rho(s', a) + \gamma \sum_{o \in \Theta_{next}} \max_{\alpha_i^k \in \Gamma_i} \sum_{s \in S} P(s, o|s', a) \alpha_i^k(s)$$

That is, there are at most $|A|$ different α_{i+1}^as we can derive using the fast informed update rule. This property makes the rule very appealing as it guarantees not to grow the size of the set of linear vectors over value iteration (dynamic programming) steps. Thus the update is always efficient with regard to $|S|$, $|A|$, and $|\Theta|$. This is unlike the exact update that may lead to a function that consists of $|A||\Gamma_i|^{|\Theta_{next}|}$ linear vectors, which is exponential in the number of observations.

4.4.2 Bound property of the new update strategy

The important property of the new fast informed update rule is that it upper bounds the exact update rule. This is proven in the following theorem. In fact the steps of the proof were originally used to derive the rule.

Theorem 11 *(Upper bound property of the fast informed update rule) Let $\widehat{V}_i$ corresponds to a piecewise linear convex value function:*

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

Then it holds:

$$H\widehat{V}_i \leq H_{FIB} \widehat{V}_i.$$

Proof. For the exact update rule, $HV_i(b)$, we can write:

$$\begin{aligned}
HV_i(b) &= \max_{a \in A} \sum_{s' \in S} \rho(s', a) b(s') + \gamma \sum_{o \in \Theta_{next}} \max_{\alpha_i^k \in \Gamma_i} \sum_{s \in S} \sum_{s' \in S} P(s, o|s', a) b(s') \alpha_i^k(s) \\
&\leq \max_{a \in A} \sum_{s' \in S} \rho(s', a) b(s') + \gamma \sum_{o \in \Theta_{next}} \sum_{s' \in S} \max_{\alpha_i^k \in \Gamma_i} \sum_{s \in S} P(s, o|s', a) b(s') \alpha_i^k(s)
\end{aligned}$$

$$
\begin{aligned}
&= \max_{a \in A} \sum_{s' \in S} b(s') \left[\rho(s', a) + \gamma \sum_{o \in \Theta_{next}} \max_{\alpha_i^k \in \Gamma_i} \sum_{s \in S} P(s, o|s', a) \alpha_i^k(s) \right] \\
&= \max_{a \in A} \sum_{s' \in S} b(s') \alpha_{i+1}^a(s') \\
&= \widehat{H}_{FIB} V_i(b)
\end{aligned}
$$

□

The trick in deriving the new update rule is to exchange the sum and max operators in the exact update formula. This will effectively allow one to choose an optimizing (maximizing) linear vector for every observation and current state dimension independently. Contrary to this, in the exact method a single optimizing linear vector for every observation and all current state dimensions is selected.

4.4.3 Infinite discounted horizon case

$\widehat{H}_{FIB}$ is a contraction mapping under the max norm, much like H, with a fixed point solution $\widehat{V}^* \geq V^*$. The fact that H_{FIB} is a contraction mapping can be shown by using the proof in theorem 7 in section 3.2.3 (similarly we can show that H_{FIB} is isotone by following the steps of the proof of theorem 6). In the following we will show that the fixed point solution $\widehat{V}^*$ is an upper bound on the optimal value function.

Theorem 12 *Let V^* be an optimal value function and $\widehat{V}^*$ be a fixed point solution computed by the fast informed bound method. Then it holds that $\widehat{V}^* \geq V^*$.*

Proof Let $\widehat{V}_i$ correspond to a piecewise linear function that upper bounds the optimal value function, that is: $V^* \leq \widehat{V}_i$. Using the theorem 11 and the fact that H is isotone we can write:

$$V^*(b) \leq H\widehat{V}_i(b) \leq H_{FIB}\widehat{V}_i(b) = \widehat{V}_{i+1}(b)$$

Therefore for any $\widehat{V}_{i+1}$ it must hold $\widehat{V}_{i+1} \geq V^*$. As $\widehat{V}_{i+1}$ is again a piecewise linear upper bound (the initial condition), by extending this result to an infinite number of steps, $\widehat{V}^* \geq V^*$ follows. □

4.4.4 Extensions of the fast informed bound method.

The main idea of the fast informed bound method is to select the optimizing linear vector for every observation and current state dimension separately. This is unlike the exact case when we seek a linear vector that gives the best result for every observation and a combination of all current state dimensions. However, there is a lot of middle ground in between the two extremes. One can, for example, design an update rule that tries to choose optimal (maximal) linear vectors for every observation and for every set of disjoint pairs of current state dimensions. Of course, one can proceed further and try to choose linear vectors that optimize the expression for three dimensions, or in general for any disjoint partitioning of the state space dimensions.

Let $\mathcal{S} = \{S_1, S_2, \cdots, S^m\}$ be a partitioning of the state space S. Then one can costruct the following approximate update rule:

$$\widehat{V}_{i+1}(b) = \max_{a \in A} \sum_{s' \in S} \rho(s', a) b(s') + \gamma \sum_{o \in \Theta_{next}} [\max_{\alpha_i^k \in \Gamma_i} \sum_{s' \in S_1} \sum_{s \in S} P(s, o|s', a) b(s') \alpha_i^k(s) +$$

$$+ \max_{\alpha_i^k \in \Gamma_i} \sum_{s' \in S_2} \sum_{s \in S} P(s, o|s', a) b(s') \alpha_i^k(s) + \cdots +$$
$$+ \max_{\alpha_i^k \in \Gamma_i} \sum_{s' \in S_m} \sum_{s \in S} P(s, o|s', a) b(s') \alpha_i^k(s)]$$

For all possible partitionings, the result acquired by such an update is guaranteed to converge to the upper bound on the optimal value function (the proof is exactly the same as for the simple fast update rule). A single partitioning obviously leads to the exact update rule. The promising can be the exploration of heuristic partitioning schemes that would combine and optimize over states "closer" to each other.

4.4.5 Summary of the fast informed bound method

The idea of the fast informed update rule and its extension to arbitrary partitioning is a new one, and was reported for the first time in [Hauskrecht 97b]. The main advantage of the fast informed update rule is that the number of linear vectors acquired after the update is bounded by the number of actions. This makes the method very suitable for computing a good upper bound fast. Our experience with using the method for approximate control is very good, and will be discussed in the next chapter.

4.5 Blind policy approximations

The MDP approximation method gives us a value function that upper-bounds the optimal value function. It is acquired relatively easily by solving the fully observable MDP problem. A similar approach that minimizes expected rewards in a fully observable MDP, as opposed to maximizing them, can be used to compute a lower bound of the optimal value function. The bound property follows from the fact that under partial observability, one cannot do worse than by minimizing rewards under perfect observability. However, it is possible to come up with far better lower bounds. The method we propose here is based on the idea of blind control policies.

4.5.1 Blind policy

Definition 5 *(Blind control policy) Let $\pi = \{\mu_1, \mu_2, \cdots, \mu_i, \cdots\}$ be a control policy with control functions $\mu_i : \mathcal{I} \rightarrow A$, where $\mathcal{I}$ denotes information vector space. The policy is called* blind *when control functions $\mu_i \in \pi$ map all information states to a single control action, that is all μ_i are of the form $\mu_i : \mathcal{I} \rightarrow \{a_i\}$ with $a_i \in A$ denoting a single action.*

The main feature of a blind control policy is that it ignores all observations. The value function corresponding to the blind policy π is computed within the fully observable Markov process model as:

$$\widehat{V}(b) = \sum_{s \in S} b(s) V_{MDP,\pi}(s).$$

The blind policy method can be described by means of value function updates, similarly to other methods. Let π_1 denotes the first element (action) of the policy π, and π_{+1} denote its remainder, that is, policy π without its first element. Let $\widehat{V}_i$ be a single linear vector $\alpha_i = V_i^{\pi+1}$

that corresponds to the remainder of the blind policy. Then:

$$\begin{aligned}\widehat{V}^{\pi}_{i+1}(b) &= \sum_{s' \in S} b(s') \max_{a \in A} \left[\rho(s,a) + \gamma \sum_{s' \in S} p(s|s',a) V_i^{\pi+1}(s) \right] \\ &= H^{\pi_1} \widehat{V}_i^{\pi+1}(b)\end{aligned}$$

The fact that a blind control policy ignores all observations means that it should not provide better control than the optimal policy that utilizes all available information. Thus a blind policy update should always lower bound the exact value function update.

Theorem 13 *(Lower bound property of a blind policy update) Let π be an arbitrary blind policy, π_1 be its first element and π^{+1} its remainder. Let $\widehat{V}_i^{\pi+1}$ be a value function corresponding to π_{+1} that consists of a single linear vector $\alpha_i = V_i^{\pi+1}$. Then it holds:*

$$\widehat{V}^{\pi}_{i+1} = H^{\pi_1} \widehat{V}_i^{\pi+1} \leq H \widehat{V}_i^{\pi+1}.$$

Proof. Knowing that $\widehat{V}_i^{\pi+1}(b) = \sum_{s \in S} b(s) V_i^{\pi+1}(s)$ holds, we can write:

$$\begin{aligned}H\widehat{V}_i^{\pi+1}(b) &= \max_{a \in A} \rho(b,a) + \gamma \sum_{o \in \Theta_{next}} P(o|b,a) \widehat{V}_i^{\pi+1}(\tau(b,o,a)) \\ &= \max_{a \in A} \rho(b,a) + \gamma \sum_{o \in \Theta_{next}} P(o|b,a) \sum_{s \in S} p(s|b,o,a) V_i^{\pi+1}(s) \\ &= \max_{a \in A} \rho(b,a) + \gamma \sum_{o \in \Theta_{next}} \sum_{s \in S} p(s,o|b,a) V_i^{\pi+1}(s) \\ &= \max_{a \in A} \rho(b,a) + \gamma \sum_{s \in S} p(s|b,a) V_i^{\pi+1}(s) \\ &= \max_{a \in A} \sum_{s' \in S} \rho(s,a) b(s') + \gamma \sum_{s' \in S} \sum_{s \in S} p(s|s',a) b(s') V_i^{\pi+1}(s) \\ &= \max_{a \in A} \sum_{s' \in S} b(s') \left[\rho(s',a) + \gamma \sum_{s \in S} p(s|s',a) V_i^{\pi+1}(s') \right] \\ &\geq \sum_{s' \in S} b(s') \left[\rho(s',\pi_1) + \gamma \sum_{s \in S} p(s|s',\pi_1) V_i^{\pi+1}(s) \right] \\ &= H^{\pi_1} \widehat{V}_i^{\pi+1}(b) = \widehat{V}^{\pi}_{i+1}(b)\end{aligned}$$

□

The fact that a blind policy update always lower-bounds the exact update can be used to construct a lower bound approximation of the optimal value function by taking an arbitrary blind policy and computing its corresponding value function. The important thing is that the blind policy value function consists of a single linear vector that is computable within the fully observable framework. Moreover, every such linear vector can be directly combined with linear vectors obtained for other blind policies.

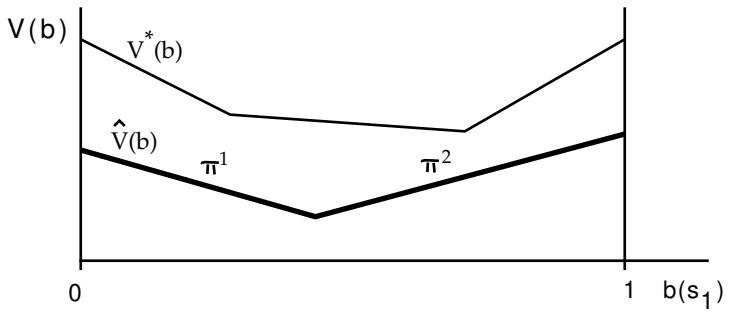


Figure 4-3: A two dimensional illustration of a piecewise linear and convex value function obtained by combining linear value functions for two blind policies π^1, π^2.

Combining value functions for more blind policies

A set of lower bound linear vectors computed for a set of blind policies can be combined into a piecewise linear and convex bound. Let α^π denote a value function acquired for some blind policy, that is $\alpha^\pi = V_\pi$, and let Γ be a collection of such functions. Then the piecewise linear and convex function:

$$\widehat{V}(b) = \max_{\alpha \in \Gamma} \sum_{s \in S} b(s)\alpha(s)$$

is a lower bound of the optimal value function $V^*(b)$. The idea of combining linear vector bounds is illustrated in figure 4-3. Here two linear vectors corresponding to two different blind policies are combined into a piecewise linear and convex lower bound.

4.5.2 Constructing a complete blind update rule

We have described how to combine solutions for a set of blind policies in order to provide a piecewise linear lower bound value function. This in principle allows one to compute a lower bound value function that combines results for all possible blind policies, by simply finding a value function for every policy and then combining the acquired linear functions into the resulting lower bound. Unfortunately the problem with such an approach is that the number of all possible blind policies can grow exponentially for n steps-to-go problem and it is infinite for the infinite discouted horizon problem. The reason for this is that for every blind policy there are $|A|$ new policies that start with some of the actions and continue with the previous policy afterwards.

The problem with the above approach is that it finds value functions also for policies that are clearly suboptimal, that is they are worse than other policies. This can be remedied by constructing a new update rule, the so-called *complete blind update* rule that effectively interleaves the enumeration of all blind policies and computation of their value functions.

Let $\widehat{V}_i$ be a piecewise linear convex function and Γ_i a set of linear vectors used to describe it. Now assume that every linear vector in Γ_i corresponds to some policy, that is $\Gamma_i = \{\alpha_i^{\pi^1}, \alpha_i^{\pi^2}, \cdots, \alpha_i^{\pi^m}\}$ where $\mathcal{P}_i = \{\pi^1, \pi^2, \cdots \pi^m\}$ denotes a set of policies. Then every policy in $\mathcal{P}_i$ can be extended in $|A|$ possible ways by selecting one of the actions. The value

function update for all possible actions and a policy π^j is:

$$\widehat{V}_{\pi_j}^{i+1}(b) = \max_{a \in A} \sum_{s' \in S} \rho(s', a) b(s') + \gamma \sum_{s' \in S} \sum_{s \in S} P(s|s', a) b(s') \alpha_i^{\pi^j}(s).$$

The optimal lower bound value function $\widehat{V}^{i+1}$ for all policies $\{\pi^1, \pi^2, \cdots, \pi^m\}$ is then obtained by combining results for all $\widehat{V}_{\pi_j}^{i+1}$:

$$\begin{aligned}
\widehat{V}^{i+1}(b) &= \max_{\pi^j \in \mathcal{P}_i} \widehat{V}_{\pi_j}^{i+1} \\
&= \max_{\pi^j \in \mathcal{P}_i} \max_{a \in A} \sum_{s' \in S} \rho(s', a) b(s') + \gamma \sum_{s' \in S} \sum_{s \in S} P(s|s', a) b(s') \alpha_i^{\pi^j}(s) \\
&= \max_{a \in A} \sum_{s' \in S} \rho(s', a) b(s') + \max_{\pi^j \in \mathcal{P}_i} \gamma \sum_{s' \in S} \sum_{s \in S} P(s|s', a) b(s') \alpha_i^{\pi^j}(s) \\
&= \max_{a \in A} \sum_{s' \in S} \rho(s', a) b(s') + \max_{\alpha_i^k \in \Gamma_i} \gamma \sum_{s' \in S} \sum_{s \in S} P(s|s', a) b(s') \alpha_i^k(s).
\end{aligned}$$

The blind update rule thus corresponds to:

$$\begin{aligned}
\widehat{V}^{i+1}(b) &= \max_{a \in A} \sum_{s' \in S} \rho(s', a) b(s') + \max_{\alpha_i^k \in \Gamma_i} \gamma \sum_{s' \in S} \sum_{s \in S} P(s|s', a) b(s') \alpha_i^k(s) \qquad (4.1) \\
&= H_{BU} \widehat{V}_i.
\end{aligned}$$

Alternative derivation of the complete blind update rule

Interestingly one can arrive at the blind update rule in a slightly different way by trying to approximate the exact value function update. The idea of this derivation is shown below.

Let Γ_i be a set of linear vectors describing an arbitrary piecewise linear convex function $\widehat{V}_i$. Then the exact value function update can be approximated as:

$$\begin{aligned}
H\widehat{V}_i(b) &= \max_{a \in A} \sum_{s' \in S} \rho(s', a) b(s') + \gamma \sum_{o \in \Theta_{next}} \max_{\alpha_i^k \in \Gamma_i} \sum_{s \in S} \sum_{s' \in S} P(s, o|s', a) b(s') \alpha_i^k(s) \\
&\geq \max_{a \in A} \sum_{s' \in S} \rho(s', a) b(s') + \gamma \max_{\alpha_i^k \in \Gamma_i} \sum_{o \in \Theta_{next}} \sum_{s' \in S} \sum_{s \in S} P(s, o|s', a) b(s') \alpha_i^k(s) \\
&= \max_{a \in A} \sum_{s' \in S} \rho(s', a) b(s') + \gamma \max_{\alpha_i^k \in \Gamma_i} \sum_{s' \in S} \sum_{s \in S} P(s|s', a) b(s') \alpha_i^k(s) \\
&= H_{BU} \widehat{V}_i(b)
\end{aligned}$$

Thus the main difference between the exact and blind update rules is that the max and the sum over next step observations are exchanged. This causes a choice of α vectors in the blind update rule to become independent of observations (once sum and max operations are exchanged, observations can be marginalized out). This is unlike the exact case in which α vectors are chosen separately for every observation.

Complexity of the blind update rule

Assume the complete blind update rule from the equation 4.1. Let $\alpha_i^{\iota(b,a)}$ be a linear vector that optimizes:

$$\max_{\alpha_i^k \in \Gamma_i} \sum_{s' \in S} \sum_{s \in S} P(s|s',a) b(s') \alpha_i^k(s)$$

for the fixed a and b. Then we can write:

$$\begin{aligned}
H_{BU} V_i(b) &= \max_{a \in A} \sum_{s' \in S} \rho(s',a) b(s') + \gamma \max_{\alpha_i^k \in \Gamma_i} \sum_{s' \in S} \sum_{s \in S} P(s|s',a) b(s') \alpha_i^k(s) \\
&= \max_{a \in A} \sum_{s' \in S} \rho(s',a) b(s') + \gamma \sum_{s' \in S} \sum_{s \in S} P(s|s',a) b(s') \alpha_i^{\iota(b,a)}(s) \\
&= \max_{a \in A} \sum_{s' \in S} b(s') \left[\rho(s',a) + \gamma \sum_{s \in S} P(s|s',a) \alpha_i^{\iota(b,a)}(s) \right] \\
&= \max_{a \in A} \sum_{s' \in S} b(s') \alpha_{i+1}^{b,a}(s')
\end{aligned}$$

where: $$\alpha_{i+1}^{b,a}(s') = \rho(s',a) + \gamma \sum_{o \in \Theta_{next}} \sum_{s \in S} P(s,o|s',a) \alpha_i^{\iota(b,a)}(s).$$

The complete blind update rule selects an optimizing alpha vector $\alpha_i^{\iota(b,a)}$ for any b independently of observations. This results in having at most $|\Gamma_i||A|$ possible linear vectors after update in $\widehat{V}_{i+1}$. This is in contrast to the exact update, where the number of possible vectors in the next step can grow exponentially with regard to the number of observations, and leads to $|A||\Gamma_i|^{|\Theta_{next}|}$ possible vectors. In this context, the blind update rule is best viewed as an approximation of the exact update rule (similarly to the fast informed bound).

Infinite horizon case

For the infinite discounted horizon problem the complete blind value function update H_{BU} is an isotone contraction, similarly to H. This can be shown by using same proofs as in theorems 6 and 7 in section 3.2.3. The contraction property implies that there is a unique fixed point solution and that the value iteration method based on the blind update rule converges to. It is easy to show that the value function corresponding to the fixed point solution is a lower bound of the optimal value function. The proof is shown below and it is identical to the one provided for the fast informed bound.

Theorem 14 *Let V^* be an optimal value function and $\widehat{V}^*$ be a fixed point solution computed by the complete blind update method. Then it holds that $\widehat{V}^* \leq V^*$.*

Proof. Let $\widehat{V}_i$ correspond to a piecewise linear lower bound of the optimal value function, that is: $\widehat{V}_i \leq V^*$. Using isotonicity of H and the fact that the blind policy update always lower bounds the exact update we can write:

$$\widehat{V}_{i+1}(b) = H_{BU} \widehat{V}_i(b) \leq H \widehat{V}_i(b) \leq V^*(b).$$

Therefore $\widehat{V}_{i+1}$ satifies $\widehat{V}_{i+1} \leq V^*$. As $\widehat{V}_{i+1}$ is also piecewise linear lower bound (same as the initial condition), we can extend the result to an infinite number of steps, and $\widehat{V}^* \leq V^*$ must follow. □

4.5.3 Efficient blind policy methods

As shown above, one can compute the optimal lower bound (or its ϵ precision approximation) for all blind policies using the derived blind update rule. However the problem is that the value function may similarly to the exact update, grow with every iteration, causing an exponential increase in the size of the linear vector set. Thus, when we need the lower bound fast, the optimal bound might not be the best solution.

The easiest way to compute a good lower bound is to use a fixed set of blind policies. The bound value function for such a set is obtained by combining value functions computed within the perfectly observable framework for every policy in the set (see above). Note that value functions for a fixed set of blind policies can be computed efficiently both for the finite as well as infinite discounted horizon cases.

There are various strategies one can use to construct a set of fixed blind policies to be combined into the lower bound value function approximation. These may range from random to various heuristic strategies. For example in our work, when we need to construct a lower bound value function for the infinite discounted horizon problem fast, we use simple one-action policies. The advantage of this selection is that respective value functions are found simply by solving $|A|$ sets of linear equations:

$$V^a(s) = \rho(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^a(s')$$

where a denotes the action used by the one-action policy.

4.5.4 Extensions to the fixed policy method

The idea of fixed blind policies can be further extended into the *fixed policy* approach. The fixed policy method permits policies that condition actions on observations. This is unlike the blind policy where actions are sequenced unconditionally. The fixed policy approach has been suggested and used by Anthony Cassandra (personal communication) and can be nicely represented using policy graphs [Cassandra 94].

The sample fixed policy for the infinite discounted problem is illustrated in figure 4-4. The arrow points to the initial action, that is, an action that is executed by the policy first. Subsequent actions in the policy depend on the results of observations. The important property of this approach is that the value function for an arbitrary fixed policy is computable efficiently within the fully observable Markov model (efficiently with regard to the size of the policy graph). For example, assuming a state space $S = \{s_1, s_2\}$, the value function for the policy on figure 4-4 is obtained by solving the set of linear equations:

$$\begin{aligned}
V(x_1,s_1) &= \rho(s_1, action(x_1)) + \gamma \sum_{o \in \Theta_{next}} \sum_{s \in S} p(o,s|s_1, action(x_1)) V(next(x_1,o), s) \\
V(x_1,s_2) &= \rho(s_2, action(x_1)) + \gamma \sum_{o \in \Theta_{next}} \sum_{s \in S} p(o,s|s_2, action(x_1)) V(next(x_1,o), s) \\
V(x_2,s_1) &= \rho(s_1, action(x_2)) + \gamma \sum_{o \in \Theta_{next}} \sum_{s \in S} p(o,s|s_1, action(x_2)) V(next(x_2,o), s) \\
V(x_2,s_2) &= \rho(s_2, action(x_2)) + \gamma \sum_{o \in \Theta_{next}} \sum_{s \in S} p(o,s|s_2, action(x_2)) V(next(x_2,o), s) \\
&\dots
\end{aligned}$$

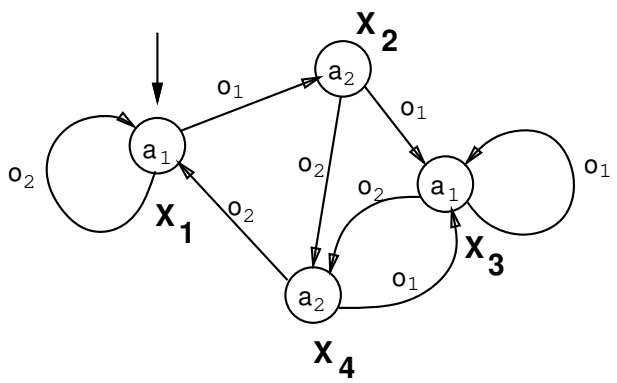


Figure 4-4: An example of a fixed policy. Actions in the policy can be conditioned on observations.

$$
\begin{aligned}
V(x_4, s_1) &= \rho(s_1, action(x_4)) + \gamma \sum_{o \in \Theta_{next}} \sum_{s \in S} p(o, s | s_1, action(x_4) V(next(x_4, o), s) \\
V(x_4, s_2) &= \rho(s_2, action(x_4)) + \gamma \sum_{o \in \Theta_{next}} \sum_{s \in S} p(o, s | s_2, action(x_4) V(next(x_4, o), s)
\end{aligned}
$$

where $action(x)$ corresponds to the action associated with node x of the policy graph, $next(x, o)$ represents a node one gets to after being in node x and seeing the observation o.

Once the system is solved, a value function corresponding to the policy π that starts at node x_1 is computed as:

$$\widehat{V}_\pi = \sum_{s \in S} b(s) V(x_1, s).$$

Note that by solving the above system of equations one effectively acquires solutions not only for the policy that starts at node x_1, but also solutions for policies that start at x_2, x_3 and x_4.

Properties of fixed policies

A value function corresponding to a fixed policy provides a lower bound of the optimal value function, similarly to the blind policy case. This is because any fixed policy is at most equivalent to the optimal policy, and thus it cannot improve on the optimal value function.

As every fixed policy is represented by a single linear vector that lower bounds the optimal value function, a convex combination of results for more fixed policies is possible and preserves the lower bound. That leads to a value function:

$$\widehat{V}(b) = \max_{\alpha \in \Gamma} \sum_{s \in S} b(s) \alpha(s)$$

that consists of a set of linear vectors Γ, each corresponding to one fixed policy. Then $\widehat{V}(b)$ provides a lower bound of the optimal value function: $\widehat{V} \leq V^*$.

4.5.5 A summary of a blind policy method

The blind policy approach provides means for computing value functions that lower bound the optimal value function. There are various versions of the method that work either with a fixed set of blind policies, or try to compute optimal lower bound that correspond to all possible blind policies, thus making methods more or less efficient. Although such bounds are often not very tight, they very often provide a very good start for exact methods or other approximation methods that are able to tighten the bound more. The important property of the blind policy method is that it can be applied to lower-bound exact value function updates for an arbitrary POMDP model [1].

The idea of blind policies can be extended to a more general fixed policy approach that computes lower bound value functions based on more complex polices that permit conditioning of actions using observations. Unfortunately, the fixed policy approach leads to a lower bound only for belief space POMDPs; thus it is not as widely applicable as the blind policy approach.

4.6 Approximation of a value function using curve fitting (least-squares fit)

A common way to approximate a function over continuous space is to use curve fitting techniques. This approach uses a predefined parametric model of the function and values associated with a finite set of points. The strategy then seeks the best possible match between model parameters and observed point values. The best match can be defined using various criteria, most often the least-squaress fit criterion. In this method parameters of the model function are fit to reduce the squared errors for all sample points, that is to reduce:

$$Error(f) = \frac{1}{2}\sum_{j}[y_j - f(b_j)]^2$$

where b_j and y_j correspond to the belief point and its associated value. The index j ranges over all points of the sample set.

The nice feature of the least-squares fit method is that it can be implemented in various forms, for example, using an exact or stochastic version of the gradient descent method.

4.6.1 Versions of least-squares fit

Let f denote a parametric value function over the belief space with adjustable weights $\overline{w} = \{w_1, w_2, \cdots, w_k\}$. Then the least-squares fit method can be implemented using any of the suitable optimization procedures, e.g.:

- A dedicated procedure that selects least-squares error weights $\overline{w}$ for f based on all sample points and their associated values;
- A gradient descent method that adjusts weights gradually in the error-reducing direction.

The on-line (or instance based) version of the least-squares error corresponds to the well-known delta rule (see e.g. [Rumelhart et.al 86]). The delta rule allows for the gradual adjustment of the function parameters for every new sample seen. Let f be a function with parameters

[1] This can be shown using minor modification of the proof of the theorem 13.

(weights) $\overline{w} = (w_1, w_2, \cdots, w_k)$. Then the delta update rule for a weight w_i corresponds to:

$$w_i \leftarrow w_i - \alpha_i (f(b_j) - y_j) \frac{\partial f}{\partial w_i}|_{b_j}$$

where α_i is a learning constant, and b_j and y_j correspond to the last seen point and its value.

The gradient descent method requires the function to be differentiable with regard to adjustable weights. This means that one needs to use smooth approximations of the value function and this also in the case when the optimal value function is nondifferentiable (for example piecewise linear).

4.6.2 Combining value iteration and least-squares fit

The least-squares fit can be used to construct an approximate value iteration (dynamic programming) algorithm with a step: $\widehat{V}_{i+1} = H_{LSF}\widehat{V}_i$. In the context of POMDPs, this approach was used in the work of [Littman et al. 95a] and [Parr, Russell 95], where they used reinforcement learning updates to speed up the parameter learning process.

The major drawback of value iteration methods with the least-squares fit is that their stability is not guaranteed and that they can also diverge. This was shown in [Tsitsiklis, Van Roy 96], [Baird 95]. In general, this makes it impossible to guarantee that the least-squares approximation of the optimal value function or a reasonably close substitute will be found via value iteration. However, the behavior of the least-squares strategy combined with value iteration is not understood very well and it is still possible that under a suitable selection of a value function model, sampling points and initial value function one can guarantee the result to stabilize in some bounded region around the optimal least-squares choice.

Unfortunately issues of divergence and stability have not been considered and investigated to sufficient depth in AI and more work needs to be done in this area. The intuition behind the threat of divergence can be illustrated in the following. Assume that the target function in some belief space region is approximated by a value function that assigns larger values to points in the region (compared to actual values). Further, assume that such a region is actively used in the computation of new value function updates for a set of sample points in the iteration step, thus producing values that are larger than the true target values. Fitting such points and new values using the least-squares approach can then translate into an increase in the error in the badly estimated region. In general such an error can grow larger with more iterations, leading possibly to the amplification of the error (a kind of positive feedback) and to divergence.

Parallel and Gauss-Seidel value iteration algorithms

Value iteration, powered with a stochastic on-line version of a least-squares fit, can use either parallel or incremental (Gauss-Seidel) updates. In the first case, the value function from the previous step is fixed, and a new value function is computed from scratch using a set of belief point samples and values computed through one step expansion. Once the parameters are stabilized (by attenuating learning rates) the newly acquired function is fixed, and the process proceeds with another iteration. In the incremental (Gauss-Seidel) version, there is a single value function model that is both updated and used to compute new values at sampled points. Note that both versions are subject to the instability and the divergence threat, as described above.

4.6.3 Parametric function models

As pointed out earlier, the on-line version of the least-squares fit method requires a function model that is differentiable. The typical choice of a convex function is simple, and usually corresponds to linear $\widehat{V}$, linear $\widehat{Q}$ [Littman et al. 95a], or a quadratic function.

One interesting and relatively simple least-squares method is based on the least-squares approximation of linear action-value functions (Q-functions) [Littman et al. 95a]. Here the value function $\widehat{V}_{i+1}$ is approximated as a piecewise linear and convex combination of $\widehat{Q}_{i+1}$ functions:

$$\widehat{V}_{i+1}(b) = \max_{a \in A} \widehat{Q}_{i+1}(b,a)$$

where:

$$\widehat{Q}_{i+1}(b,a) = \rho(b,a) + \gamma \sum_{o \in \Theta_{next}} p(o|b,a)\widehat{V}_i(\tau(b,o,a)).$$

The least-squares fit approach is applied to approximate every linear Q-function. This leads to the approximation with $|A|$ linear vectors. Note that least-squares Q-function method is different from the fast informed bound method that also works with $|A|$ linear vectors. The main differences are that the fast informed bound updates linear vectors directly, and it guarantees an upper bound and unique convergence, while Q-function least-squares relies on updates at some number of sample points, and does not guarantee neither bound nor unique convergence.

More sophisticated parametric function models are possible as well. For example one convex parametric function model suggested in the literature is [Parr, Russell 95]:

$$\widehat{V}(b) = \left[\sum_{\alpha \in \Gamma} \left[\sum_{s \in S} \alpha(s) b(s) \right]^k \right]^{\frac{1}{k}}$$

where Γ stands for the set of linear vectors α with adaptive parameters to fit and k is a "temperature" parameter that provides a better fit to the underlying piecewise linear convex function for larger values. The function represents a soft approximation of a piecewise linear convex function, with the parameter k smoothing more or less the piecewise linear approximation.

4.6.4 Summary of least-squares fit

The main advantage of least-squares error methods is that they implement a relatively simple update rule that needs to compute new updates of values only for a finite set of sample points. The typical choice of a function used in approximations is simple, and usually relies on linear models. The advantage of such functions is that they reduce to relatively simple weight update rules. However, in principle one can use the outlined methods, also with more complex parameter functions that try to fit better the optimal value function (see e.g. [Parr, Russell 95]).

On the other hand, the quality of methods based on least-squares error depends strongly on a given function model, initial parameter values, as well as a choice of belief points used in the least-squares. Devising suitable function models as well as proper initial values is in many cases like providing information that we do not know and need to compute, for example the number of linear regions needed to approximate the resulting function. Another troublesome thing is its combination with the value iteration procedure. In general such a combination cannot guarantee the stability and convergence to the best possible approximation. Another disadvantage of methods based on least-squares fit is that the resulting approximation does not provide a bound, and therefore does not provide any clue or suggestion about the optimal

solution.

4.7 Grid-based interpolation-extrapolation strategies

A value function over the continuous belief space can be approximated nonparametrically by a set of grid points, their associated values and an interpolation-extrapolation rule that is used to estimate values at non-grid points. The main advantage of such a value function model is that it can be updated easily by computing new values only for a finite set of grid points.

Definition 6 *(Interpolation-extrapolation rule) Let $f : \mathcal{I} \rightarrow \mathcal{R}$ be a real valued function defined over the information space, $G = \{b_1^G, b_2^G, \cdots b_k^G\}$ be a set of grid points and $\Psi^G = \{(b_1^G, f(b_1^G)), (b_2^G, f(b_2^G)), \cdots, (b_k^G, f(b_k^G))\}$ be a set of point-value pairs. Then $R_G : \mathcal{I} \times \Psi^G \rightarrow \mathcal{R}$ that estimates a function value f for any point of the information space $\mathcal{I}$ using only values associated with grid points is called an interpolation-extrapolation rule*

Using the interpolation-extrapolation rule, the complete value function is updated easily by computing updates only for a selected set of grid points. Let $\widehat{V}_i$ be an arbitrary value function. Then new updated function $\widehat{V}_{i+1}$ is computed using grid-based interpolation-extrapolation update as:

$$\widehat{V}_{i+1}(b) = R_G(b, \Psi_{i+1}^G)$$

where values associated with every grid point b_j^G in Ψ_{i+1}^G are computed as:

$$\widehat{V}_{i+1}(b_j^G) = \max_{a \in A} \rho(b, a) + \gamma \sum_{o \in \Theta_{next}} P(o|b, a) \widehat{V}_i(\tau(b_j^G, o, a)).$$

The grid-based value function update can be described also using a value function mapping H_G as: $\widehat{V}_{i+1} = H_G \widehat{V}_i$.

A family of convex rules

A set of all possible interpolation-extrapolation rules is enormous. In our work we will focus on a set of *convex rules* that represents a relatively small but but very important subset of interpolation-extrapolation rules.

Definition 7 *(Convex rule) Let f be some function defined over the information space, $G = \{b_1^G, b_2^G, \cdots b_k^G\}$ be a set of grid points, and $\Psi^G = \{(b_1^G, f(b_1^G)), (b_2^G, f(b_2^G)), \cdots, (b_k^G, f(b_k^G))\}$ be a set of point-value pairs. The rule R_G for estimating f using values $f(b_1^G), f(b_2^G), \cdots f(b_k^G)$ is called convex when for every information state b the value $\widehat{f}(b)$ is computed as:*

$$\widehat{f}(b) = R_G(b, \Psi^G) = \sum_{j=1}^{|G|} \lambda_j f(b_j)$$

such that $0 \leq \lambda_j \leq 1$ for every $j = 1, \cdots, |G|$ and $\sum_{j=1}^{|G|} \lambda_j = 1$.

A convex function-approximation rule is a special case of the averager approximation scheme described by Gordon [Gordon 95a]. The slight difference is that Gordon's model allows one to express a bias that is independend of the sample (values at grid points). The family of convex rules includes rules very common in practice, like: *nearest neighbor, kernel regression, and point interpolation.*

Nearest neighbor

In the nearest nearest neighbor the value function for some point b is estimated using the value at the closest grid point, where closest is defined with regard to some metric over the information state space. Then for every information state b there is exacly one nonzero parameter $\lambda_j = 1$ and all other λs are zero. That is:

$$\widehat{f}(b) = R_G(b, \Psi^G) = f(b_j^G)$$

where $\| b - b_j^G \|_M \leq \| b - b_i^G \|_M$ holds for all $i = 1, 2, \cdots, k$. M represents a distance metric defined on the information space.

The nearest neighbor rule computes a value function using a single grid point. This leads to a piecewise constant function where regions with equal values correspond to regions with a common nearest grid point.

Kernel regression

The value computed by a nearest neighbour rule depends on a single grid point. This causes it to absorb all the biases introduced by such a point. In order to remedy this problem, one can compute the approximation using more grid points in its neighborhood. A function approximation rule that takes into an account more grid points and their associated values is kernel regression.

In kernel regression, λs represent normalized weights associated with grid points that are derived using some distance metric M. The approximate function $\widehat{f}(b)$ for an arbitrary information state b is computed as:

$$\widehat{f}(b) = R_G(b, \Psi^G) = \sum_{j=1}^{k} \lambda_j^b f(b_j^G)$$

where

$$\lambda_j^b = \beta \exp^{-\| b - b_j^G \|_M^2 / 2\sigma^2}$$

with β being a normalizing constant equal to:

$$\beta = \sum_{j=1}^{k} \exp^{-\| b - b_j^G \|_M^2 / 2\sigma^2},$$

and where σ is a parameter that flattens or narrows weight functions. The important property of a kernel regression rule is that it computes a smooth approximation of the function, unlike the nearest neighbor rule.

Point interpolation

The point interpolation rule not only prescribes, how values at grid points are combined, but also imposes an additional constraint that explicitly relates the grid points and λ coefficients used.

In the point interpolation, the approximate function $\widehat{f}(b)$ for an arbitrary information state

b is computed as:

$$\widehat{f}(b) = R_G(b, \Psi^G) = \sum_{j=1}^{|G|} \lambda_j f(b_j^G)$$

such that all additional constraints hold:

$$b = \sum_{j=1}^{|G|} \lambda_j b_j^G$$

$$0 \leq \lambda_j \leq 1 \qquad \text{for every} \qquad j = 1, \cdots, |G|$$

$$\sum_{j=1}^{|G|} \lambda_j = 1$$

The fact that grid points used to compute function approximation must always interpolate the unknown point will help us to show the upper bound property for belief state POMDPs. This topic will be discussed later in the section.

4.7.1 Properties of convex rules

A set of convex rules differs from other interpolation-extrapolation rules in many respects. In the following we will examine two properties of high importance for the computation of value function approximations. These are: isotonicity of the value function mapping H_G and the contraction property of H_G for the infinite discounted horizon.

Isotonicity of a value function mapping based on a convex rule

It is well known that the exact value function mapping H is isotone (see [Heyman, Sobel 84]). However we are interested in learning if the isotonicity of H is preserved in H_G. Although isotonicity is not guaranteed to be preserved for an arbitrary interpolation-extrapolation rule it can be shown that it is satisfied for every convex rule. That is: $U \leq V$ imples $H_G U \leq H_G V$.

Theorem 15 *(isotone mapping) A value function mapping based on convex rule H_G is isotone.*
Proof. The proof of isotonicity is simple and directly follows from the isotonicity of the original exact mapping H (see also [Lovejoy 93]). The isotonicity of value function mapping H implies that when $V \leq U$ then $HV \leq HU$ must hold. As grid-based value function mapping with a convex rule allows only nonnegative coefficients λ then H_G derived from H must be isotone as well. □

Convergence of value iteration with a convex rule

In general the mapping H_G for the infinite discounted horizon problem may not lead to the convergence of the value iteration method. However it is possible to show that it converges uniquely for all convex rules.

The proof of the convergence of the approximate value iteration with a convex rule is based on the reduction of the problem to the MDP problem with the same discount factor. Note that the convergence result is independent of the form of the optimal value function, and thus can be used not only for the standard POMDP models but also for models with observation channel lags or continuous state MDPs. For an alternative proof of convergence that uses the contraction property see [Gordon 95a].

Theorem 16 *Let $\widehat{V}^G$ be a grid-based value function approximation defined by a finite set G of grid points, their associated values $\{\widehat{V}(b_j^G) : b_j^G \in G\}$ and a convex rule R_G. Then a value iteration method with an update step:*

$$\widehat{V}_{i+1}^G = H_G \widehat{V}_i^G$$

converges to a unique fixed point solution $\widehat{V}_G^*$.

Proof. The main idea is to convert the problem of a grid-based update to an MDP update with the same discount factor. For any grid point b_j^G we can write:

$$\begin{aligned}
\widehat{V}_{i+1}(b_j^G) &= \max_{a \in A} \rho(b_j^G, a) + \gamma \sum_{o \in \Theta_{next}} p(o|b_j^G, a) \widehat{V}_i^G(\tau(b_j^G, a, o)) \\
&= \max_{a \in A} \rho(b_j^G, a) + \gamma \sum_{o \in \Theta_{next}} p(o|b_j^G, a) \sum_{k=1}^{|G|} \lambda_{j,k}^{o,a} \widehat{V}_i^G(b_k^G) \\
&= \max_{a \in A} \rho(b_j^G, a) + \gamma \sum_{k=1}^{|G|} \widehat{V}_i^G(b_k^G) \left[\sum_{o \in \Theta_{next}} p(o|b_j^G, a) \lambda_{j,k}^{o,a} \right]
\end{aligned}$$

Now denoting $[\sum_{o \in \Theta_{next}} p(o|b_j, a)^G \lambda_{j,k}^{o,a}]$ as $P(b_k^G|b_j^G, a)$, the whole problem can be reduced to the MDP problem with the identical discount factor γ, and with states corresponding to grid points:

$$\widehat{V}_{i+1}(b_j^G) = \max_{a \in A} \rho(b_j^G, a) + \gamma \sum_{k=1}^{|G|} P(b_k^G|b_j^G, a) \widehat{V}_i^G(b_k^G).$$

The prerequisite $0 \leq \lambda_j \leq 1$ for every $j = 1, \cdots, |G|$ and $\sum_{j=1}^{|G|} \lambda_j = 1$ guarantees that $P(b_k^G|b_j^G, a)$ can be interpreted as true probabilities.

It is well known (see e.g.[Puterman 94]) that the mapping H with a discount factor $0 \leq \gamma < 1$ for the MDP is a contraction mapping, and that the value iteration method based on it converges to a unique fixed point solution. Therefore the approximate value iteration method converges to the unique solution as well. □

Note that both the isotonicity and convergence proofs apply for any POMDP model, not only belief space POMDPs. Therefore by using any of the convex rules, we always guarantee the convergence of the grid based update for any POMDP, and this also despite the fact that we have no idea about the shape of their value functions.

Grid-based approximate value iteration algorithm

A convex rule can be used to construct a simple grid-based approximate value iteration algorithm. Such an algorithm is illustrated below. The algorithm starts from the initial value function $\widehat{V}_{init}$ and stops when a relative stopping criterion defined for grid point changes is satisfied. The algorithm implements a Gauss-Seidel version of the value iteration in which each newly obtained grid point value is used immediately to update values for other grid points.

Approximate value iteration $(\widehat{V}_{init}, |G|)$
    select a set of grid of points G of size $|G|$
    for every point $b \in G$
        compute $\widehat{V}_{init}(b)$ and store it in the $\widehat{V}^G$ definition
    repeat until the relative stopping criterion is met
        for every point b in G
            compute new update $\widehat{V}(b)$ and
            update the value in $\widehat{V}^G$
return $\widehat{V}^G$

4.7.2 Constructing grids

A problem that has been left open is related to the grid point selection. There are various methods to select grid points that include:

- regular grids;
- random grids;
- heuristic grids.

Regular grids [Lovejoy 91b] partition (triangulate) the belief space evenly to equal size regions. This is basically the same idea that is used to partition evenly the n-dimensional subspace of $\mathcal{R}^n$. In fact there is an affine transform that allows us to map isomorphically grid points in the belief space to grid points in the n-dimensional space (see [Lovejoy 91b] for the discussion).

In contrast to regular grids, random and heuristic grids do not provide any regular partitioning of the belief space. In the first case grids are selected randomly using sampling methods, in the second case various heuristics that bias the selection of points are employed.

The advantage of nonregular grids (sometimes called variable grids) is that any increase in the resultion of the grid can be achieved by simply addding new belief points. On the other hand, regular grids are restricted to a specific number of points, and any increase in the resolution of a grid is paid for by an exponential increase in the grid size. For example a sequence of regular grids for a 20-dimensional belief space (corresponds to a POMDP with 20 states) consists of 20, 210, 1540, 8855, 42504, $\cdots$ grid points[2]. This prevents one from using the method with higher grid resolutions for problems with larger state spaces.

Neccessary condition for the point interpolation grids

The nearest neighbor and kernel regression rules do not impose any special requirement on what the grid must look like or what points must be present. However, one can easily notice that the point interpolation grid must always include critical points of the belief simplex. The reason for this is that in order to make interpolation work for any point of the belief space,

[2] The number of points in the regular grid sequence can be computed as [Lovejoy 91b]:

$$|G| = \frac{(M + |S| - 1)!}{M!(|S| - 1)!}$$

where $M = 1, 2, \cdots$ is a grid refinement parameter.

critical points must be present. Otherwise, one would not be able to interpolate missing critical belief points or any points in their neighborhood.

4.7.3 Bound property of the point-interpolation rule

The isotonicity and convergence properties of grid-based methods with convex rules have been shown regardless of the form and shape of the optimal value function. But the fact that the optimal value function V^* is convex (holds for belief space POMDPs) allows one to say more about properties of a resulting approximate value function. More specifically it is possible to show that the value function computed by the grid-based update combined with point interpolation always upper-bounds the value function computed by an exact update (see [Lovejoy 91b], [Lovejoy 93])

Theorem 17 *(Upper bound property of a grid-based point interpolation update) Let $\widehat{V}_i$ be a piecewise linear and convex value function. Then it holds: $H\widehat{V}_i \leq H_G\widehat{V}_i$.*

Proof. The proof is based on Jensen's inequality. Let $\widehat{V}_i$ be a piecewise linear convex function and $G = \{b_1^G, b_2^G, \cdots b_k^G\}$ be set of grid points used in the point interpolation update. Let b be a belief point such that $b = \sum_{j=1}^{k} \lambda_j^b b_j^G$ and such that $0 \leq \lambda_j^b \leq 1$ and $\sum_{j=1}^{k} \lambda_j^b = 1$ hold.

As an exact update for a belief space POMDP preserves piecewise linearity and convexness, we know that $H\widehat{V}_i$ is piecewise linear and convex. Then for a belief point b we can write:

$$\begin{aligned} H\widehat{V}_i(b) &= H\widehat{V}_i(\sum_{j=1}^{k} \lambda_j^b b_j^G) \\ &\leq \sum_{j=1}^{k} \lambda_j^b \left[H\widehat{V}_i(b_j^G) \right] = H_G\widehat{V}_i(b) \end{aligned}$$

where the upper bound follows from Jensen's inequality. □

Infinite discounted horizon solution

A value function mapping H_G implementing a convex rule has been shown to satisfy the isotone contraction property for the infinite discouted horizon problem. That means, there is a fixed point solution $\widehat{V}^* = H_G\widehat{V}^*$ the value iteration method will converge to. The fact that the grid-based point interpolation update upper bounds the exact update can be used to show that the approximate value iteration method converges to the value function that upper bounds the optimal value function for belief state POMDPs, that is: $\widehat{V}^* \geq V^*$ [Lovejoy 91b, Lovejoy 93].

Theorem 18 *(Upper bound property of a fixed point solution) Let H be a value function mapping for the POMDP problem with a sufficient belief information space and H_G be a value function mapping constructed from it using a grid-based point interpolation rule. Then the fixed point solution $\widehat{V}^* = \widehat{H}_G\widehat{V}^*$ is an upper bound on the optimal value function V^*, i.e. $V^* \leq \widehat{V}^*$.*

Proof. Let $\widehat{V}_i$ correspond to a piecewise linear function that upper bounds the optimal value function, $V^* \leq \widehat{V}_i$. Then using the result of the previous theorem and the fact that H is isotone, we can write:

$$V^*(b) \leq H\widehat{V}_i(b) \leq \widehat{V}_i(b)$$

and

$$V^*(b) \leq H\widehat{V}_i(b) \leq H_G\widehat{V}_i(b)$$

As $H\widehat{V}_i(b)$ is a piecewise linear and convex function (initial assumption) and both H and H_G are isotone we can write:

$$V^*(b) \leq H^2\widehat{V}_i(b) \leq H_G H\widehat{V}_i(b) \leq H_G^2\widehat{V}_i(b)$$

Knowing that both H and H_G are contractions and converge to their respective fixed point solutions, then applying the previous step repeatedly infinitely many times the following must be satisfied:

$$V^*(b) = HV^*(b) \leq H_G\widehat{V}^* = \widehat{V}^*$$

□

This means that the approximate value iteration method with a grid-based point interpolation rule computes upper bound value functions. Note that neither the kernel regression nor the nearest neighbor can guarantee any bound property.

Constructing the interpolation rule

The efficiency of the grid-based point interpolation update depends strongly on the efficiency of the implentation of such an interpolation rule. The interpolation rule must first select a set of points from the grid G suitable for interpolation, that consists of at least $|S|$ linearly independent belief points for any nonboundary point of a belief simplex. In general there can be $\binom{|S|}{|G|}$ possible minimal sets, and finding the best interpolating set can be time-consuming. One possible solution to this is to use regular grids (see above) that evenly partition the belief space and allow one to choose an interpolating set efficiently. However such grids must use a specific number of points and any increase in the resolution of a grid is paid for by an exponential increase in the grid size. This prevents one from using the method with higher grid resolutions for problems with larger state spaces.

To provide for more flexibility of the method, we have proposed a new point interpolation method that can use arbitrary grids and is guaranteed to run in time linear in the size of the grid [Hauskrecht 97b]. The rule builds on the fact that any point b of the belief space of dimension $|S|$ can be easily interpolated with a set of grid points that consists of an arbitary point $b' \in G$ and $|S|-1$ critical points of the belief simplex (critical points correspond to $(1,0,0,\cdots), (0,1,0,\cdots)$, etc.). That is, for any grid point $b' \in G$ there is a simple interpolating set that allows one to compute a linear interpolation $\widehat{V}_i^{b'}(b)$ at an arbitrary point b. As for any convex function the interpolation guarantees an upper bound, the tightest possible bound value achieved for a set of grid points can be chosen:

$$\widehat{V}_i(b) = \min_{b' \in G} \widehat{V}_i^{b'}(b).$$

The value function approximation corresponding to the described point interpolation rule is illustrated in figure 4-5. The approximation is characterized by its "saw" shape, which is influenced by the choice of the interpolating points.

The proposed interpolation rule can be computed in $O(|G||S|)$ time, which is linear in the size of the grid. This makes it a good candidate to use for a larger number of grid points. Also, any increase in the grid resolution is very easy, as one simply needs to add new points to the previous ones. The simplicity of grid extension allows one to implement relatively easily

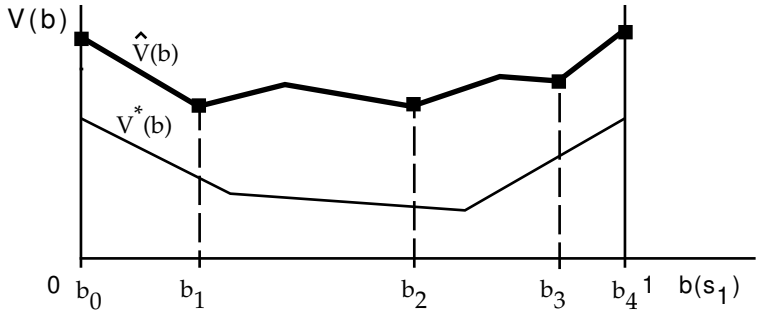


Figure 4-5: A two dimensional illustration of the simple linear point interpolation rule. The candidate interpolating set is restricted to a single internal point of the belief space.

various efficient incremental strategies that improve the upper bound for the infinite discounted horizon problem.

Incremental grid based methods

A simple incremental improvement algorithm for the infinite discounted horizon problem is illustrated below. The algorithm starts from the initial upper bound $\widehat{V}_{init}$, expands the grid gradually in k point increments, and uses Gauss-Seidel updates for points in the active grid. As the grid size is bounded by linear growth, the algorithm is guaranteed to run efficiently for a fixed number of iterations.

Incremental upper bound $(k, \widehat{V}_{init})$
    select an initial set of grid points G
    for every point $b \in G$
        compute $\widehat{V}_{init}(b)$ and store it in $\widehat{V}^G$ definition
    repeat until the stopping criterion is satisfied
        repeat until the grid expansion criterion is met
            for every point b in G
                compute new update $\widehat{V}(b)$ and
                update the value in $\widehat{V}^G$
        select a set of k points G_{EXP} to expand G
        for every $b \in G_{EXP}$
            add b to G and $\widehat{V}(b)$ to $\widehat{V}^G$
return $\widehat{V}^G$

An initial bound $\widehat{V}_{init}$ can be computed using either MDP-based approximation or the fast informed bound method presented earlier. Note that MDP-based approximation corresponds exactly to the solution obtained by the approximate value iteration with the point interpolation rule and with the grid that consists solely of critical belief points.

Constructing a heuristic point interpolation grid

In general the quality of bounds produced by the grid-based point interpolation method is strongly influenced by a grid selection strategy. The advantage of our simple interpolation rule is that it does not enforce a specific grid (like regular grids). Thus it can be easily combined with an arbitrary selection method, which may include various heuristics.

A heuristic method for selecting grid points that we have designed, implemented and tested attempts to maximize improvements in bound values using stochastic simulations. The method builds on the fact that every grid suitable for interpolation must include critical points (otherwise the interpolation cannot be guaranteed). A value at any grid point b improves more when more precise values are used for its successor belief states, i.e. belief states that correspond to $\tau(b, a, o)$ for an optimizing action a and an observation o. Incorporating such points into the grid would then increase the chance of larger improvement of values associated with critical points. Naturally one can proceed with selection further, by incorporating succesor points for the first level successors into the grid set as well, and so on.

The stochastic simulation method samples likely successor belief points in the following steps:

1. select an action a that is optimal for b given the current upper bound value function;
2. select the next observation randomly according to the probability distribution $p(o|b, a)$
3. compute the next belief point $b^+ = \tau(b, o, a)$.

Similar stochastic simulation methods within the POMDP framework were used for example in [Parr, Russell 95] [Littman et al. 95a]. Note that other approaches for constructing heuristic grids for the point interpolation strategy are possible. One such approach has been proposed recently in [Brafman 97] and it refines the grid by examining differences in value function values at current grid points.

4.7.4 Extensions of the simple interpolation rule

The idea behind the simple interpolation rule can be extended further to improve the selection of interpolating sets used. For example, one can try to select interpolating sets that consist of two arbitrary belief points and $|S| - 2$ critical points, three belief points and $|S| - 3$ critical points, and so on, up to $|S|$ arbitrary belief points. However, these improvements are mostly paid for by an increased computational complexity asssociated with enumerating all plausible combinations. Note that the process of selecting points to be combined does not have to be done blindly and smart heuristics for focusing on suitable combinations can be utilized.

4.7.5 Summary of grid-based interpolation-extrapolation methods

The exact value function update can be approximated using a grid-based update rule. The rule computes value function updates for a finite set of information states (grid points) and uses interpolation-extrapolation techniques to derive new value function values for all other states. This makes it possible to efficiently derive a new value function.

There are numerous interpolation-extrapolation strategies. However most suitable and frequently applied interpolation-extrapolation rules belong to the family of convex rules. Updates based on convex rules are isotone and are guaranteed to converge for the infinite discounted horizon problem. The important thing is that this holds for any infomation state space and thus it covers an arbitrary POMDP model.

The fact that for belief space POMDPs the value function is known to be piecewise linear and convex can be used to show that any approximate update based on point interpolation upper bounds the exact update. Thus, it can be used to compute an upper bound of the optimal value function and this both for finite and infinite horizons.

4.8 Grid-based linear vector method (grid-based Sondik's method)

An alternate value function approximation method can be constructed by applying Sondik's approach for updating linear vectors (derivatives) to a grid of points [Lovejoy 93] [Hauskrecht 97b].

Let $\widehat{V}_i$ be a piecewise linear convex function described by a set of linear vectors Γ_i. Then a new candidate linear vector for a belief point b and action a can be computed efficiently as [Smallwood, Sondik 73]:

$$\alpha_{i+1}^{b,a}(s) = \rho(s,a) + \gamma \sum_{o \in \Theta_{next}} \sum_{s' \in S} P(s', o|s, a) \alpha_i^{\iota(b,a,o)}(s') \tag{4.2}$$

where $\iota(b,a,o)$ indexes a linear vector α_i in a set of linear vectors Γ_i (defining $\widehat{V}_i$) that maximizes the expression:

$$\sum_{s' \in S} \left[\sum_{s \in S} P(s', o|s, a) b(s) \right] \alpha_i(s')$$

for a fixed combination of b, a, o. The optimizing linear vector for a point b is then acquired by choosing the vector with the best overall value from vectors computed for all actions. That is, assuming Γ_{i+1}^b is a set of all candidate vectors, the resulting vector must satisfy:

$$\alpha_{i+1}^{b,*} = \text{argmax}_{\alpha_{i+1}^b \in \Gamma_{i+1}^b} \sum_{s \in S} \alpha_{i+1}^b(s) b(s).$$

The point based linear vector update is a basis of a number of exact algorithms (Sondik's, Cheng's) that update value function over iteration or dynamic programming steps. However exact methods require one to always find a complete set of points that seed new linear vectors and thus guarantee the complete update. Unfortunately the search for a complete set of points can also turn out to be a source of major inefficiency. In contrast to this approach a class of approximation methods can be based on incomplete sets of points that are easy to locate (via random, or efficient heuristic selection). Let H_{GL} denote a value function mapping that restricts linear vector updates to a set of arbitrary, and thus often incomplete, grid points G.

4.8.1 Lower bound property of the grid-based Sondik's update

In both exact and grid based updates one computes a set of linear vectors that define new piecewise linear and convex value functions. However if an incomplete set of points is used for the update, the resulting value function lower bounds the value function one acquires using the complete exact update rule. The proof of this is shown bellow.

Theorem 19 *(Lower bound property of the grid-based linear vector update). Let $\widehat{V}_i$ be a piecewise linear value function and G a set of grid points one uses to compute linear vector updates. Then it holds: $H_{GL}\widehat{V}_i \leq H\widehat{V}_i$.*

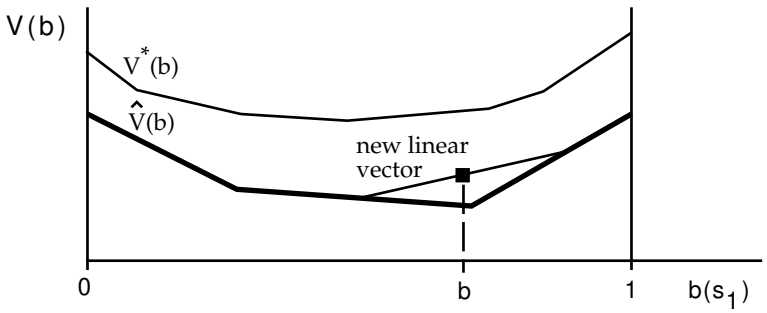


Figure 4-6: An incremental linear vector method. The lower bound piecewise linear function is improved by a new linear vector computed for a belief point b using Sondik's method.

Proof. The proof is trivial and is based on a completeness argument. Let Γ_{i+1} be a set of optimizing linear vectors computed for a grid set G and $\widehat{V}_i$. As points used for a grid-based update may be incomplete, the resulting value function defined by Γ_{i+1} may lack useful linear vectors that optimize (maximize) a value function for some region of the belief space. Thus $H_{GL}\widehat{V}_i \leq H\widehat{V}_i$ must hold. □

4.8.2 Infinite horizon case

The grid-based linear vector update method uses an incomplete set of points. Because of this, a value function mapping H_{GL} for the infinite discounted horizon case does not have to satisfy a contraction property and a value iteration method based on such a mapping does not have to converge. The grid-based update rule with an incomplete set of grid points can lead to various behaviors over value iteration steps, most often oscillations.

In order to guarantee the stability and convergence of the value iteration method when working with an incomplete set of points, we propose the following incremental method that gradually improves the piecewise linear and convex lower bound value functions.

Incremental lower bound method

Assume that $\widehat{V}_i \leq V^*$ is a convex piecewise linear lower bound on the optimal value function, defined by a linear vector set Γ_i, and let α_b be a linear vector for a point b that is computed from $\widehat{V}_i$ by the Sondik's method. As it holds that $\widehat{V}_i(b) \leq \sum_s b(s)\alpha_b(s) \leq V^*(b)$ (from the isotonicity of H) one can construct a new improved value function $\widehat{V}_{i+1} \geq \widehat{V}_i$ by simply adding new linear vector α^b to Γ_i [Hauskrecht 97b]. That is: $\Gamma_{i+1} = \Gamma_i \cup \alpha_b$.

The idea of the new update rule is illustrated in figure 4-6. Note that the rule can be easily extended to handle a set of grid points G. In such a case the new linear vector set is: $\Gamma_{i+1} = \Gamma_i \cup \Gamma^G_{i+1}$ where Γ^G_{i+1} consists of new linear vectors computed for grid points from Γ_i. However, it is advantageous to perform updates of grid points one by one, as this allows one to implement a much faster Gauss-Seidel approach. Note also that after adding one or more new linear vectors to Γ_i, some of the previous linear vectors can become redundant and can be removed from the convex value function definition. Various ways to do this are discussed in [Monahan 82] [Eagle 84] [Cassandra 94].

A simple incremental lower bound algorithm is shown below. The algorithm starts from the initial lower bound $\widehat{V}_{init}$ (with a linear vector set Γ_{init}), selects a belief point and updates the existing lower bound with a new linear vector. An initial bound is computed using for example the blind or fixed policy approach discussed earlier.

Incremental lower bound $(\widehat{V}_{init})$
  set Γ defining current bound $\widehat{V}$ to Γ_{init}
  repeat until the stopping criterion is met
    select a belief point b
    compute new update α^b for b
    add the α^b to Γ
return $\widehat{V}$

The above Gauss-Seidel style algorithm tends to grow the size of the linear vector set $\widehat{\Gamma}_i$ with every iteration. However, this growth is only linear compared to the potentially exponential growth of exact methods. The major advantage of the method is that it gives room for an application of various point selection heuristics that can lead to a better and tighter linear vector set. Various modifications of the above incremental lower bound algorithm are possible, e.g. one can use a fixed set of grid points to be updated repeatedly, or one can select the points to be updated using some heuristics.

Heuristic point selection strategies

The update phase of the incremental lower bound method is not limited to a specific point choice. Thus one may combine it with arbitrary point selection strategies. The strategies can be based on simple random selection of grid points, or more sophisticated strategies based on various heuristics. Random grid selection strategies can be then used to benchmark the improvement from heuristic strategies.

With an objective to speed up the improvement of the bound, we have designed and implemented two relatively simple heuristic strategies that try to optimize updates of a bound value function.

The first strategy attempts to optimize updates only at critical points by ordering them appropriately. It builds on the fact that states with higher expected rewards (e.g. some designated goal states) backpropagate their effects locally. Therefore it is desirable that states in the neighborhood of the highest reward state are updated first, and distant ones later. The strategy for ordering critical points uses the current value function to identify the highest expected reward states, and the POMDP model to determine local dependencies and order neighboring states.

The second strategy uses the idea of stochastic simulation, similar to the one used in the upper bound method. The strategy generates a sequence of belief points that can result from an (initial) belief point through simulation, such that a sequence of belief points with higher probablity are more likely to be generated. The points of the sequence are then used in reverse order to update the current value function.

The two heuristic strategies can be combined into one two-tier strategy, in which the top level strategy orders critical belief points, and the lower level strategy uses stochastic simulation to generate a sequence of belief points that are likely to result from a given critical point.

4.8.3 Summary of the grid based linear vector method

The grid-based linear vector method represents a refinement of the Sondik's method to arbitrary grids. The grid-based update leads to a piecewise linear convex function that is defined using a smaller number of linear vectors and that lower bounds the exact update. The main advantage of the grid-based method compared to the exact update is that it can compute a value function approximation fast, not wasting time by trying to locate all belief points that would guarantee the exact update.

The grid-based linear update rule can be turned into a new incremental linear update rule for the infinite discounted horizon problem. The rule gradually improves a piecewise linear and convex lower bound. It uses the Gauss-Seidel style of updates and avoids the need to recompute the value function from scratch for every iteration step.

4.9 Approximation of policies

Although a control response can be always computed from the value function approximation through one step decision tree expansion, it is also possible to compute the policy directly. This approach requires control functions that are defined over possibly infinite information space in some flexible and finite form. In the following we will briefly describe a method that uses policy graphs (trees) [Cassandra 94] [Littman 94].

4.9.1 Representing control using policy (control) trees

A policy for the belief space POMDP framework and for both finite and infinite horizon problems can be represented using a policy tree [Cassandra 94] [Littman 94]. The policy tree consists of nodes that are associated with action choices and links that represent conditional continuations of control choices based on observations. The policy tree can be also viewed as a collapsed decision tree in which decision nodes are substituted with a fixed action choice. For the infinite discounted horizon problems policy trees can be represented using policy graphs with cycles (wrap-around trees). An example of a policy graph for the infinite horizon problem was shown in figure 4-4 in section 4.5.4.

A policy graph can be used to represent any control policy, including the optimal one. In fact there is a strong correlation between policy tree representation and the structure of value function for belief information spaces, and one can construct a solution policy tree using a slight modification of the exact update value function [Littman 94]. In principle, every region of the belief space that is represented by a linear vector corresponds to a node in the policy tree, and links between policy tree nodes represent optimal choices of linear vectors used in update steps.

Constructing approximate policies using policy trees

Every node in the policy tree corresponds to a linear vector that describes a piecewise linear and convex value function corresponding to such a policy. The interesting thing is that we can compute the linear vector for any node and any fixed policy simply by solving a set of linear equations. This has been shown and described in section 4.5.4. That reflects the fact that it is relatively easy to compute a value function for a fixed policy, although one must not forget that the policy itself can be quite complex.

The fact that we know how to compute the value function for any fixed policy tree can be used to construct a policy approximation algorithm that starts from an initial policy tree, and by performing structural or action changes, gradually produces a better policy approximation.

Note that the improvement would be relatively easy to check as any fixed policy lower bounds the value function for the optimal policy. Such an approach can employ various heuristic strategies for making structural changes that are likely to further improve the quality of a policy.

The policy approximation approach outlined above has not been investigated to our knowledge, and thus offers a promising alternative to various value function approximation methods. The advantage of the approach is that optimal policies have less structure than optimal value functions, and therefore are representable more compactly.

4.9.2 Other policy approximation methods

Alternatively, a control function for a belief information space can be represented using a finite set of grid points, their associated actions, and a rule that defines how to determine an action for a nongrid point. Nearest neighbor is a simple rule choice and the action for any belief point is an action associated with the grid point closest to it. More complex rules, that select an action for a non-grid target belief point using actions associated with more than one grid points in its neighborhood can be also created. However, in such cases, one must provide a strategy for resolving conflicts when different actions are suggested by several relevant grid points.

An approximate policy can be constructed by computing control responses for all grid points from the value function approximation. This is a direct approach and works fine for both finite as well as infinite discounted horizon cases. But, when one needs to compute the approximate policy for the infinite discounted horizon case, it is possible take advantage of the form of the control function that needs to be found. Then it is possible to adapt the policy iteration method, described for the fully observable MDP, also to belief state MDPs. Such a method is also refered to as *approximate policy iteration* method[Bertsekas 95]. The method starts from some fixed policy, computes its value function approximation using an arbitrary method (computing a value function for a fixed policy is easier). Then, every action associated with a grid point representing a control function is checked to see if it improves the value function for such a point. If yes the change is made and process continues.

The main problem with approximate policy iteration is that it does not have to converge, and can oscillate among a set of policies. This is because of approximations, as it can happen that value function values for the "improved" policy may turn out to be worse than value function values for the previous policy.

4.10 Model based approximations

The main idea behind value function approximation methods was to replace the exact update rule with a more efficient approximation. In all cases the resulting value function was defined using the original information state space $\mathcal{I}$.

A complementary approach to the value function approximation is based on the approximation (reduction) of the information-state MDP. The reduction can target components of the information-state MDP or components of the underlying POMDP model (states, actions, observations, transitions, observation and cost models). The most typical approximations are those that in some way transform or reduce the sufficient information state space.

4.10.1 Approximation of the information state space

The approximation of the information-state space could be achieved by substituting more complex information space with a simpler *feature state space* [Bertsekas 95] [Tsitsiklis, Van Roy 96]. The feature space is usually of smaller size, summarizes the important characteristics of the information state with regard to the control, and is easier to manipulate and work with. Feature states (vectors) can be often viewed as abstractions or aggregations of sufficient information states.

The relation between the information and feature vectors is captured by a *feature extraction mapping* $\mathcal{F}$, that maps information states to feature states:

$$\mathcal{F} : \mathcal{I} \rightarrow \widehat{\mathcal{I}}.$$

Then, assuming the feature-based value and control functions:

$$\widehat{V}_{\mathcal{F}} : \widehat{\mathcal{I}} \rightarrow \mathcal{R}$$

$$\widehat{\mu}_{\mathcal{F}} : \widehat{\mathcal{I}} \rightarrow A$$

are known, one can express approximate value or control functions for the information state I as:

$$\widehat{V}(I) = \widehat{V}_{\mathcal{F}}(\mathcal{F}(I))$$

$$\widehat{\mu}(I) = \widehat{\mu}_{\mathcal{F}}(\mathcal{F}(I)).$$

Note that while the optimal policies for the original information-state MDP are deterministic, the optimal feature-based policies for the same problem can be stochastic (see e.g. [Singh et al. 94]). However stochastic policies are harder to compute and deterministic feature-based policies are often assumed.

Feature vector updates

A feature vector $\widehat{I}_t$ at time t can be obtained in two different ways (see figure 4-7):

1. from the original information vector I_t, i.e. $\widehat{I}_t = \mathcal{F}(I_t)$;
2. from the previous step feature vector $\widehat{I}_{t-1}$, action a_{t-1} and observation o_t, i.e. $\widehat{I}_t = \tau_{\mathcal{F}}(\widehat{I}_{t-1}, o_t, a_{t-1})$.

In the first case the feature process is always associated with the underlying sufficient information process, and every information state is always mapped to the same feature state. On the other hand, when the feature process is defined as a separate process (can be described by a separate MDP) there is a potential for a continuous loss of information content due to the approximation of the information space. This may lead to the situation in which information and feature state processes are not tightly mapped (aligned) and a single information state can occur together with more than one feature state. This is also the reason why optimal policies for reduced models may be stochastic.

Constructing feature state space

A feature state space and associated feature extraction mapping introduce a bias telling what features of the problem need to be considered and what can be abstracted out. Note that the feature space together with the mapping in fact represents partitioning of the original

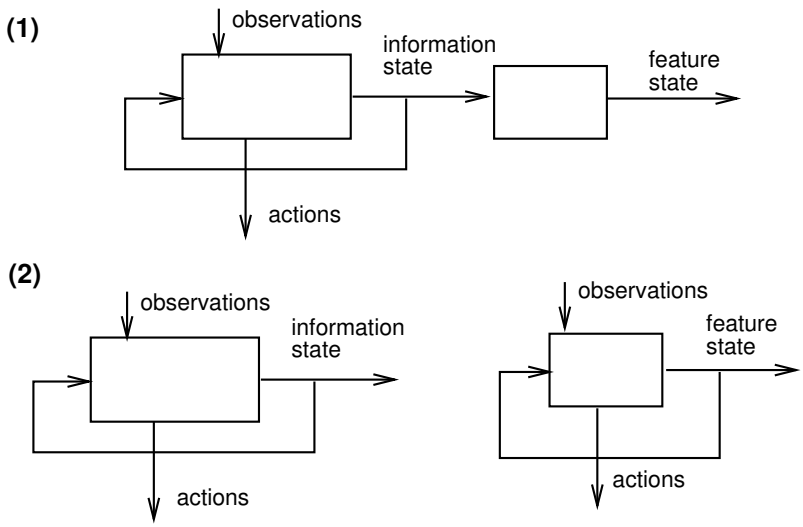


Figure 4-7: Two different models for updating feature states: 1. through feature extraction mapping from the underlying information state; 2. using a special feature-based update procedure that is independent of the information state update.

information space, with information states in the same partition being mapped to the same feature state. Information states in the same partition are treated as a single aggregate state on the feature level, leading to the loss of precision and approximation.

For the purpose of control one would like to use features that reduce the complexity of the state space and have the smallest possible effect on the quality of control. The feature space and related mapping can be either:

- defined by the designer or expert in the domain of interest;
- automatically inferred from the original model.

The first approach can be used to reduce the complexity of the original problem in areas where an expert is able to define the most important reductions. The feature space and a related mapping then incorporate knowledge reflecting the expert's intuition or experience about the control domain and about the importance of various problem characteristics to achieve better control. This approach can be very valueable especially for problems with large state or observation spaces.

In the second case the feature space and the feature mapping is inferred from the original more complex model. Usually the goal here is to come up with the feature space and the mapping that cuts down the complexity of the original information state as much as possible and that has also minimal possible effect on the quality of control that would result from the approximation. Such an approach is crucial for solving control problems for which an expert's knowledge is not available, or where one must work with large and complex models. This problem has not been sufficiently investigated, and remains open.

In the following we will discuss two representatives of feature-based approximations. These are based on:

- truncated histories;
- POMDP model reductions.

Other information-state reduction methods are possible as well. For example, [D'Ambrosio 96] proposed and tested reductions in which continuous belief space was transformed to its qualitative abstraction using κ-calculus [Goldszmidt 95].

4.10.2 Truncated history

One approach to information-state reductions is based on truncated histories [White, Scherer 94] [Platzman 77]. The approach builds on the heuristic saying that the decision about the control can be done reasonably well using only a set of recent actions and observations. The approach thus seeks the replacement of the complete information vector in both finite and infinite discounted horizon update formulas:

$$V(I_t) = \max_{a \in A} \sum_{s \in S} \rho(s,a) P(s|I_t) + \gamma \sum_{o \in \Theta_{next}} P(o|I_t, a) V(\tau(I_t, o, a)) \tag{4.3}$$

with truncated information states:

$$\widehat{I}_t^M = \{a_{t-M}, o_{t-M+1}, a_{t-M+1}, \cdots, a_{t-1}, o_t\}$$

that reflect only the recent M step process history.

Note that by using the truncated histories, the problem of expanding dimension that made the complete information vector (corresponds to a complete history of all actions and observations) unsuitable for the computation has been eliminated. The feature vector space based on truncated histories consists of a discrete set of recent history vectors that replace infinite information space. However, the feature vector space based on truncated histories can still be exponential in the number of history items used. For example for the POMDP model with action and observation spaces A, Θ the full M step truncated history space consists of $|A|^M |\Theta|^M$ feature vectors. When considering also cases in which history length is shorter than M the size of the feature vector space is $\frac{|A|^{M+1}|\Theta|^{M+1}-1}{|A||\Theta|-1}$ [White, Scherer 94].

Computing a value function for a feature space

Feature vector space reduces the complexity of the sufficient information state space. This leads to a loss of detail and precision as more than one sufficient information states are mapped to one feature vector. This opens the problem of how to compute the optimal value function (maximum expected reward) for an aggregate feature state. In general, one can think about defining or computing a conditional probability distribution of being in some information state given a feature vector and using this distribution to compute aggregate value function for the feature vector as a weighted average of value functions for all corresponding information vectors. However, it is often easier to choose simpler aggregation method. The obvious choice is to select a lower (upper) bound aggregate value function that assigns a value to a feature vector based on the minimum (maximum) value function value of its components.

For an M-step truncated history the choice of minimum or maximum values leads to the following upper and lower bound aggregate value functions (see [White, Scherer 94]):

$$V_L(I_t^M) = \max_{a \in A} \min_{s^*_{t-M} \in S^*} \sum_{s \in S} \rho(s,a) P(s|I_t^M, s^*_{t-M}) + \gamma \sum_{o \in \Theta_{next}} P(o|a, I_t^M, s^*_{t-M}) V_L(\tau(I_t^M, o, a))$$

$$V_U(I_t^M) = \max_{a \in A} \max_{s^*_{t-M} \in S^*} \sum_{s \in S} \rho(s,a) p(s|I_t^M, s^*_{t-M}) + \gamma \sum_{o \in \Theta_{next}} P(o|a, I_t^M, s^*_{t-M}) V_U(\tau(I_t^M, o, a));$$

where V_L and V_U stand for upper and lower bound functions, state s^*_{t-M} represents a process state at time $t-M$, that is the state just before the history information was taken. Taking the worst and best choice of a state s^*_{t-M} we get upper and lower bounds on the optimal value function. Note that whenever the specific observation and action sequence cannot be reached from s_{t-M}, that is when $P(I_t^M|s_{t-M}) = 0$, s_{t-M} should not be considered as a choice. This can happen in situations in which transitions or observation matrices contain zeros and some combinations of action-observation sequences are not possible. Thus S^* in the equations stands for a set of states that are consistent with the observed history.

Computing value function bounds for a finite horizon problem

Value function approximations based on a truncated history can be computed using dynamic programming. In order to account for all possible states, equations described above must be modified to reflect the fact that a truncated history at the beginning can be shorter than the maximum truncated length M. Assuming an n steps-to-go problem, a value function for a step $i \leq n$ is computed as:

$$V_L(I_i^k) = \max_{a \in A} \min_{s^*_{i+k} \in S^*} \sum_{s \in S} \rho(s,a) p(s|I_i^k, s^*_{i+k}) + \gamma \sum_{o \in \Theta_{next}} P(o|a, I_i^k, s^*_{i+k}) V_L(\tau(I_i^k, o, a))$$

where $k = \min(n-i, M)$.

Computing value function bounds for the infinite discounted horizon problem

Similarly to the optimal value function one can compute the value function using the value iteration method. However, the major question is whether the method converges to the unique solution for every posible initial value function. This property follows whenever the new value function mapping H_{TH} defined for truncated histories satisfies the contraction property. The contraction property of H_{TH} has been proved for example in [White, Scherer 94], and thus a value iteration method with H_{TH} converges to a unique fixed point solution. Moreover, the result also preserves the bound. Therefore one is able to use both H_{TH} mappings to compute the optimal value function bounds for the infinite discounted horizon case.

Reducing a set of possible truncated histories

The major problem with the approximation that uses an M step truncated history is that the state space size can be exponential in M. There can be $\frac{|A|^{M+1}|\Theta|^{M+1}-1}{|A||\Theta|-1}$ possible histories one needs to work with in the worst case. This causes the major slowdown whenever the truncated history length M is large.

The size of the space of truncated histories can be in many cases reduced directly by excluding suboptimal actions or impossible observations. Various tricks to eliminate such elements

from the space of histories are discussed in [Platzman 77]. Alternatively one can include in the feature space only those items from the history that are most relevant and influence the quality of the control more. Deriving autonomously which items in the history are more relevant and need to be included would help to reduce the growth of the feature space as well.

4.10.3 POMDP model reduction

An alternate approach to reduce the complexity of the information state MDP, and by this means all associated computations, is to reduce the complexity of the underlying POMDP model. This is most often done by reducing the number of process states and substituting them with aggregate process states. Note that this is slightly different from simplifing information states, although changes in the process state will show up in the information state as well.

The components of the new POMDP model can be built using state space reduction techniques similar to the model reduction techniques described in the MDP chapter. For example the transition probabilities for the new POMDP model can be computed from the original POMDP model using a new aggregate state space S^{Agg} and a conditional probability of being in some state $s \in S$ given an aggregate state $s^{Agg} \in S^{Agg}$: $P(s|s^{Agg})$. Knowing this probability distribution one can easily compute the new transition probability matrix:

$$P(s_1^{Agg}|s_2^{Agg}, a) = \sum_{s \in S} P(s|s_2^{Agg}) \sum_{s' \in s_1^{Agg}} P(s'|s, a)$$

where s' ranges over all states covered by an aggregate state s_1^{Agg}.

The major problem with this approach is related to the selection of the aggregate state space and the probability $P(s|s^{Agg})$. In the ideal case, one would like to select these such that aggregate Markov chain reflects the properties of the original chain and expected rewards associated with new aggregate states are good approximations of expected rewards defined over the original state space. The problem with this is that it would require one to aggregate together states with similar value function values. This is an open area of research, and methods that utilize a priori expert knowledge or derive appropriate aggregations autonomously can be used for this task.

The POMDP reduction method discussed above assumed that the relation between the original model and aggregate state model can be completely defined through relations between aggregate and original states. However there is always a possiblity that one can define a new (abstracted) POMDP model directly by providing all the neccessary information about its components and the relations between the original and new state spaces.

4.11 Summary

The problem of computational complexity of exact methods can be resolved by using approximation methods that trade off accuracy and precision of the solutions for speed. There are numerous methods one can use to compute approximate solutions for the POMDP policy problem. These are mostly based on value function approximations that attempt to approximate optimal value functions, using more efficient dynamic programming and value iteration updates.

method	bound	convergence
MDP-approximation	upper	yes
Blind-policy (fixed policy) method	lower	yes
Fast-informed bound	upper	yes
Curve fitting (least-squares fit)	no	no
General grid-based interpolation-extrapolation	no	no
Grid-based linear interpolation	upper	yes
Grid-based nearest neighbor	no	yes
Grid-based kernel-regression	no	yes
Grid-based incremental linear vector method	lower	yes

Table 4-1: Bound and convergence properties of value function approximation methods.

Bound and convergence properties of approximation methods

The methods and their solutions can be analyzed and compared theoretically along various properties. The two that are most important are bound, and convergence for infinite discounted horizon problems. The table 4-1 summarizes bound and convergence properties of several value function approximation methods and their solutions.

Contributions

The main contributions of our work in this chapter are:

- Summary of approximation methods for solving complex POMDP problems, analysis and proofs of their properties. Some of the proofs are based on the work of other researchers but some are new and are presented here for the first time. We have tried to present all methods in a uniform way, that is every method was described by means of an update rule it implements. This in turn makes easier their comparison with the exact and other approximate update rules.

- New fast informed bound method, that uses a simple and efficient update approximation scheme and upper bounds the exact update rule. The rule approximates value function using at most $|A|$ linear vectors.

- Blind policy method that uses a set of blind policies to compute components of the piecewise linear lower bound of the optimal value function.

- New grid-based point interpolation rule that supports arbitrary (variable) grids, and thus arbitrary grid selection strategies. This is unlike regular grid methods that evenly partition the belief space and use fixed sets of grid points

- New heuristic approach for constructing point interpolation grids. The method uses stochastic simulations and attempts to improve the value function value for critical belief points. The method can be combined also with other grid-based interpolation-extrapolation strategies, for example nearest neighbor.

- New incremental linear vector method for infinite discounted horizon problems that is based on Sondik's linear vector updates. The method computes and incrementally improves a piecewise linear and convex lower bound of the optimal value function over

iterations steps. The method can use arbitrary set of grid points (including heuristic ones) and is also a basis of the Gauss-Seidel speedup technique for exact value iteration.

Chapter 5

Value function approximation methods: an experimental study

The objective of this chapter is to empirically compare the performance of several value function approximation methods and their solutions on complex POMDP control problems. It naturally complements the previous chapter, which was more formal and focused on the description of approximation methods, their properties, and relations. In the following, approximation methods will be evaluated using two criteria:

- the quality of value function bounds;
- their control performance.

In the first part of the experiment, approximation methods that provide upper and lower bounds on the optimal value functions will be tested. In the second part, several value function approximation methods will be compared directly on the control task and will be judged solely based on their control performance.

The role of empirical research in scientific exploration is enormous. It helps us confirm or refute our expectations and guides our exploration of the field by giving us a better understanding of features that had not been shown theoretically. Unfortunately the area of POMDP approximations lacks large scale experimental work. Thus our primary mission is to take a small step in this direction and provide a comparison of methods and their extensions. We will compare both new and known approximation methods, including simple approximations based on perfect observability, the curve fitting approach based on least-squares fit and more sophisticated heuristic grid-based methods.

In the following, we will first describe a set of three control problems we used in the experiments. After that, upper and lower bound value function approximations will be compared. Finally the control performance of various approximations will be examined and analysed.

5.1 Test problems

We tested value function solutions using a set of three infinite discounted horizon POMDP problems of different complexity. The problems tested are:

- The Maze20 maze navigation problem [Hauskrecht 97b];

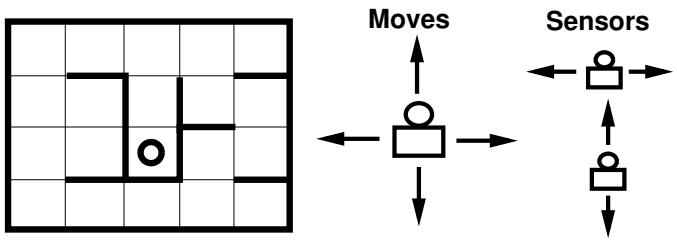


Figure 5-1: The robot navigation problem: Maze20

- The Maze20B maze navigation problem with a zero cost absorbing state
- The Shuttle docking problem [Chrisman 92]

The Maze20 navigation problem was designed to provide the hardest problem, with investigative actions being of high importance for the optimization of the objective function. This was achieved by providing a highly structured and narrow maze, with many obstacles that could lead to a high score loss. The Maze20B problem uses a slightly less structured maze, and cost-reward model that penalizes blind maneuvering less compared to the Maze20 problem. Thus, it is less dependent on investigative actions. Shuttle docking is a problem with low uncertainty in both transition and observation models. This significantly reduces the impact of partial observability on the problem and solution. In the following we only give a brief description of each problem. All three problems are described fully in appendix A (they can be also downloaded on-line at: "http://www.medg.lcs.mit.edu/people/milos/thesis/").

Test problem 1: Maze20

Maze20 [Hauskrecht 97b] is a maze navigation problem with 20 states, 6 actions and 8 observations. The maze (figure 5-1) consists of 20 partially connected rooms (states) in which a robot functions and collects rewards. The robot can move in 4 directions (North, South, East and West) and can check for the presence of walls using its sensors. Neither "move" actions nor sensor inputs are perfect and the robot can wind up moving in unintended directions. The robot moves in an unintended direction with probability of 0.3 (0.15 for each of the neighboring directions). A move into the wall keeps the robot in the same position. Investigative actions help the robot to navigate by activating sensor inputs. There are 2 investigative actions that allow the robot to check inputs (presence of a wall) in the North-South and East-West directions. Sensor accuracy in detecting walls is 0.75 for a two wall case (e.g. both north and south wall), 0.8 for a one wall case (north or south) and 0.89 for a no wall case, with smaller probabilities for wrong perceptions.

The control objective is to maximize the expected discounted rewards with a discount factor of 0.9. A small reward is given for every action not leading to bumping into the wall (4 points for a move and 2 points for an investigative action), and one big reward (150 points) is given for achieving the special target room (shown as a circle on the figure) and recognizing it by performing one of the move actions. After doing that and collecting the reward, the robot is placed with some probability into one of the 'initial" rooms.

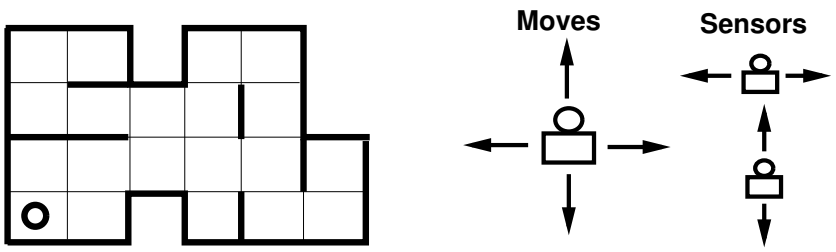


Figure 5-2: The robot navigation problem: Maze20B

Test problem 2: Maze20B

The Maze20B problem (see figure 5-2) is similar to Maze20. The two problems use different maze topologies. However, the uncertainty associated with the outcomes of "move" actions and the quality of perceptual information is the same as for Maze20.

The other major difference between the two maze problems is in the payoff model: Maze20B uses costs instead of rewards. Costs are assigned in the following way: 20 points for every action that does not cause the robot to crash into the wall, 30 points for any action (move) that bumps the robot against the wall and 0 points for any action in the goal state (represented by a circle). Note that the costs of various actions and their outcomes favor more move actions compared to the Maze20 problem.

The goal state is a zero cost absorbing state (sink). The objective is to optimize control for the infinite discounted horizon, with a discount factor of 0.95.

Test problem 3: Shuttle docking

The Shuttle docking problem [Chrisman 92] consists of 8 states, 3 actions and 5 observations. The states consists of the position of the shuttle relating to the most and least recently visited space station. The objective is to continuously move and dock the shuttle at the least recently visited space station, which is rewarded with 10 points. The discount factor used is 0.95.

The major difference between the Shuttle problem and the maze problems is that it does not have investigative actions. The observations used are considered to be free (no cost) and always available. Also uncertainties associated with either transitions or observations are not as bad as in the case of the maze problems, and in many cases, the relations are deterministic. The Shuttle problem has features that make a control problem easier to solve (small amount of partial observability, no investigative actions, a lot of determinism in action outcomes).

5.2 Comparing quality of bounds

5.2.1 Methods tested

We tested the bounds on value functions produced by several methods that were discussed in the previous chapter and that were proved to have upper or lower bound properties. However, we note that there are other methods one can use to compute upper or lower bounds that we did not test, for example model reduction methods based on truncated histories discussed in

section 4.10.2.

Upper bound methods

We tested and compared the following upper bound methods:

- MDP-based approximation;
- Fast informed bound method;
- Grid-based point interpolation with regular, random and heuristic grids.

The MDP based approximation is a basic method for computing an upper bound. The solution it produces consists of a single linear vector, and is often used to initialize other, more complex upper bound methods. Thus, the quality of the MDP-based bound will provide the score against which the improvements of other methods can be measured and compared.

The fast informed bound method improves the MDP-based bound using a piecewise linear and convex value function that consists of $|A|$ linear vectors.

The MDP-based bound can be improved futher by using the grid-based point interpolation method. Grid-based point interpolation can be implemented using different types of grids including regular, random and heuristic grids. We have tried and tested all three types of grids. Regular grids were combined with the efficient point interpolation strategy due to [Lovejoy 91b] that always interpolates a target point using the grid-points that are closest to it. In addition, both random and heuristic grids were implemented with a new point interpolation method described in section 4.7.3. The heuristic approach implemented a new strategy proposed in 4.7.3. Different types of grids have been tried for different grid resolutions. We used grids of 40 points up to 440 grid points (in 40 point increments). The heuristic grids for larger resolutions were constructed incrementally using previous step solutions. The regular grid method was tested only on regular grids that fell in the tested range. These included a grid of 210 points for both maze problems, and grids of 36, 120, and 330 points for the smaller Shuttle problem.

Lower bound methods

We tested the following lower bound methods:

- Simple blind policy method;
- Incremental linear vector method with various point selection strategies.

The simple blind policy method (section 4.5.3) computes a piecewise linear and convex value function that consists of $|A|$ linear vectors, one for every blind one-action policy. The solution lower bounds the optimal value function and can be used to initialize incremental linear vector methods.

The incremental linear vector method (section 4.8.2) is designed to gradually improve a piecewise linear and convex lower bound. It can be combined with various strategies for selecting points for updates. We tested four different point selection strategies. These were evaluated using 40 point update cycles for up to 440 point updates. The strategies we compared are:

- A fixed grid strategy with a fixed set of 40 belief points that are used repeatedly. The grid points consist of all critical belief points, and the remaining points are selected randomly.
- A random grid strategy that selects every belief point to be updated randomly;

- An order heuristic strategy (see section 4.8.2) that repeatedly picks 40 belief points, including all critical points. The critical points are ordered to maximize the update effect.
- A two tier heuristic strategy (see section 4.8.2) that combines the heuristic ordering strategy with a forward simulation strategy. Every critical point (ordered) is simulated forward for 5 steps, and a sequence of points obtained is updated in reverse order.

5.2.2 Experimental design

Value function solutions are defined over the continuous belief space. This makes it impossible to compare bound results for every possible belief state. In order to compare the quality of bounds obtained by different methods we use a single score that measures the average value obtained for a fixed set of 2500 randomly generated belief points together with all of the critical points of the belief simplex.

5.2.3 Test results

The result scores achieved for both bounds are listed in: figures 5-3 and 5-4 for the Maze20 problem; figures 5-5 and 5-6 for the Maze20B; and 5-7 and 5-8 for the Shuttle docking problem. Note that the Maze20B problem minimizes costs, and therefore the upper and lower bound methods are exchanged compared to the problems that maximize rewards.

5.2.4 Evaluation of results

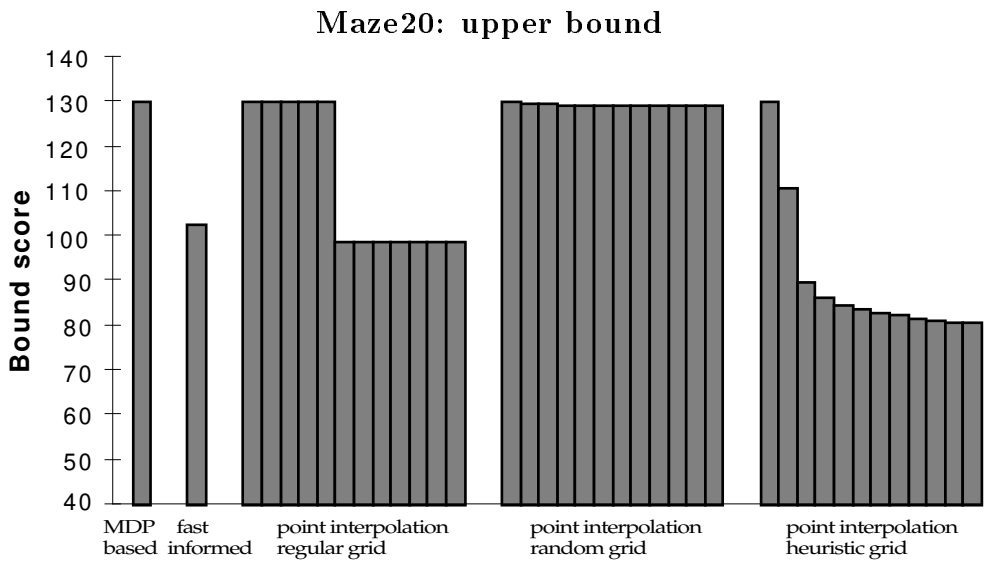
Upper bound

The worst results were achieved by the grid-based point interpolation method with random grids. This is mostly because transitions in all models are local and sparse. This means that from any critical point one can only get to belief states that lie on the boundary of the belief simplex, that is, those belief points that contain a lot of zeros. In contrast to this, random sampling is more likely to produce a belief point with nonzero probabilities. Since any boundary point can be interpolated using only points on the same boundary, the internal points of the belief simplex have no effect on their interpolation, and thus there is a very slim chance that critical points will get updated by randomly generated grids.

Regular grids with small resolution have a significantly better bound score because they consist only of points on the belief simplex boundaries.

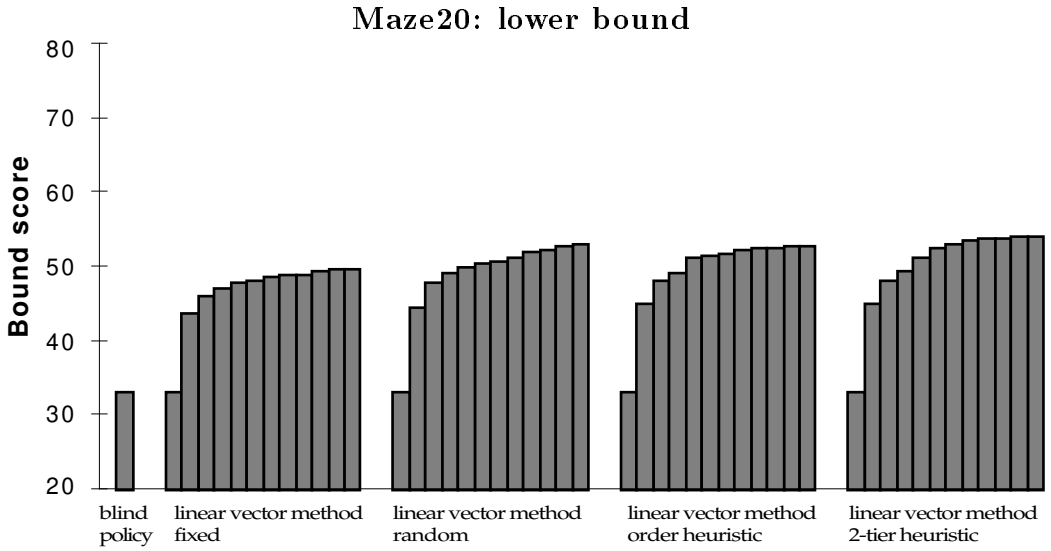
Overall, the best results were achieved by the heuristic grid method with forward point simulations. The method was significantly better than random and regular grids for both Maze20 problems, and was beaten by a low margin by a regular grid method only on the Shuttle problem. We believe that the main reason for this is that the heuristic grid method uses a simple point interpolation rule that does not search for the best interpolating set, while the regular grid method uses a minimum distance point interpolation rule.

The other contender – the newly designed fast informed bound method performed very well on all test problems and was able to beat the grid based methods with lower grid resolutions. The main advantage of the method is that it is easy and fast to compute, thus it is able to give us a good upper bound in relatively short time.



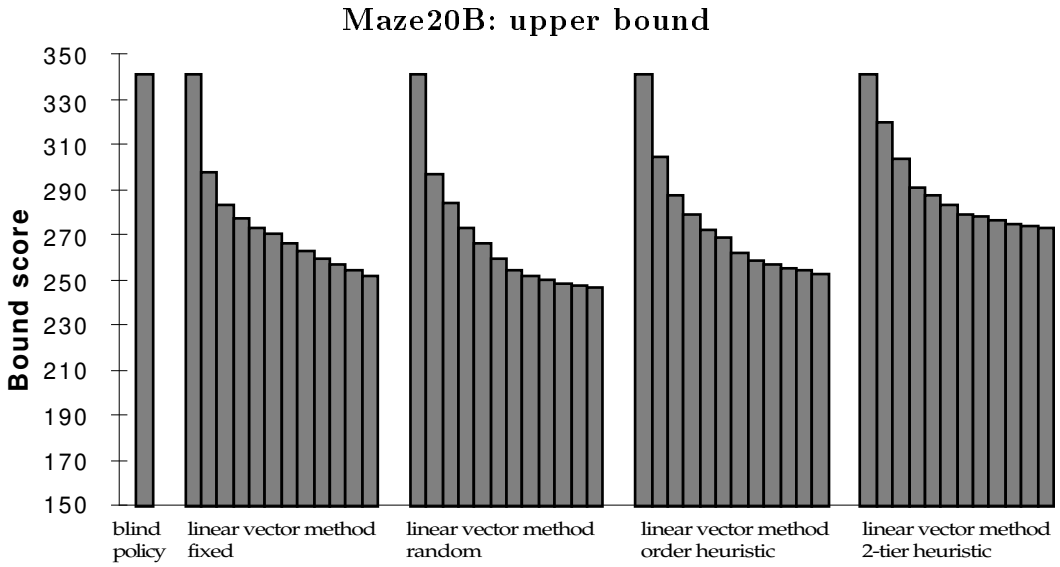
grid size	MDP-based approximation	fast informed bound method	regular grids	random grids	heuristic grids
initial	130.11	102.48	130.11	130.11	130.11
40	-	-	-	129.72	110.92
80	-	-	-	129.55	89.59
120	-	-	-	129.44	86.38
160	-	-	-	129.35	84.49
200	-	-	98.91*	129.28	83.84
240	-	-	-	129.25	83.06
280	-	-	-	129.24	82.32
320	-	-	-	129.19	81.84
360	-	-	-	129.17	81.35
400	-	-	-	129.16	80.98
440	-	-	-	129.14	80.70

Figure 5-3: Maze20: quality of upper bounds. The table and graph show bound scores obtained by an MDP-based approximation, the fast informed bound method, and point interpolation methods with three types of grids: regular, random and heuristic. The grid-based point interpolation methods were tested using different resolutions (grid sizes) starting from MDP-based approximations. The sequence of possible regular grids is sparse and the only grid (excluding the initial one) that was within the tested range used 210 grid points (score is labeled with an asterisk in the table). The other grid resolutions did not work and therefore they were not able to improve the bound score.



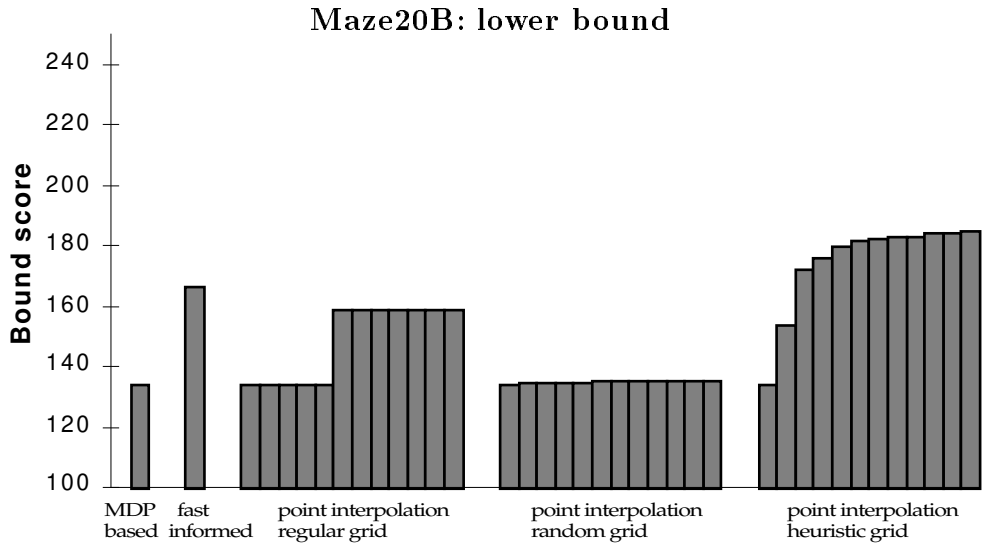
number of updates	point selection strategy			
	fixed	random	order heuristic	2-tier heuristic
initial (blind policy)	33.10	33.10	33.10	33.10
40	43.80	44.65	44.96	45.16
80	46.20	47.93	48.27	48.17
120	47.10	49.28	49.18	49.60
160	47.89	50.12	51.24	51.19
200	48.23	50.43	51.56	52.64
240	48.60	50.70	51.79	53.04
280	48.89	51.18	52.29	53.58
320	49.06	52.02	52.46	53.88
360	49.38	52.40	52.65	53.96
400	49.67	52.84	52.76	54.13
440	49.86	53.09	52.89	54.17

Figure 5-4: Maze20: quality of lower bounds. The table and graph show bound scores obtained by the incremental linear vector method and four different point selection strategies. They were tested and compared after every 40 point updates. The initial value function was obtained using a simple blind policy method that combines all one action policies.



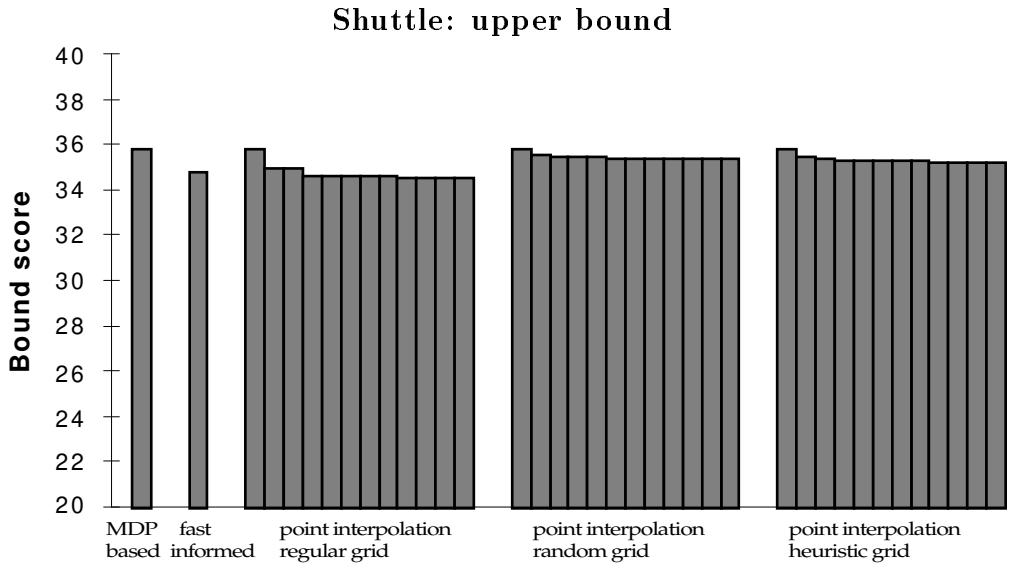
number of updates	point selection strategy			
	fixed	random	order heuristic	2-tier heuristic
initial (blind policy)	341.40	341.40	341.40	341.40
40	298.25	297.41	304.83	320.04
80	283.91	284.29	287.77	303.86
120	277.64	273.1	279.26	291.37
160	273.29	266.85	272.41	288.22
200	271.20	259.46	269.15	283.86
240	266.31	254.58	262.09	279.00
280	262.81	252.39	258.69	278.19
320	259.79	250.43	257.01	276.72
360	257.49	248.93	255.71	274.78
400	254.27	247.91	254.76	274.29
440	252.22	247.03	252.66	273.43

Figure 5-5: Maze20B (cost minimization): quality of upper bounds. The table and graph show bound scores obtained by the incremental linear vector method and four different point selection strategies. They were tested and compared after every 40 point updates. The initial value function was obtained using a simple blind policy method that combines all one action policies.



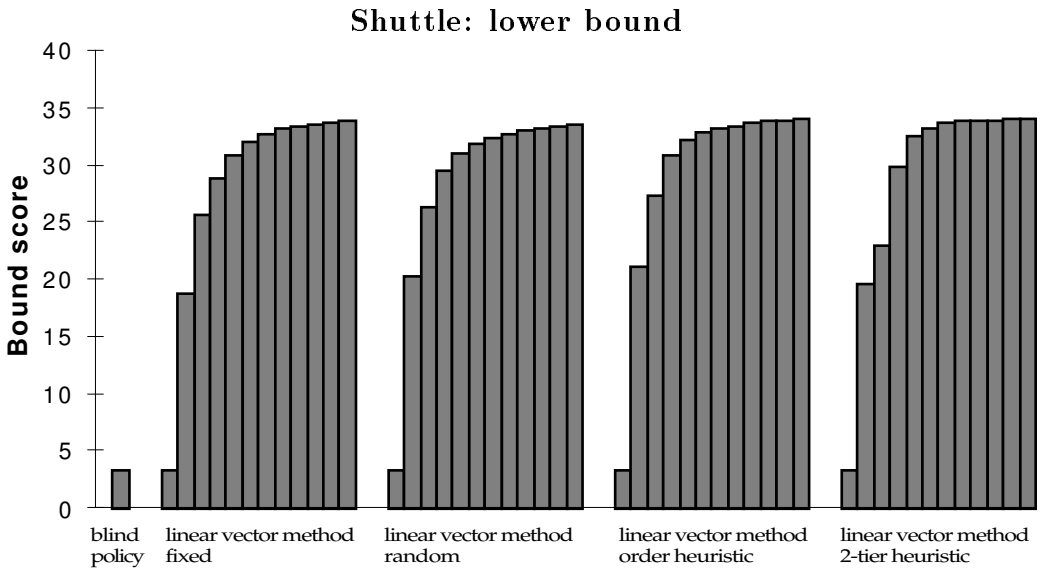
grid size	MDP approximation	fast informed bound method	regular grids	random grids	heuristic grids
initial	134.36	166.95	134.36	134.36	134.36
40	-	-	-	134.81	154.04
80	-	-	-	134.92	172.66
120	-	-	-	135.04	176.25
160	-	-	-	135.17	180.11
200	-	-	159.17*	135.30	181.68
240	-	-	-	135.32	182.60
280	-	-	-	135.37	183.30
320	-	-	-	135.42	183.51
360	-	-	-	135.46	184.26
400	-	-	-	135.49	184.67
440	-	-	-	135.53	185.00

Figure 5-6: Maze20B (cost minimization): quality of lower bounds. The table and graph show bound scores obtained by an MDP-based approximation, the fast informed bound method, and point interpolation methods with three types of grids: regular, random, and heuristic. The grid-based point interpolation methods were tested using different resolutions (grid sizes) starting from MDP-based approximations. The only regular grid that was within the tested range used 210 grid points (the score is labeled with an asterisk in the table). The other grid resolutions did not work and therefore they were not able to improve the bound score.



grid size	MDP approximation	fast informed bound method	regular grids	random grids	heuristic grids
initial	35.88	34.80	35.88	35.88	35.88
40	-	-	35.01*	35.62	35.49
80	-	-	-	35.55	35.43
120	-	-	34.66*	35.52	35.38
160	-	-	-	35.49	35.35
200	-	-	-	35.46	35.33
240	-	-	-	35.44	35.31
280	-	-	-	35.43	35.30
320	-	-	34.56*	35.42	35.29
360	-	-	-	35.41	35.28
400	-	-	-	35.41	35.27
440	-	-	-	35.40	35.27

Figure 5-7: Shuttle docking problem: quality of upper bounds. The table and graph show results obtained by an MDP-based approximation, the fast informed bound method, and point interpolation methods with three types of grids: regular, random and heuristic. The grid-based point interpolation methods were tested using different resolutions (grid sizes) starting from MDP-based approximations. The regular grids that were within the tested range used 36, 120, and 330 grid points (their scores are labeled with asterisks). The other grid resolutions did not work and therefore they were not able to improve the bound score.



number of updates	point selection strategy			
	fixed	random	order heuristic	2-tier heuristic
initial (blind policy)	3.40	3.40	3.40	3.40
40	18.78	20.40	21.13	19.67
80	25.69	26.37	27.43	23.01
120	28.93	29.54	30.89	29.86
160	30.98	31.04	32.31	32.55
200	32.14	31.86	32.97	33.27
240	32.81	32.43	33.32	33.75
280	33.20	32.85	33.49	33.90
320	33.46	33.08	33.75	33.99
360	33.66	33.21	33.91	34.03
400	33.83	33.47	34.01	34.05
440	33.95	33.64	34.08	34.06

Figure 5-8: Shuttle docking problem: quality of lower bounds. The table and graph show bound scores obtained by the incremental linear vector method and four different point selection strategies. They were tested and compared after every 40 point updates. The initial value function was obtained using a simple blind policy method that combines all one action policies.

Lower bound

The experiments showed that there is no clear winning point selection strategy for the incremental linear vector method. The differences among the strategies were very small. Also, on different problems, different strategies performed best. A surprise was a relatively bad showing of the two-tier heuristic strategy on the Maze20B problem (minimization) where the quality of its solutions fell behind all other methods. We believe that the reason for this is that the strategy sampled the same set of belief points repeatedly with small potential for improvements. Thus, the method would probably be better if it switched and interleaved heuristic selections with some random selection strategy.

In general, the reason for small differences in the lower bound quality could be explained by updating linear vectors (derivatives) for belief points. In such a case the new linear vector influences a larger portion of the belief space and thus it is less sensitive to a specific point selection strategy. Also possible is the explanation that we did not use a very good heuristic, and better heuristics or their combinations can be constructed.

Bound results summary

Both upper and lower bound incremental methods were combined with various heuristic methods for locating new grid points. The heuristic grid approach for the grid-based point interpolation based on forward simulations seems to be justified and was able to outperform significantly both random and regular grids most of the time. The results suggest that the point-interpolation method is sensitive to a selection of grid points. Interestingly, we were not able to get any significant improvement from any of the heuristic approaches for the incremental lower bound method. Moreover both random point selection and fixed random strategies are producing similar results. This suggest that the incremental linear vector is less sensitive to the selection of the grid points used for updates.

Overall, upper and lower incremental bound methods (heuristic grid assumed for the point-interpolation bound) were able to improve significantly on the initial bounds provided by the MDP-based method and the simple blind policy methods. However, despite this, the combination of upper and lower bound methods did not achieve very tight bound spans for the tested range, except on the Shuttle docking problem with 8 states, 3 actions and 5 observations. We believe we did not get very tight bounds on maze problems because one needs to use more grid-points or updates for more complex problems (with larger state and observation spaces) in order to get closer to the optimal solution. The tested ranges of grid sizes and updates were simply not sufficient for the two more complex problems.

5.3 Testing control performance of approximation methods

There are many approximation methods that have been proposed to compute POMDP control efficiently (see [Lovejoy 91b] [Littman et al. 95a] [Parr, Russell 95]). However, the comparisons of these methods were either insufficient or did not include problems of larger complexity. For example, comparison studies that appeared in the AI literature have focused mostly on the application of least-squares fit strategies, and have not tried grid based approaches even though they are common in operations research. Therefore a primary focus of our work in this is section is to compare a spectrum of value function approximation approaches and their solutions on the set of infinite discounted horizon problems.

5.3.1 Methods tested

We tested all of the value function approximation methods described in the previous chapter: MDP-based approximation, the blind policy method, the fast informed bound method, the least-square fit approach, linear interpolation-extrapolation rules and incremental lower bound method with Sondik's updates. Some methods were represented by multiple entries because they used different grids or different heuristics. The purpose of this variation was to show how specific tricks or heuristics influence the approximation.

Optimal solutions

The results obtained for different approximation methods were compared to the results one would achieve with the optimal solution for the perfectly observable Markov process (where process states are assumed to be perfectly observable), and for the Shuttle docking problem, they were also compared to results for the optimal POMDP solution with 10^{-5} precision[1]. The optimal solution for both maze problems was too hard to compute to any reasonable precision due to the huge increase in the size of the linear vector set. The objective of the comparison of the perfectly and partially observable cases was to provide some idea about how hard the control task under imperfect observability really is. Note that in the perfectly observable case the investigative actions usually become suboptimal.

Methods with multiple entries

Despite the threat of instability we computed and tested value functions obtained by value iteration with a least-squares fit. We tried two function models: a linear Q-function model [Littman et al. 95a] and a softmax model [Parr, Russell 95]. They were described in section 4.6.3. Q-functions were updated in parallel for a fixed set of 100 points that included all the critical points of the belief simplex. The least-squares fits were computed at every step using gradient parameter learning techniques. The initial set of Q-functions was based on solutions acquired for the corresponding blind one-action policies. The least-squares function was tested after 10, 20 and 30 iterations. Softmax function model was only used on the Maze20 problem. Solutions with 10 and 15 linear vectors were acquired after 10 and 20 iteration steps. The functions were updated in parallel for a fixed set of 50 and 100 points respectively that included all critical points of the belief simplex. In both cases models were initialized with the solution acquired by the simple blind policy method.

Grid-based interpolation-extrapolation methods were tested using nearest-neighbor and point interpolation rules, for grid sizes of 40, 200 and 400 belief points. The interpolation rule has been implemented by a simple interpolation method proposed in section 4.7.3 that fits convex and piecewise linear value functions with a "saw"-shaped function. Both nearest-neighbor and point interpolation were tried on both random and heuristic grids. Heuristic grids were generated using model-based sampling as described in the previous chapter (section 4.7.3). The point interpolation method was also tested on regular grids using the efficient interpolation strategy proposed in [Lovejoy 91b].

The control performance of the incremental linear vector method with Sondik's updates was tested for solutions acquired after 40, 200, and 400 point updates. We used the same strategies to select the belief points that were used for the bound experiments (section 5.2).

[1] The solution for the Shuttle docking problem with the 10^{-5} precision was kindly provided by Anthony Cassandra. It consists of 208 linear vectors.

5.3.2 Experimental design

The quality of each method's performance was tested using simulations for different sets of initial belief points. The simulation runs for each initial belief state were 60 steps long. In each run, the actual discounted reward or cost obtained by a control agent powered with a specific value function approximation were collected. This gave us an approximation of the discounted score the agent would achieve if it were to run forever. Beyond the overall reward (cost) score, the other statistics were collected, such as the number of times a goal state was reached and the number of observations in the run. Methods and their solutions were tested on two test sets that consisted of:

- 2000 randomly generated belief points (arbitrary points);
- 1500 randomly generated critical belief points.

The Shuttle docking problem has not been tested on a set of random belief points. The reason for this is that in the Shuttle problem, observations are very good indicators of the underlying state and thus it is always possible to exclude the majority of underlying process states (only belief states with few nonzero states are possible). Therefore a test on the set with randomly generated belief points does not make much sense.

5.3.3 Test results

Simulation results obtained for various methods and test sets are presented both in tables and graphically (using bar diagrams) in the following way :

- Maze20 in tables 5-1, 5-2 and figures 5-9, 5-10;
- Maze20B in tables 5-3, 5-4 and figures 5-11, 5-12;
- Shuttle in table 5-5 and figure 5-12.

The simulation results listed in the tables include: the average of discounted rewards (costs) achieved for all simulation runs (achieved score), the percentage of times the "goal" state was reached in 60 steps, the average of expected discounted rewards (costs) predicted by an approximate value function for all simulation runs (expected score), and the average number of investigative actions per simulation run (60 steps). The achieved score (average reward) is the primary criterion to evaluate the performance of the method. The other statistics are informative and tend to reveal more about the nature and the behavior of the methods.

Testing methods differences

The overall achieved score (average reward or average cost score) for a given test set quantifies the quality of control. However the average score itself does not tell us if two methods with different average scores are also statistically significantly different. The reason for this is that two methods can produce different average scores simply as a result of some underlying random process. Thus to validate that the scores obtained are not the result of randomness we need to show that the methods are in fact significantly different. We do this by comparing not only their average performance, but by comparing their performance on many individual simulation runs.

All methods were run and tested on the same set of belief points. We also assured that the simulator was always initialized from the same process state. This means that sample rewards

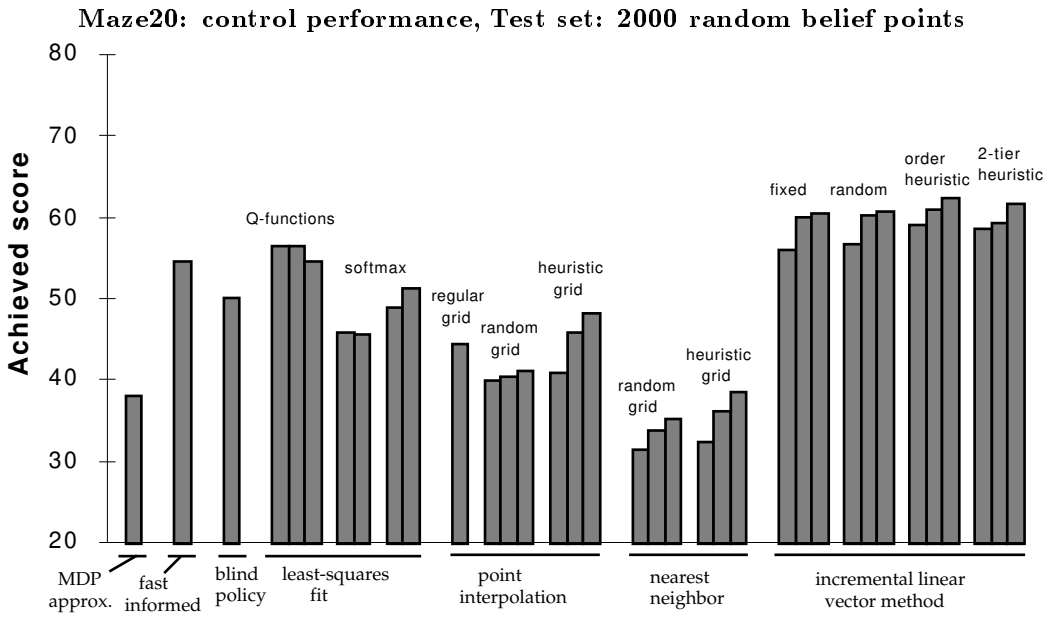


Figure 5-9: Maze20 comparison of control performance. The achieved score represents the average reward on the set of 2000 random belief points.

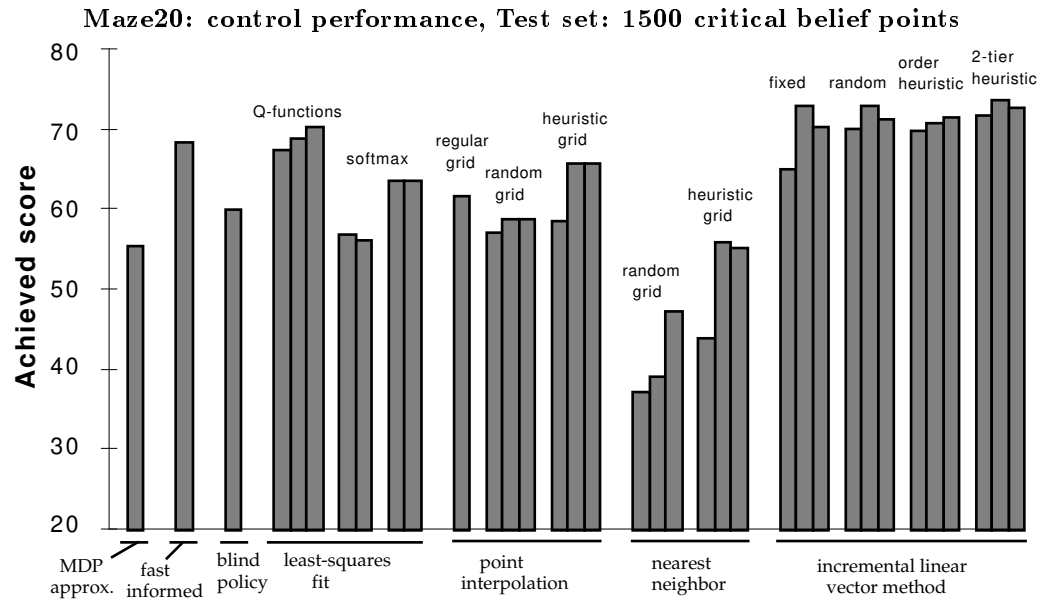


Figure 5-10: Maze20 comparison of control performance. The achieved score represents the average reward on the set of 1500 randomly selected critical belief points.

Maze20: control performance, Test set: 2000 random belief points

method	method parameters	achieved score	percent of goal	expected score	average observ.
MDP (observable)	-	131.16	100	129.76	0
MDP-approximation	-	38.24	34.25	129.75	0.01
Fast informed bound	-	54.73	80.20	102.05	10.54
Simple blind policy	-	50.18	42.75	32.60	10.28
least square fit with Q-functions	points: 100, iter: 10	56.70	78.40	54.13	10.91
	points: 100, iter: 20	56.59	79.30	61.54	10.80
	points: 100, iter: 30	54.73	78.50	64.14	10.97
least square fit method with softmax function	vectors:10, points:50, iter:10	45.94	62.45	48.32	37.83
	vectors:10, points:50, iter:20	45.68	55.55	50.81	39.34
	vectors:15, points:100, iter:10	49.10	67.90	60.84	34.68
	vectors:15, points:100, iter:20	51.43	72.15	58.74	32.95
Grid based point interpolation method	regular grid (210 points)	44.60	39.35	98.47	0.00
	random grid (40 points)	40.08	62.75	129.35	7.24
	random grid (200 points)	40.45	67.15	128.91	10.39
	random grid (400 points)	41.37	69.40	128.78	9.70
	heuristic grid (40 points)	41.13	38.25	110.57	0.27
	heuristic grid (200 points)	46.08	67.95	83.48	19.18
	heuristic grid (400 points)	48.37	70.60	80.61	19.49
Grid based nearest neighbor method	random grid (40 points)	31.49	16	294.69	33.70
	random grid (200 points)	33.87	17.10	176.73	33.37
	random grid (400 points)	35.38	17.45	161.56	37.93
	heuristic grid (40 points)	32.43	33.70	114.58	30.36
	heuristic grid (200 points)	36.33	22.50	129.10	17.54
	heuristic grid (400 points)	38.75	29.70	96.43	22.19
Incremental linear vector method	fixed (40 updates)	56.16	62.95	43.19	16.87
	fixed (200 updates)	60.18	90.35	47.68	24.05
	fixed (400 updates)	60.62	89.50	49.08	24.27
	random (40 updates)	56.82	75.95	43.96	18.26
	random (200 updates)	60.45	86.75	49.88	21.51
	random (400 updates)	60.98	88	52.31	21.81
	order heuristic (40 updates)	59.18	86.35	44.39	23.36
	order heuristic (200 updates)	61.01	90.40	51.03	23.22
	order heuristic (400 updates)	62.41	90.30	52.25	23.17
	2-tier heuristic (40 updates)	58.63	86.65	44.52	23.79
	2-tier heuristic (200 updates)	59.47	90.35	52.07	21.56
	2-tier heuristic (400 updates)	61.72	88.05	53.55	23.37

Table 5-1: Simulation results for the Maze20 problem and 2000 random belief points. The table includes results for the perfectly observable MDP control (for the purpose of comparison), MDP-based approximation, fast informed bound method, simple blind policy, least square fit method with Q-function and softmax functions - tested for different number of iteration steps (10, 20, 30) and different numbers of sample points (softmax also for 10 or 15 linear vectors), grid-based point interpolation strategy with regular, random and heuristic grids (for various grid sizes), grid-based nearest neighbor with random and heuristic grid (for various grid sizes), and incremental linear vector method for fixed random, dynamic random, order heuristic and two-tier heuristic point selection strategies (for different number of updates).

Maze20: control performance, Test set: 1500 critical belief points

method	method parameters	achieved score	percent of goal	expected score	average observ.
MDP (observable)	-	130.21	100	129.83	0
MDP-approximation	-	55.51	44.67	129.81	0.11
Fast informed bound	-	68.46	82.80	107.70	9.73
Simple blind policy	-	60.16	47.47	38.72	8.72
least square fit with Q-functions	points: 100, iter: 10	67.61	80.33	59.83	9.91
	points: 100, iter: 20	68.94	81.40	67.22	9.90
	points: 100, iter: 30	70.43	82.33	69.87	10.14
least square fit method with softmax function	vectors:10, points:50, iter:10	57.05	71.47	61.81	36.06
	vectors:10, points:50, iter:20	56.36	62.73	64.66	37.32
	vectors:15, points:100, iter:10	63.73	81.67	73.81	29.81
	vectors:15, points:100, iter:20	63.77	78.93	71.63	30.05
Grid based point interpolation method	regular grid (210 points)	61.76	52.60	102.30	0.06
	random grid (40 points)	57.18	67	129.81	6.47
	random grid (200 points)	58.86	72.80	129.81	9.69
	random grid (400 points)	58.92	72.73	129.81	9.33
	heuristic grid (40 points)	58.76	51.20	111.13	0.28
	heuristic grid (200 points)	65.83	77.73	85.05	17.43
	heuristic grid (400 points)	65.80	77.20	82.36	17.29
Grid based nearest neighbor method	random grid (40 points)	37.22	14.40	286.94	39.29
	random grid (200 points)	39.29	17.67	175.51	29.80
	random grid (400 points)	47.33	26.13	157.28	38.74
	heuristic grid (40 points)	44.00	49.27	124.22	30.03
	heuristic grid (200 points)	55.89	40.93	123.07	20.69
	heuristic grid (400 points)	55.28	42.47	99.44	17.67
Incremental linear vector method	fixed (40 updates)	65.12	62.80	52.22	14.99
	fixed (200 updates)	73.10	93.40	62.52	22.63
	fixed (400 updates)	70.37	92.40	64.02	22.98
	random (40 updates)	70.08	76.67	52.94	16.20
	random (200 updates)	73.08	90.33	62.03	20.19
	random (400 updates)	71.33	90	64.81	20.35
	order heuristic (40 updates)	69.81	88.53	57.73	22.08
	order heuristic (200 updates)	70.96	91.67	65.31	21.91
	order heuristic (400 updates)	71.62	90.73	66.82	21.84
	2-tier heuristic (40 updates)	71.79	90.13	58.70	22.31
	2-tier heuristic (200 updates)	73.82	92.33	69.04	19.92
	2-tier heuristic (400 updates)	72.90	91.20	71.17	22.02

Table 5-2: Simulation results for the Maze20 problem and the set of 1500 critical belief points. The table includes results for the perfectly observable MDP control (for the purpose of comparison), MDP-based approximation, fast informed bound method, simple blind policy, least square fit method with Q-function and softmax functions - tested for different number of iteration steps (10, 20, 30) and different numbers of sample points (softmax also for 10 or 15 linear vectors), grid-based point interpolation strategy with regular, random and heuristic grids (for various grid sizes), grid-based nearest neighbor with random and heuristic grid (for various grid sizes), and incremental linear vector method for fixed random, dynamic random, order heuristic and two-tier heuristic point selection strategies (for different number of updates).

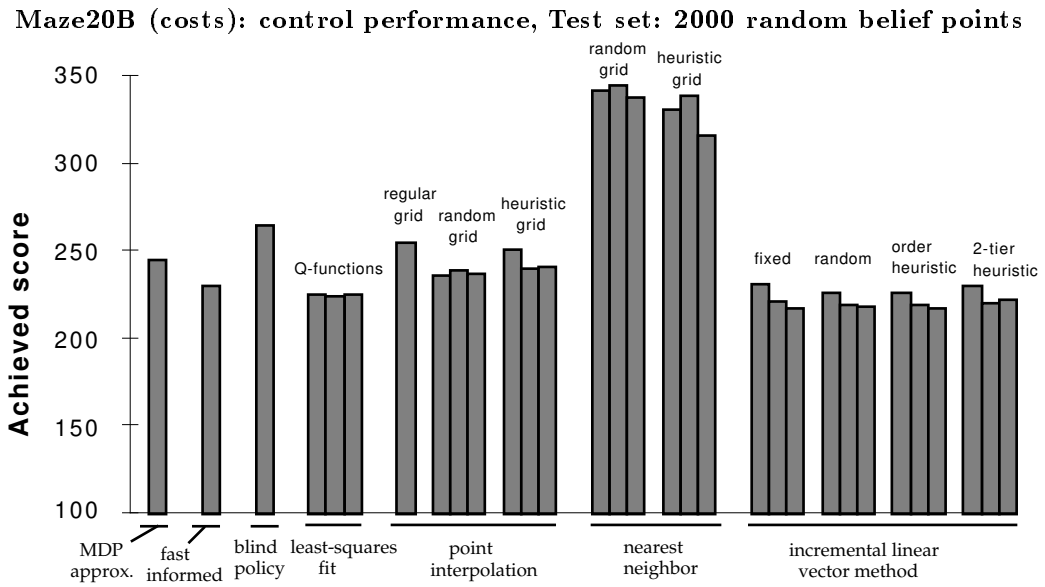


Figure 5-11: Maze20B: comparison of control performance. The achieved score represents the average cost on the set of 2000 random belief points. Lower scores correspond to better performance on this problem.

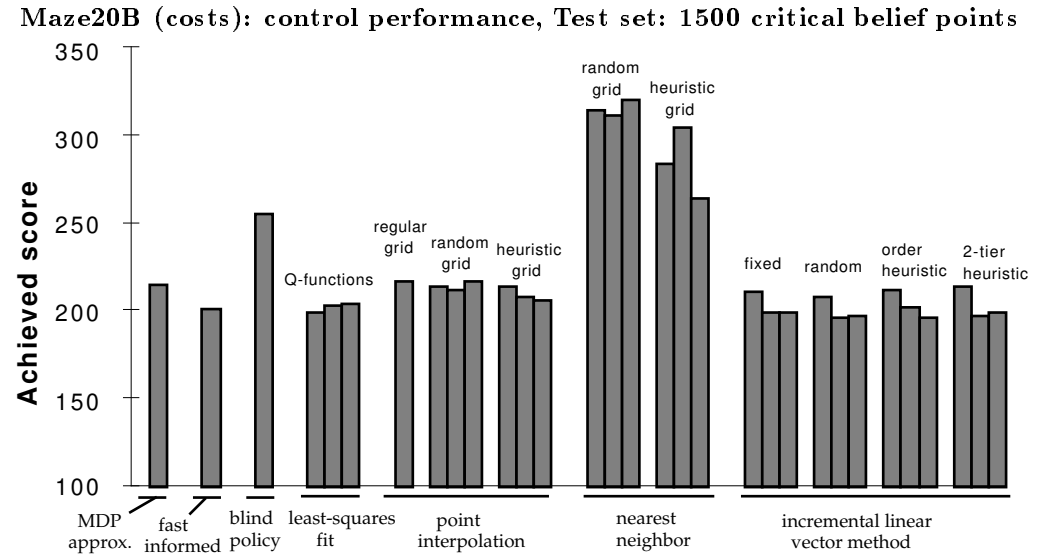


Figure 5-12: Maze20B: comparison of control performance. The achieved score represents the average reward on the set of 1500 random critical belief points. Lower scores are better for this problem.

Maze20B (costs): control performance, Test set: 2000 random belief points

method	method parameters	achieved score	percent of goal	expected score	average observ.
MDP (observable)	-	133.37	100	134.23	0
MDP-approximation	-	245.06	90.40	134.23	6.28
Fast informed bound	-	230.48	91.25	166.80	7.36
Simple blind policy	-	264.74	64.10	341.50	20.26
least square fit with Q-functions	points: 100, iter: 10	225.08	94.90	262.13	35.78
	points: 100, iter: 20	224.47	95.25	241.48	35.98
	points: 100, iter: 30	225.43	95.30	238.52	35.80
Grid based point interpolation method	regular grid (210 points)	254.68	70.75	159.06	12.38
	random grid (40 points)	236.63	92.35	134.67	5.70
	random grid (200 points)	239.75	90.55	135.18	6.53
	random grid (400 points)	236.94	92.25	135.38	6.34
	heuristic grid (40 points)	250.88	78.30	153.90	12.83
	heuristic grid (200 points)	240.74	82.35	181.55	12.89
	heuristic grid (400 points)	241.33	80.95	184.56	13.37
Grid based nearest neighbor method	random grid (40 points)	342.41	17.55	156.59	45.25
	random grid (200 points)	345.23	26.50	175.10	31.12
	random grid (400 points)	338.30	28.20	164.11	30.14
	heuristic grid (40 points)	331.55	42.35	107.85	12.53
	heuristic grid (200 points)	339.35	35.65	116.19	15.47
	heuristic grid (400 points)	315.97	50.50	154.04	14.52
Incremental linear vector method	fixed (40 updates)	230.96	79.60	297.99	14.68
	fixed (200 updates)	221.28	96.70	271.04	9.32
	fixed (400 updates)	217.14	98.35	254.15	7.77
	random (40 updates)	226.41	90.90	296.96	9.55
	random (200 updates)	219.34	97.70	259.19	6.67
	random (400 updates)	218.13	98.50	247.66	7.07
	order heuristic (40 updates)	226.56	84.15	304.39	12.52
	order heuristic (200 updates)	219.16	97.60	269.01	8.33
	order heuristic (400 updates)	217.13	98.15	254.70	7.85
	2-tier heuristic (40 updates)	230.57	83	319.81	15.40
	2-tier heuristic (200 updates)	220.73	98.30	283.65	8.40
	2-tier heuristic (400 updates)	222.05	98.05	274.17	8.90

Table 5-3: Simulation results for the Maze20B problem (cost minimization) and 2000 random belief points. The table includes results for the perfectly observable MDP control (for the purpose of comparison), MDP-based approximation, fast informed bound method, simple blind policy, least square fit method with Q-function functions - tested for different number of iteration steps (10, 20, 30), grid-based point interpolation strategy with regular, random and heuristic grids (for various grid sizes), grid-based nearest neighbor with random and heuristic grid (for various grid sizes), and incremental linear vector method for fixed random, dynamic random, order heuristic and two-tier heuristic point selection strategies (for different number of updates).

Maze20B (costs): control performance, Test set: 1500 critical belief points

method	method parameters	achieved score	percent of goal	expected score	average observ.
MDP (observable)	-	134.05	100	134.71	0
MDP-approximation	-	214.95	93.40	134.71	3.66
Fast informed bound	-	201.65	96.20	159.77	3.68
Simple blind policy	-	255.99	55.47	313.66	26.03
least square fit with Q-functions	points: 100, iter: 10	199.10	96.20	251.20	28.59
	points: 100, iter: 20	203.45	97.40	232.97	28.05
	points: 100, iter: 30	204.12	97.27	230.70	28.01
Grid based point interpolation method	regular grid (210 points)	216.94	81.53	155.84	7.82
	random grid (40 points)	214.21	94	134.71	3.25
	random grid (200 points)	212.58	94.33	134.71	3.42
	random grid (400 points)	217.45	93.67	134.71	3.83
	heuristic grid (40 points)	213.69	90.27	153.85	5.36
	heuristic grid (200 points)	208.27	93.13	180.48	6.23
	heuristic grid (400 points)	206.31	94.13	183.24	5.93
Grid based nearest neighbor method	random grid (40 points)	314.28	24.60	124.17	42.40
	random grid (200 points)	311.56	36	165.94	25.18
	random grid (400 points)	320.59	26	158.72	38.53
	heuristic grid (40 points)	283.83	53.20	112.14	13.40
	heuristic grid (200 points)	304.31	45.67	131.24	13.09
	heuristic grid (400 points)	264.74	60.40	161.78	11.01
Incremental linear vector method	fixed (40 updates)	211.00	80.87	271.25	13.07
	fixed (200 updates)	199.73	92.73	234.44	8.89
	fixed (400 updates)	199.03	97.60	221.44	5.81
	random (40 updates)	208.35	87.47	270.61	9.90
	random (200 updates)	196.38	98.27	234.11	4.65
	random (400 updates)	197.21	98.07	222.03	4.98
	order heuristic (40 updates)	212.51	81.33	271.44	13.28
	order heuristic (200 updates)	202.78	93.87	232.51	8.43
	order heuristic (400 updates)	196.46	98.87	221.32	5.43
	2-tier heuristic (40 updates)	213.77	81.20	263.59	14.69
	2-tier heuristic (200 updates)	197.63	98.60	224.56	5.98
	2-tier heuristic (400 updates)	199.44	98.80	211.61	6.28

Table 5-4: Simulation results for the Maze20B problem (cost minimization) and 1500 random critical belief points. The table includes results for the perfectly observable MDP control (for the purpose of comparison), MDP-based approximation, fast informed bound method, simple blind policy, least square fit method with Q-function functions - tested for different number of iteration steps (10, 20, 30), grid-based point interpolation strategy with regular, random and heuristic grids (for various grid sizes), grid-based nearest neighbor with random and heuristic grid (for various grid sizes), and incremental linear vector method for fixed random, dynamic random, order heuristic and two-tier heuristic point selection strategies (for different number of updates).

Shuttle: control performance, Test set: 1500 critical belief points

method	method parameters	achieved score	percent of goal	expected score	average observ.
MDP (observable)	-	34.17	100	35.79	0
POMDP optimal	10^{-5} precision	33.99	100	35.71	0
MDP-approximation	-	34.00	100	35.77	0
Fast informed bound	-	34.13	100	35.71	0
Simple blind policy	-	22.03	100	3.26	0
least square fit with Q-functions	points: 100, iter: 10	33.89	100	15.09	0
	points: 100, iter: 20	33.98	100	21.50	0
	points: 100, iter: 30	33.99	100	25.31	0
Grid-based point interpolation method	regular grid (36 points)	33.95	100	35.71	0
	regular grid (120 points)	34.01	100	35.71	0
	regular-grid (330 points)	34.17	100	35.71	0
	random grid (40 points)	33.94	100	35.77	0
	random grid (200 points)	33.96	100	35.77	0
	random grid (400 points)	33.86	100	35.77	0
	heuristic grid (40 points)	34.08	100	35.71	0
	heuristic grid (200 points)	34.11	100	35.71	0
	heuristic grid (400 points)	34.04	100	35.71	0
Grid-based nearest neighbor method	random grid (40 points)	33.96	100	35.81	0
	random grid (200 points)	33.94	100	35.72	0
	random grid (400 points)	34.07	100	35.72	0
	heuristic grid (40 points)	34.01	100	35.71	0
	heuristic grid (200 points)	34.00	100	35.71	0
	heuristic grid (400 points)	33.98	100	35.71	0
Incremental linear vector method	fixed (40 updates)	34.04	100	19.98	0
	fixed (200 updates)	34.09	100	33.44	0
	fixed (400 updates)	34.08	100	35.16	0
	random (40 updates)	25.86	100	21.04	0
	random (200 updates)	34.06	100	33.17	0
	random (400 updates)	34.05	100	34.81	0
	order heuristic (40 updates)	34.02	100	22.50	0
	order heuristic (200 updates)	33.99	100	34.26	0
	order heuristic (400 updates)	34.04	100	35.36	0
	2-tier heuristic (40 updates)	25.58	100	20.50	0
	2-tier heuristic (200 updates)	33.94	100	34.95	0
	2-tier heuristic (400 updates)	34.00	100	35.70	0

Table 5-5: Simulation results for the Shuttle problem and 1500 random critical belief points. The table includes results for the perfectly observable MDP control (for the purpose of comparison), POMDP optimal solution (10^{-5} precision), MDP-based approximation, fast informed bound method, simple blind policy, least square fit method with Q-function functions - tested for different number of iteration steps (10, 20, 30), grid-based point interpolation strategy with regular, random and heuristic grids (for various grid sizes), grid-based nearest neighbor with random and heuristic grid (for various grid sizes), and incremental linear vector method for fixed random, dynamic random, order heuristic and two-tier heuristic point selection strategies (for different number of updates).

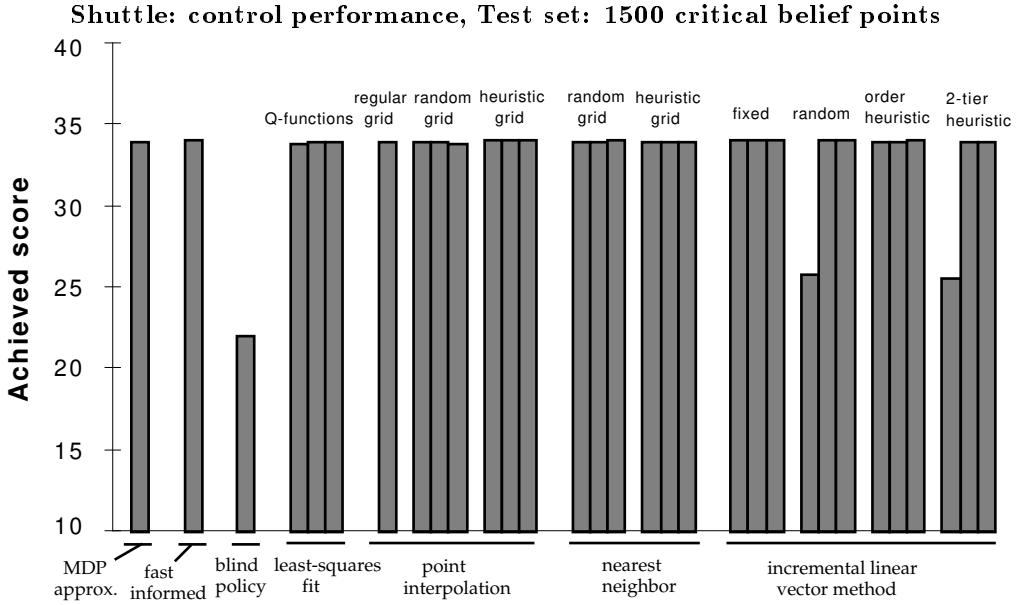


Figure 5-13: Shuttle problem: comparison of control performance. The achieved score represents the average reward on the set of 1500 randomly selected critical belief points.

for the same initial belief point and for any two methods were dependent, that is samples were connected (paired) [Sachs 84]. Then to determine that two methods are different we:

- computed their sample score differences (differences in rewards for all simulation runs);
- tested the null hypothesis that the mean (median) of the differences is zero.

Rejecting the null hypothesis at some significance level then corresponds to the situation in which two methods compared are different.

To check mutual differences of all methods we applied the above difference test pairwise. As sample differences turned out to be nonnormally distributed, we applied the nonparametric Wilcoxon matched pair signed-rank test to test the null hypothesis for every pair of methods (see [Sachs 84]).

The methods were tested pairwise and compared using three significance levels $0.05, 0.01$ and 0.001. Thus it might be the case that the null hypothesis (two methods are equal) is rejected at a higher significance level (say 0.05) but not at the lower level (e.g. 0.01). We performed pairwise significance tests on all problems, using a set of 2000 random belief points for maze problems and a set of 1500 random critical belief points for the Shuttle problem. The complete results of pairwise tests for Maze20 and Maze20B problems, significance level 0.05 and a set of 2000 random belief points are summarized in tables 5-7 and 5-8. The pairwise significance results for the Shuttle problem are simpler. This is because there are only three groups of methods, such that methods from different groups are significantly different, while methods within groups are not significantly different from each other. The first group consists of the blind policy method only; the second from solutions for incremental linear vector method with

problem	test set	worst case difference at significance level		
		0.05	0.01	0.001
Maze20	2000 random	3.13	3.13	3.29
Maze20B	2000 random	29.25	29.25	29.25
Maze20B w/o nearest-neighbor	2000 random	10.05	13.86	14.92
Shuttle	1500 critical	0.27	0.30	0.30

Table 5-6: The table lists worst case differences in achieved scores for all pairs of methods that are not statistically different at significance levels 0.05, 0.01, and 0.001

40 update steps and random and two-tier heuristic strategies; and third group corresponds to all other methods. Note that these groups mimic nicely the differences in average scores achieved.

For the set of test results obtained, when two methods are shown to be statistically significantly different at some level of significance, we would like to know whether and how that difference shows up in their average score. To get an estimate of this, we performed the following:

For each problem, and for each pair of methods that failed to be significantly different (at a given significance level), we recorded the difference in average scores registered by the two methods. To present these data in a simplified form, we show the maximum of these differences in Table 5-6. Any score differences larger than these numbers correspond, in our experiments, with pairs of methods that are indeed significantly different from each other.

Worst-case difference quantities for the Maze20B problem turned out to be influenced mostly by grid-based nearest-neighbor methods combinations (with bad performance, relatively large score spans and no significant statistical difference). Because of that we also computed worst-case differences without nearest-neighbor entries and included them in the table 5-6. Note that the listed quantities do not mean that a new method with a larger score difference is automatically different at the given significance level nor that two methods with smaller average score span cannot be significantly different. This is simply because the test used relies on 2000 or 1500 matched pairs of results and not the average score. Note also that difference quantities should not be compared across problems.

Overall the significance test showed that methods with larger achieved score differences are indeed statistically different. This means that observed differences are highly likely not to be simply the result of randomness and comparison of the methods along achieved average scores is justified.

5.3.4 Evaluation of results

MDP based approximation

The MDP based approximation method constructs a value function using a solution for a perfectly observable case, that is one in which no partial observability and no investigative actions need to be considered. Therefore, the solution is likely to approximate the optimal value function better for problems with less uncertainty and partial observability. This was confirmed also in our experiments, in which MDP-based approximation posted poor results for the Maze20 problem, good results for the Maze20B problem and excellent scores for the Shuttle docking problem.

Maze20: pairwise significance test, Test set: 2000 random belief points

method	method parameters	ref. num.	achieved score	methods not different at significance level 0.05
MDP (observable)	-	1	131.16	
MDP-approximation	-	2	38.24	13, 14, 15
Fast informed bound	-	3	54.73	5, 6, 7, 25
Simple blind policies	-	4	50.18	10, 18
Least square method with Q-functions	points: 100, iter: 10	5	56.7	3, 6, 25, 28
	points: 100, iter: 20	6	56.59	3, 5, 25, 28
	points: 100, iter: 30	7	54.73	3, 25
Least square method with softmax function	vectors:10, points:50, iter:10	8	45.94	17
	vectors:10, points:50, iter:20	9	45.68	17
	vectors:15, points:100, iter:10	10	49.1	4, 18
	vectors:15, points:100, iter:20	11	51.43	
Grid based point interpolation method	regular grid	12	44.6	
	random grid (40 points)	13	40.08	2, 14, 15, 16
	random grid (200 points)	14	40.45	2, 13, 15, 16
	random grid (400 points)	15	41.37	2, 13, 14
	heuristic grid (40 points)	16	41.13	13, 14
	heuristic grid (200 points)	17	46.08	8, 9
	heuristic grid (400 points)	18	48.37	4, 10
Grid based nearest neighbor method	random grid (40 points)	19	31.49	
	random grid (200 points)	20	33.87	21, 23
	random grid (400 points)	21	35.38	20, 23
	heuristic grid (40 points)	22	32.43	
	heuristic grid (200 points)	23	36.33	20, 21
	heuristic grid (400 points)	24	38.75	
Incremental linear vector method	fixed (40 updates)	25	56.16	3, 5, 6, 7, 28
	fixed (200 updates)	26	60.18	27, 29, 30, 32, 35, 36
	fixed (400 updates)	27	60.62	26, 29, 30, 32, 33, 35, 36
	random (40 updates)	28	56.82	5, 6, 25
	random (200 updates)	29	60.45	26, 27, 30, 31, 32, 35, 36
	random (400 updates)	30	60.98	26, 27, 29, 32, 33, 35, 36
	order heuristic (40 updates)	31	59.18	29, 34, 35
	order heuristic (200 updates)	32	61.01	26, 27, 29, 30, 35, 36
	order heuristic (400 updates)	33	62.41	27, 30, 36
	heuristic 2-tier (40 updates)	34	58.63	31, 35
	heuristic 2-tier (200 updates)	35	59.47	26, 27, 29, 30, 31, 32, 34
	heuristic-2-tier (400 updates)	36	61.72	26, 27, 29, 30, 32, 33

Table 5-7: The results of the pairwise significance tests for the Maze20 problem and a set of 2000 randomly selected belief points. The combinations of methods for which the null hypothesis (two methods are same) cannot be rejected at significance level 0.05 are listed. Every pair of methods was tested using nonparametric Wilcoxon matched pair signed-rank test.

Maze20B (costs): pairwise significance test, Test set: 2000 random belief points

method	method parameters	ref. num.	achieved score	methods not different at significance level 0.05
MDP (observable)	-	1	133.37	
MDP-approximation	-	2	245.06	10, 13, 14
Fast informed bound	-	3	230.48	7, 11, 21, 24, 27, 30, 32
Simple blind policies	-	4	264.74	8
Least square method with Q-functions	points: 100, iter: 10	5	225.08	6, 7, 22, 24, 25, 27, 28, 31, 32
	points: 100, iter: 20	6	224.47	5, 7, 22, 24, 25, 26, 28, 31, 32
	points: 100, iter: 30	7	225.43	3, 5, 6, 22, 24, 27, 28, 31, 32
Grid based point interpolation method	regular grid	8	254.68	4
	random grid (40 points)	9	236.63	10, 11, 13, 14, 21, 30
	random grid (200 points)	10	239.75	2, 9, 11, 13, 14, 21
	random grid (400 points)	11	236.94	3, 9, 10, 13, 14, 21, 30
	heuristic grid (40 points)	12	250.88	
	heuristic grid (200 points)	13	240.74	2, 9, 10, 11, 14
	heuristic grid (400 points)	14	241.33	2, 9, 10, 11, 13
Grid based nearest neighbor method	random grid (40 points)	15	342.41	
	random grid (200 points)	16	345.23	20
	random grid (400 points)	17	338.3	20
	heuristic grid (40 points)	18	331.55	19, 20
	heuristic grid (200 points)	19	339.35	18
	heuristic grid (400 points)	20	315.97	16, 17, 18
Incremental linear vector method	fixed (40 updates)	21	230.96	3, 9, 10, 11, 24, 30
	fixed (200 updates)	22	221.28	5, 6, 7, 24, 25, 26, 27, 28, 29, 31, 32
	fixed (400 updates)	23	217.14	25, 26, 28, 29
	random (40 updates)	24	226.41	3, 5, 6, 7, 21, 22, 27, 30, 31, 32
	random (200 updates)	25	219.34	5, 6, 22, 23, 26, 28, 29, 31, 32
	random (400 updates)	26	218.13	6, 22, 23, 25, 28, 29, 31, 32
	order heuristic (40 updates)	27	226.56	3, 5, 7, 22, 24, 30, 31, 32
	order heuristic (200 updates)	28	219.16	5, 6, 7, 22, 23, 25, 26, 29, 31, 32
	order heuristic (400 updates)	29	217.13	22, 23, 25, 26, 28
	heuristic 2-tier (40 updates)	30	230.57	3, 9, 11, 21, 24, 27
	heuristic 2-tier (200 updates)	31	220.73	5, 6, 7, 22, 24, 25, 26, 27, 28, 32
	heuristic-2-tier (400 updates)	32	222.05	3, 5, 6, 7, 22, 24, 25, 26, 27, 28, 31

Table 5-8: The results of the pairwise significance tests for the Maze20B problem and a set of 2000 random belief points. The combinations of methods for which the null hypothesis (two methods are same) cannot be rejected at significance level 0.05 are listed. Every pair of methods was tested using nonparametric Wilcoxon matched pair signed-rank test.

Blind policy method

The simple blind policy method produces a solution that combines the behaviors of blind (no-information) agents. This means that the performance of such a solution should perform badly when observations are very informative and can help significantly reduce the uncertainty about the underlying process state. On the other hand, the solution should be better when observations are very noisy and are imprecise indicators of the underlying process state. In such a case, acting blindly should yield results closer to results from informed but very weak information source. Again, we were able to see such a behavior on our test problems in which a relatively good performance was achieved on the Maze20 problem and worse results for the Maze20B and Shuttle docking problems.

Fast informed bound method

We were surprised by the very good performance of the newly designed fast informed bound method. Interestingly, this method uses only $|A|$ linear vectors (equals the number of actions). Very good results on all test problem can be attributed mostly to the update strategy the method employs. It is best viewed as an approximation of the Sondik's update rule. It looks like that this strategy produced a relatively good approximation of the shape of the "exact" value function for all of the tested POMDP problems. However, the fact that our problems had sparse transition matrices and single "goal" state might have been an important factor in this respect.

Least-squares fit

The approximate value iteration method with the least-squares fit approach was tested with a linear Q-functions model and for the Maze20 also with a softmax function model. The Q-function method achieved very good simulation results on all three problems. Interestingly, for Maze20, a simpler Q-function model achieved better results than more complex softmax model with 10 and 15 linear vectors. Despite the threat of instability, the functions seemed to stabilize after about 15 iterations and did not change dramatically afterwards. For the softmax model we also tried to start the approximate value iteration using different initial functions and the same set of grid points. The value function seemed to stabilize again but in a different region. This behavior can be attributed to the method's problems with unique convergence.

In general, the high performance of the least-squares fit can be attributed to the choice of a function model that allowed us to match the optimal value function reasonably well because of its the convex and piecewise linear shape.

Grid-based interpolation-extrapolation methods

The grid-based interpolation-extrapolation methods represented by the point interpolation and nearest-neighbor approaches posted different results on different problems. The results differed also for random and heuristic grid-point selection strategies.

The results show that for the point interpolation entries the scores achieved and their quality are closely related to and depend on the performance of the MDP-approximation. That is, a performance of MDP-approximation solution is correlated with a performance of point interpolation methods. Although methods with point interpolation rules improved on the MDP-based approximation, especially for problems in which MDP-based approximation performed poorly (Maze20, Maze20B), we did not see a dramatic difference (even for larger grid sizes), and other methods (e.g. fast informed bound and incremental linear vector method) were usually more

successful. Heuristic grids tended to improve the performance most of the time, especially for problems in which MDP-based approximation method performed badly (Maze20 problem).

The nearest neighbor approach delivered the worst performance on both maze problems, although the performance was usually boosted by heuristic grids. The only problem where nearest neighbor achieved results comparable to other methods was the Shuttle docking problem. We believe that main reasons for this are: the optimal value function for the Shuttle docking problem is relatively flat; the differences in values for all critical points of the belief space are not very big; and the grid sizes we tested were sufficient to sample enough of the relevant belief space (8 states). On the other hand, a poor showing of the method on both maze problems for tested grid sizes can be attributed to the inability of the method to approximate the shape of the optimal value function properly.

We believe that the main reason for the poor performance of nearest neighbor and not very convincing results for point interpolation rules on both maze problems was that they were not able to fit the shape of the optimal piecewise linear and convex value function properly. The shape of the value function for the grid-based point interpolation-extrapolation techniques is influenced strongly by the selection of grid points and an interpolation-extrapolation rule used to estimate nongrid points. Then poor choices of grid points and interpolation-extrapolation rules lead to poor shape approximations (e.g. the choice of nearest neighbour rule leads to piecewise constant function).

Incremental linear vector methods

The best performance was obtained by the incremental linear vector method with Sondik's updates. The method was tested using multiple point selection strategies that included fixed, random, and heuristic approaches. In all cases the method was started from the initial simple blind policy solution. Interestingly, the differences between various point selection strategies turned out to be very small, and we did not observe any significant improvement using any of the methods. The slim differences can be explained by:

- The effect of a derivative (linear vector) update is that it approximates well also points in the neighborhood of the belief point that seeded the update. This reduces the sensitivity of the method to a specific point selection strategy.
- The shape of the optimal solution over the belief space is approximated well in all cases using a relatively small number of incremental updates. This is also supported by the fact that no or small improvements in performance for value functions were seen for solutions ontained after 200 and 400 updates.
- The heuristics used are not very good and better point selection strategies can be devised.

The method was able to eliminate relatively rapidly the dependence and disadvantage from the initial value function choice (simple blind policy solutions) on all three test problems. This is documented by a significant improvement of performance of the method (after more updates) for cases in which blind policy solution performed poorly, like the Maze20B and the Shuttle docking problem. Overall we believe that the high performance of the method is mainly due to its ability to approximate the shape of the optimal value function well.

5.3.5 Summary of test results

Top performers

Overall, the best control performance was obtained by the new incremental linear vector method with Sondik's updates. It achieved the best or close to best results on all three test problems. The second best performer was the least-squares method with linear Q-functions. The third best performer was the newly proposed fast informed bound method. Interestingly all three methods delivered top results on all three test problems; their performance was not test specific. Also significance tests for the Maze20 and Maze20B problems confirmed that these methods indeed differ from the others. For the Maze20 problem the linear vector methods (with more updates) turned out also to be significantly different from the other two top performers.

The unifying factor of the three best performing methods is that all of them try to approximate the shape of the value function over iteration steps. This is done: in the case of linear vector method by updating derivatives (linear vectors) using exact Sondik's update rule, for the fast bound method by using a simple update rule that approximates the derivatives of the value function, and for the least-squares fit by providing suitable piecewise linear and convex parametric models. The value iteration procedure that tries to preserve the shape over iteration steps also tends to approximate the shape of the optimal value function better. This is very important for the control problem, where we would like to guess the right relative (rather than absolute) value function values for different belief points.

The top three methods have quite different properties. The incremental linear vector method gradually improves the piecewise linear function. Under the appropriate point selection strategy is guaranteed to converge to the optimal solution. Unfortunately, the price paid for this is that the complexity of the value function grows with every iteration although this growth is at most linear (in contrast to the potential exponential growth for the exact method). On the other hand, both the Q-function least-squares fit method and the fast informed bound method work with restricted value functions that consist of $|A|$ linear vectors. This keeps the complexity of their updates constant over iteration steps. The main difference between the two methods is that the fast informed bound method computes new updates directly (which can be done quickly). It upper bounds the exact update and it is guaranteed to converge uniquely. On the other hand, the Q-function least-squares fit needs to sample and update a set of belief points first, does not provide bounds and is not guaranteed to converge uniquely.

Worst performing methods

All of the other methods tested had results that were more problem sensitive. We believe that the main factor in all cases was the shape of the value function used, which can be more or less suitable for the problem at hand.

The simulation results showed that the grid-based nearest neighbor method performs worst. Especially bad were the results it obtained on maze problems, where it was outperformed by all other methods (even by simple blind policy and MDP approximations) by a large margin. This is both for heuristic and for random grids. The significance tests for maze problems showed that results achieved by grid-based nearest neighbor methods are not a consequence of randomness and that they differ significantly from all other contenders. The main reason for its poor performance was the usage of the piecewise constant value functions that do not fit the underlying piecewise linear and convex value functions for smaller grid resolutions very well. This makes the nearest neighbor method unsuitable for larger belief space POMDP problems, and even in a case when the grid resolution is sufficient, there are always more efficient and better methods available.

method	runtime (sec)
MDP-based approximation	2
Simple blind policy	7
Fast-informed bound	95
Least-squares fit with Q-functions (100 sample points), 10 parallel updates	299
Grid-based point interpolation with the 210 point regular grid	736

Table 5-9: Execution times of some of the implemented algorithms on Maze20 problem. The algorithms were implemented in Lucid Common Lisp and were run on a Sun Microsystems SPARC-10. All methods except least-squares fit were implemented using value iteration strategies and relative stopping criterion with a precision parameter 0.1. Least-squares fit method with linear Q-functions used 100 sample points and parallel updates. Time it took the method to compute 10 iterations is listed. Also grid-based point interpolation method with the regular grid of size 210 points is used.

5.4 Runtimes of methods

Value function approximation methods were implemented in Lucid Common Lisp and were run on a Sun Microsystems SPARC-10. Times to compute results for different methods varied on different test problems, point selection strategies and/or choice of other parameters (like precision parameters for the value iteration strategy). Also various methods presented here have been implemented with different degrees of effort devoted to optimization of the calculations. Therefore, differences in execution time should not be a basis for relative comparison of the methods. The runtime results are presented here for informational purposes only.

Tables 5-9 and 5-10 show runtimes of some of the implemented algorithms for the Maze20 problem in seconds. Table 5-10 is used for incremental methods and lists times needed to achieve the improvement for a new grid-size or new set of updates. To illustrate the strong dependence of methods on parameters, assume that we change the precision parameter for the grid-based point interpolation method with the regular grid in table 5-9 from 0.1 to 0.4. Then the execution time of the method drops from 736 seconds to 256 seconds.

The running times for the Maze20B problem were not very far from those for the Maze20. On the other hand, solutions for the simpler Shuttle problem were acquired very quickly, for example a solution for the fast informed bound method (with precision 0.1) was computed in less than 1 second, and all 400 incremental linear vector method updates took about 30 seconds.

5.5 Experimental biases

Although we have tried to cover a spectrum of POMDP problems of different complexity in our experiment, it is our obligation to point out known deficiencies and possible gaps in our approach. The main problem we currently see is related to the selection of test problems used for our experiments.

In general all of the test problems have relatively small transition and observation complexity, for example there are at most four possible adjacent rooms the robot can move into from any specific room. Because of this the transition and observation matrices are sparse with a lot of zeros. Thus the simulation results and the evaluation of methods are biased towards problems with such local characteristics. Although this bias may be justified for many real-world

method	init. time	runtime (sec) for incremental improvements									
		40	80	120	160	200	240	280	320	360	400
grid-based point interpolation	2	24	135	162	288	395	502	778	881	818	1402
grid-based nearest-neighbor	2	70	642	1213	1318	1387	2304	2934	5938	7237	3731
incremental linear vector method	7	129	397	652	932	1168	1328	1455	1593	1772	1933

Table 5-10: Execution times of some incremental algorithms for the Maze20 problem. Grid-based point interpolation method uses the interpolation strategy and the the heuristic grid expansions proposed in section 4.7.3. For every grid refinement (increase of a grid by 40 points) the method is iterated until the precision of 0.4 reached. Time it takes the method to obtain the solution for a new grid is measured and listed. Grid based nearest-neighbor works with the same heuristic grid selection strategy and relative stopping criterion. Incremental linear vector method uses a fixed set of 40 points that are repeatedly updated (using Sondik's linear vector updates). The method is initiated with a simple blind policy solution and times to execute 40 update increments are listed.

problems (with natural local characteristics) to make the experiment better we also need to test problems with high connectivity.

5.6 Summary

Test conclusions

In the first part of the experiment we tested bounds produced by several different value function methods using various grid sizes and number of updates. Although we were able to obtain significant improvements in the bound quality for both initial bounds (MDP-based approximation and simple blind policy method), we were not able to achieve very tight bound differences for the Maze20 problems for the tested range of grids and updates. This suggests that high quality bounds are very likely hard to obtain for larger and more complex POMDP problems. Thus, in general, one cannot expect to get very good bounds for complex problems for free and one needs to pay the toll for each improvement.

In the second part of the experiment we tested the control performance of several value function approximations. All tested methods were evaluated using a score representing average achieved reward (cost) for a set of simulation runs for two sets of belief points. To make sure that scores obtained are not a result of randomness (a sensible concern when dealing with stochasticity) we performed pairwise statistical significance tests for all methods. These tests showed that methods with larger achieved score differences indeed differ significantly and thus evaluation along average scores for tested methods is justified.

The control performance achieved by different methods seems to be completely unrelated to the quality of the bounds. This confirms that for the purpose of control, it is not absolute but relative values, that is the shape of the value function that matters. This also gives us hope that there are fast and efficient methods that can lead to good control performance for more complex POMDP problems. This is not true when the criterion used to judge the method is the quality of the bounds in absolute terms.

Methods that achieved the best simulation results used piecewise linear and convex representations of value functions, and attempted to approximate the shape of the value function over update steps. The clear winner for all three problems was the new incremental linear vector method that computes linear vectors (derivatives) for selected points and incrementally updates the existing piecewise linear and convex approximation with new vectors. The advantage of the incremental linear vector method is that it can continue to improve the acquired solution with more time. In the limit, under a suitable point selection strategy, it converges to the optimal value function. The other top performers were the least-squares method with Q-function model and the fast informed bound method (also a new one). They use a restricted number of $|A|$ linear vectors over all iteration steps. This is unlike the incremental linear vector method which can grow the size of the linear vector set with every iteration. Of the two, only the fast informed bound method is guaranteed to converge; the least-squares fit can suffer from the problem of instability and divergence.

The worst control performance was achieved by the grid-based nearest-neighbor method, which approximates the optimal piecewise linear and convex value function with a piecewise constant function. The results suggests that nearest-neighbor is not suitable for the purpose of control for large belief space POMDPs and that far better alternatives are available.

Contributions

This chapter presents and analyzes experiments we have performed on a set of three infinite discounted horizon problems of different complexity using a large variety of different approximation and bound methods. The need of large scale experiments in the POMDP domain for the future exploration and understanding of the domain is enormous. Thus, the main contribution of our work is in the large experimental study, providing achieved results and their interpretation.

In our work we have experimentally tested various new and existing value function approximation methods, their extensions and modifications. The results presented showed that there are various efficient alternatives to the least-squares fit approach that seemed to dominate the AI literature. These are based on grid-based or other alternative approaches, like fast informed bound. Their main advantage is that they do not suffer from the threat of potential instability when combined with the value iteration method to solve infinite discounted horizon problems. Also results they achieve are often superior or comparable to those by the least-squares method.

Chapter 6

Extending POMDP framework: Management of ischemic heart disease

The main advantage of the POMDP framework is its ability to model two sources of uncertainty that stem from action outcome and imperfect observability. This does not mean that the basic framework can be applied to any domain without changes or that the framework can capture all important features of any domain. When dealing with real-world problems we must often "adapt" the formalism to fit the domain. Extensions can be made in both directions. Some extensions make the problem more complex but are required to solve the problem, e.g. models in which observations are delayed. Other extensions take advantage of domain specific features, model more of the underlying problem structure and make it possible to speed-up problem-solving routines.

In the following we will explore and propose various new structural extensions to the basic POMDP framework. These exploit additional problem structure and help us to reduce the complexity of problem-solving methods. The extensions are studied in the context of a real-world problem from the area of medical therapy planning – the problem of management of patients with ischemic heart disease(IHD) [Wong et al. 90] [Leong 94] [Hauskrecht 96a] [Hauskrecht 97a].

6.1 Modeling diagnostic and therapeutical processes using POMDPs

Throughout the history of AI in medicine, a large amount of research work has been devoted to the development of methods and techniques capable of modeling the decision process of a physician under uncertainty. The focus of work in this area has gradually shifted from problems with static features to ones that emphasize the dynamic aspect of the decision process. While most of the work on dynamic decision making addresses the issue of action outcome uncertainty, the feature of partial observability is often irrelevant or is abstracted out. Research work that assumes perfect observability includes the management of chronic heart disease [Leong 94] and diabetes therapy planning [Hovorka et al. 92].

The assumption of perfect observability may not work well for problems in which observa-

tions are imprecise indicators of the patient state (as is often the case in assessing underlying disease) and when investigative actions have significant cost (such as invasiveness and economic expense). In such cases, careful evaluation of the costs and benefits associated with both treatment and investigative actions with regard to global objectives is necessary and therapeutical and investigative actions are often interleaved over the course of the treatement. A formalism that allows us to capture the complexity of such a process is the POMDP framework, which models both sources of uncertainty, various investigative and treatment choices, and costs and benefits associated with such interventions and their outcomes.

6.1.1 Chronic ischemic heart disease

An example of a therapy planning problem that requires one to consider two sources of uncertainty is the management of chronic ischemic heart disease (IHD) [Wong et al. 90] [Leong 94] [Hauskrecht 96a] [Hauskrecht 97a].

Ischemic heart disease is a condition that is caused by an imbalance between the supply of available oxygen and the demand for oxygen by heart muscle. This imbalance can cause cardiac disfunction and subsequent impairment to blood circulation. The most common cause of ischemia is coronary artery disease, which corresponds to the narrowing of coronary vessels that reduces the perfusion of heart muscle most commonly due to atherosclerotic changes. Coronary artery disease is a progressive disease with aggrevating symptomatology and cardiac impairment. The leading symptom of ischemic heart disease is chest pain (angina). Coronary artery disease can also be accompanied by various complications, for example acute myocardial infarction (MI).

Treatment of coronary artery disease can be conservative, using medications like Beta blockers, or more invasive, using surgery that attempts to repair obstructed coronary arteries. There are two commonly used surgical procedures: percutaneous transluminal coronary angioplasty (PTCA) and coronary artery bypass graft surgery (CABG). Both procedures carry an increased risk of death, a risk of perioperative MI and cause a lot of pain and discomfort for the patient.

One problem with assessing the status of coronary artery disease involves its ability to change over time. In general, it is not possible to state the current status of patient's coronary arteries, or identify the level of the patient's ischemia (O_2 demand-suppply mismatch). However there are investigative procedures that can reveal more about the underlying disease, such as an angiogram or a stress test. Unfortunately these procedures are also invasive and/or carry increased risks of MI and death.

Investigative and treatment actions can be repeated or changed over time depending on the progression of the disease. For example, a patient can have several PTCA's over a span of a few years to clear coronary arteries, improve perfusion and relieve chest pain symptoms, or a patient might undergo several stress tests. The *objective of the problem* is to determine the best possible treatment step or sequence of steps with regard to various treatment goals or objectives. These include the following qualitative goals:

- increase in the quality of life (e.g. relieve chest pain symptoms)
- decrease the chance of acute episodes (MI)
- increase length of life
- decrease the invasiveness of procedures
- decrease the economical cost of associated procedures

In order to optimize such objectives one needs to consider various treatment alternatives now and in future, their possible outcomes, and benefits and risks of such choices with regard to globally pursued goals.

6.1.2 A typical decision scenario

While doing routine medical screening, it is discovered that a patient shows signs of ischemia on a resting EKG. No other observations (chest pain, etc.) are positive. Possible actions for the physician are:

- *Do nothing and observe the patient. However, this leaves the patient at a higher risk of MI or possible death.*
- *Administer medication, that tends to reduce the risk of MI (e.g. Beta blocker, aspirin).*
- *Request an angiogram, that reveals the precise status of the coronary arteries, and helps determine whether the patient is in a higher risk group. Unfortunately, the angiogram has an additional expense including patient discomfort, and risks of MI or death.*

Once a decision has been made, such as prescribing the medication, the same decision process must be repeated again after some time period. The result can be the same or an alternative choice based on new observed symptoms and findings at that time. For example, worsening of the symptoms would cause one to consider an angiogram possibly followed by angioplasty or bypass surgery. At each time point, the actual decision must take into account future progress of the disease, as well as future treatment and investigative choices.

6.2 Using POMDP to model IHD

The management of ischemic heart disease embodies characteristics that match well many features of the POMDP formalism. The major problem faced when applying the POMDP framework to the management of IHD is the size and complexity of the IHD model. Its complexity is far beyond current limits of exact POMDP problem solving procedures. In order to overcome this hurdle we focused on two complementary solutions:

- reduction of the complexity of the POMDP model by representing underlying structure;
- substitution of exact solution methods with approximate ones.

The objective of the first approach is to capture and represent more features and structure of the underlying domain model, as compared to the direct application of the POMDP formalism, and thus hope to reduce significantly state and observation space sizes one needs to work with. This work led to a model with factored states consisting of both observable and partially observable state variables. The second approach is typical when applying the POMDP methodology to more complex problems and was the center of our discussion in previous chapters.

In the following we will describe the components of the POMDP model proposed and designed for the ischemic heart disease domain.

6.2.1 Representing states

The state of a patient at any instance of time can be described using a finite set of random state variables. State variables used in the IHD problem and their possible values are shown

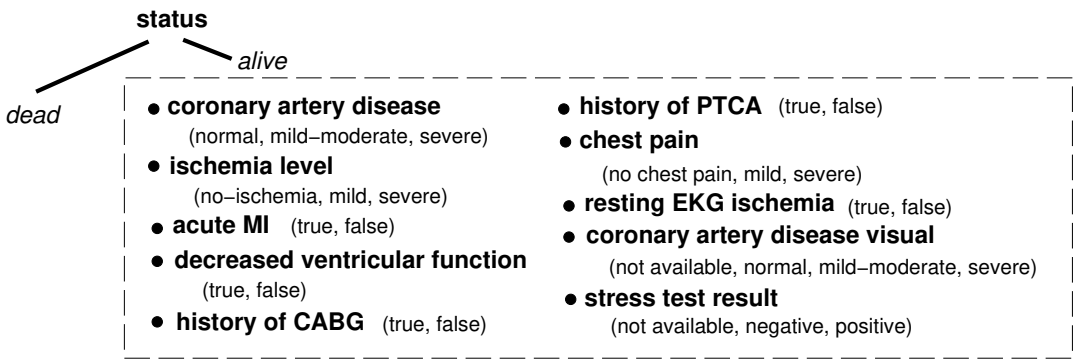


Figure 6-1: Ischemic heart disease model: state variables representing process states and observations.

in figure 6-1. In general every possible assignment of values to state variables describes one unique patient state. However, the set of state variables we used for the ischemic heart disease (IHD) problem is not homogenous and explicitly restricts certain combinations of variable values using hierarchical (conditional) subsumption. The hierarchical constraints represent situations in which some combinations of variables and their values are either impossible or are irrelevant for the problem at hand. For example for the state variable, *status*, with possible values of dead and alive, values associated with *acute-MI, coronary artery disease* or *chest pain* either do not make sense or are irrelevant when the patient is dead. Using hierarchical subsumption, state variables are "enabled" only when "status" is set to an appropriate value. Hierarchical subsumption is also useful for describing a state using different levels of detail, i.e. state variables can describe both higher level abstracted state components as well as their lower level elaboration. The state variable structure for the IHD problem is represented in figure 6-1; lower level state variables are enclosed in the rectangle.

State variables in the model can be used to describe the dynamics of a disease process over time. These variables are called *process state variables*. A set of process state variables for the IHD problem is shown in figure 6-2. Notice that for example state variable *chest pain* is not a process state variable. This is because it is assumed not to influence directly the dynamic behavior of the disease process.

State variables can be *observable*, that is they correspond to variables that can be seen directly at any point in time. On the other hand, variables that cannot be observed directly correspond to *hidden variables*. The set of observable variables for the IHD problem is listed in figure 6-2. Notice that a state variable can be both a process state variable and an observable variable (e.g. *decreased ventricular function*). This captures the fact that not all process state variables in a POMDP need to be represented and treated as hidden (or partially observable) variables. This is also one of the major deviations from the ordinary POMDP model that assumes that only partially observable process states exist. Explicit representation of both observable and hidden process state variables allows us to combine the advantages of MDP and POMDP formalisms. It leads to speed-ups in manipulation, inference and planning routines by means of reducing the complexity of the information state space. This issue will be discussed later.

Our POMDP model for the IHD is designed to evaluate and reason about the consequences

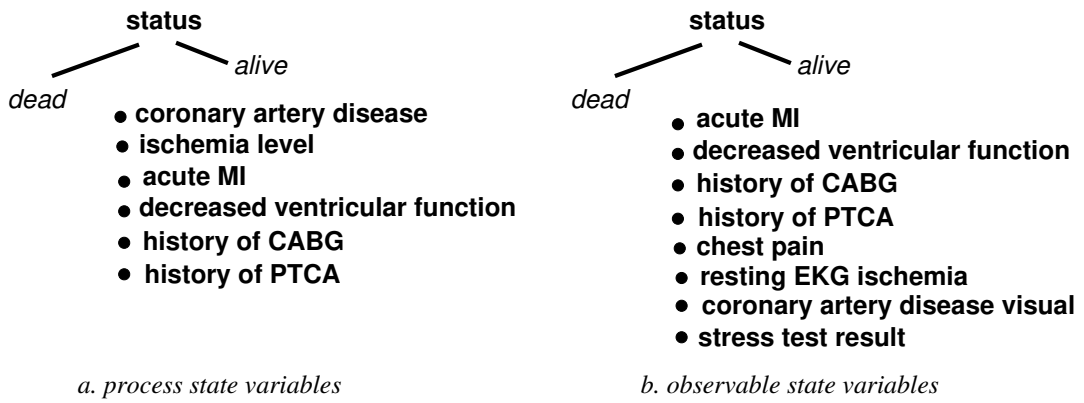


Figure 6-2: Ischemic heart disease model: types of state variables.

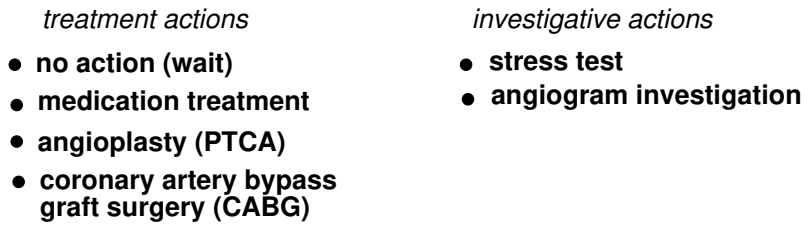


Figure 6-3: Ischemic heart disease model: actions.

of long term treatment. The model omits many short term decisions, e.g. those that are related to the management of acute chest pain. Also some state variables like *acute MI* are considered to be observed directly. The state description also includes state variables reflecting the history of angiopasty (*history PTCA*) and the history of bypass surgery (*history CABG*). The reason for including these is that process states need to satisfy the Markov property and the transition function that maps previous to next state is believed to be influenced strongly (at least in some instances) by past PTCA or CABG procedures.

6.2.2 Actions

Actions correspond to treatment or investigative procedures (see figure 6-3). *no-action* corresponds to the choice in which no treatment or investigative action has been selected. In general, treatment actions actively change the state of the patient to more appropriate state. Investigative actions explore the state of the patient, especially the related hidden process state variables. However investigative actions may not only reveal more about the underlying patient state but can also lead to a change in the state (e.g. patient can die or get *acute MI* as a result of an *angiogram* procedure).

In general a set of *actions* in the POMDP model can have exploratory, transitional and cost effects. The exploratory effect of actions is based on their ability to induce observations that

in turn can be suggestive of some internal states. An example is an angiogram investigation or stress test. The transition effect of the action is represented by its ability to change the internal state of the patient: for example *PTCA* can lead to the reopening of the blood supply in the main vessels. The third effect of actions is their cost which can be measured in terms of patient suffering, patient discomfort and/or financial cost. Actions that have only an exploratory effect and are neither associated with a cost nor affect the state transition are not explicitly represented in the action set.

6.2.3 Representing stochastic transition and observation models

Representing states in a factored form is very useful for representing various independencies and regularites that enrich the stochastic relation between states and actions over time. In the traditional POMDP model, stochastic relations between Markov process states over time on one side and process states and observations on the other, are represented using transition and observation matrices. These define probability distributions for the patient state changes under specific interventions, reflecting for example the fact that the patient with coronary disease can either die, suffer MI, or receive coronary artery repair as a result of PTCA or CABG, with some probability or that severe ischemia can, to various degrees, lead to mild, severe or even no chest pain.

Graphical models

Probabilistic independences and regularities between variables in factored form can be often represented using graphical models, e.g. a Bayesian network. Figure 6-4 illustrates the transition and observation model built for the ischemic heart disease problem using a Bayesian network approach. In this figure, random (chance) variables are represented by circles, and actions as rectangles. Patterned circles correspond to observable variables, that is variables for which values are assumed to be known at every time step. The graphical model shown in the figure does not correspond to a typical Bayesian network but to its hierarchical extension, where sets of random variables can be hierarchically subsumed by other variables. This extension allows us to represent types of independences that otherwise could not be represented when a flat variable set is used.

The advantage of the hierarchical Bayesian network is illustrated on our IHD model. The transition probabilities between previous state (at time t-1) and a state where the patient is alive and suffers from coronary artery disease of some severity are represented using two probability distributions, each exploiting different independences. The first probability distribution concerns the variable *status* and represents the distribution of a patient being alive or dead as a result of some procedure performed in the previous state. This distribution depends on values of state variables in the previous time step. The second distribution represents a conditional distribution of coronary artery disease given a previous state, an action and patient being alive. Such a distribution can exploit a different set of independences, i.e. the value of the *coronary artery disease* variable being independent of some previous state variables when the patient is alive. In general the hierarchical subsumption may capture different sets of independences for different levels of detail. This leads to a simpler representation of the conditional distribution for a patient being alive and suffering from coronary artery disease of certain severity, because the conditional distribution can nicely decouple along different levels of detail.

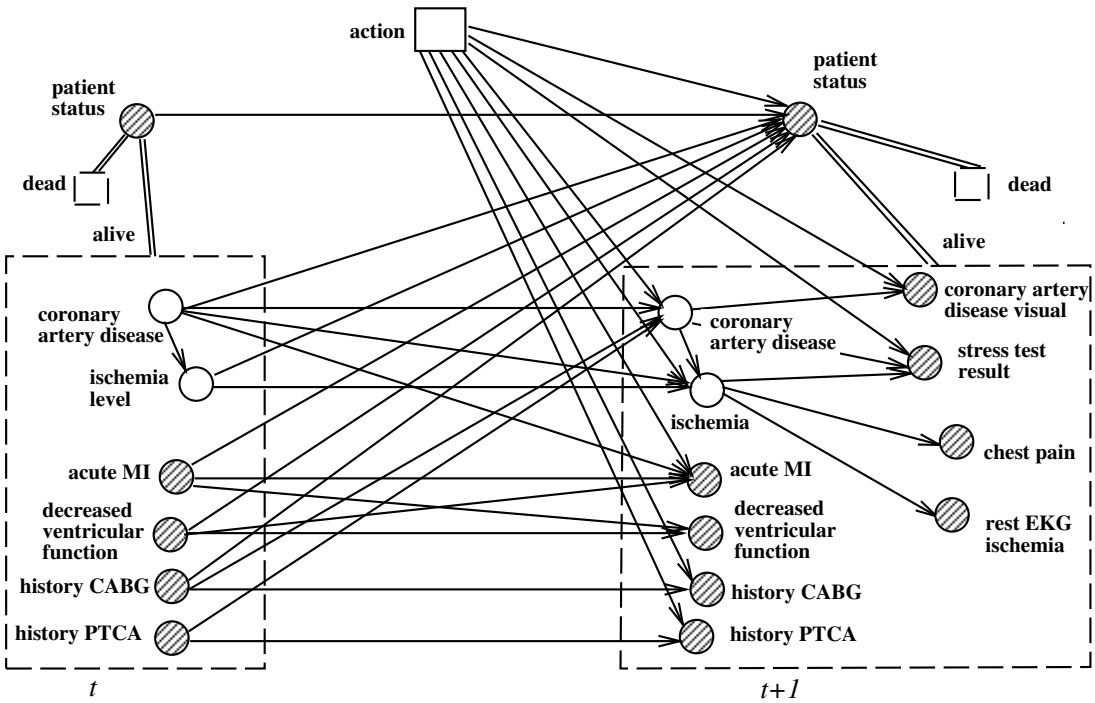


Figure 6-4: Ischemic heart disease: transition and observation model.

Representing additional relations and constraints

Ordinary Bayesian networks can represent conditional or marginal independences that hold between variables. However, there are other relations that often hold and can be useful for both making the probabilistic model more compact and for planning. These may include partial conditional independence (the independence relation does not hold for all possible values of conditioning variables but for a subset of values) or various deterministic relations and constraints (some value assignments are impossible because values of two or more variables are incompatible).

For the IHD problem we focused on the problem of modeling additional deterministic constraints. Constraints are expressed by means of rules that restrict some combinations of the variable values. For example the variable *history PTCA* in the transition model can change from *false* to *true* only when action *PTCA* has been chosen in the previous step. Similarly, the same variable once it is *true* remains *true* forever. The deterministic constrains are represented by a set of rules, such as:

Rule 5: **If** (PAST (history-PTCA true))
then (history-PTCA true)

Deterministic constraints can be useful in speeding up probabilistic computation and so we used them heavily to construct compiled POMDP model, which we discuss later.

6.2.4 Initial state model

Once the transition and observation model is defined, we can expand the above belief network model as many time steps as needed. This allows us to compute various probabilistic queries with regard to variables over different time instances. However, before we can answer such queries we need to know prior probabilities for the initial process state variables, that is priors for the process state variables at time $t = 0$.

For our IHD model and related computations, it is neccessary to compute initial probablities for all hidden process state variables (namely *coronary artery disease, ischemia*). All other process state variables are assumed to be observable and are therefore directly available. The set of initial probabilities can be computed using the prior model in figure 6-5. The prior model overlaps with the transition and observation model and adds a new variable of *prior coronary artery disease* that influences the hidden variable *coronary artery disease* and models explicitly the prior knowledge about the distibution of coronary disease severities. The distribution can be either provided directly or computed using additional context information not explicitly considered in the transition and observation model. The context variables, such as age, sex, or smoking history are used to estimate prior distributions more accurately.

Once the prior model is defined we can compute the initial probability of a patient having coronary artery disease and ischemia of various severities, based on initial findings, observations and relevant prior information. This in turn allows us to compute and answer other probablistic queries, and to predict the next patient state for a specific intervention.

In the current version of the IHD model we assume that the probability distribution for the *prior coronary artery disease* variable is given directly and no context information is assumed. However this is relatively easy to change and one can use some simple model to compute new priors, e.g. a logistic regression model from [Anderson et al. 90].

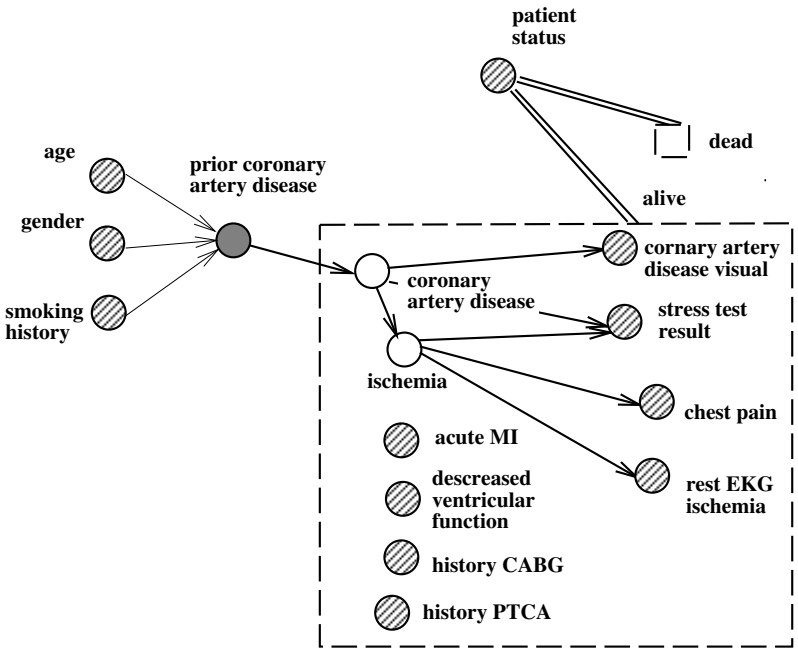


Figure 6-5: Ischemic heart disease: prior model.

Inconsistency of prior and transition models

The prior model makes it possible to compute the probability distribution for hidden states given all current observations as well as all prior information. In principle one could apply the same prior model to determine the probability distribution for the next state. However this approach differs significantly from what we do when we update the current state using the transition model.

Changing models in this way introduces the tricky issue of model consistency. Assume a patient with specific context information and an initial set of observations is given no treatment and at the follow-up visit the patient has the same set of observations and context. Naturally if we use the same prior model we will get the same answer for all hidden variables. However if we use the prior model for the initial state and the transition model subsequently for the following steps, the resulting probability distribution will usually differ. This causes us to question the consistency of these two approaches and whether our belief about the patient's state should be the same in both the initial state and at the next sequential state.

The answer is straightforward. Using the prior model sequentially over and over again ignores the information obtained in previous steps that consist of a history of observations and actions (or complete information state). A better approach employs the transition model and takes into account all available information. Thus information available initially is considered different from information used later and so, we do not get the same answer for initial and sequential sitations despite the fact that all context and current information may be the same. This also means that no consistency enforcement between the prior and transition models should be done.

6.2.5 Cost model

The cost model in a POMDP describes payoffs associated with possible state transitions. For example in managing ischemic heart disease we associate the highest cost with transitions to the dead state, smaller but still substantial cost with occurrences of MI, and severe chest pain.

In a POMDP costs and rewards are associated with possible transitions and different costs can be defined for every transition between any two states and an action. However, it is often reasonable to assume the cost model has more structure. In the IHD problem we propose a cost model that consists of two components:

- a cost associated only with the resulting state. This cost is independent of the initial state as well as any action performed in that state, for example there is a cost associated with a live patient that is suffering from severe chest pain and has experienced an acute MI.
- a cost associated with an action, regardless of initial and resulting states. For example there is a cost associated with performing coronary bypass surgery that includes the economic cost, patient's suffering, discomfort, and so forth.

In such a case the transition cost from state s to state s' given action a can be expressed as:

$$\rho(s, a, s') = \rho(s') + \rho(a)$$

The cost associated with a state that results from a transition can be further broken down into component state variable costs using an assumption of cost independence. A cost associated with a compound state of being alive, having an acute MI, suffering from severe chest pain and having moderate coronary artery disease, can be expressed using a cost model that adds up cost contributions from severe chest pain, acute MI, moderate coronary artery disease and the other state variables. This can be determined as follows:

$$\rho(s) = \sum_i \rho(s_i).$$

where s_i is the state variable assignment to variable i.

The fact that the cost model decomposes into atomic costs associated with state variables and actions significantly reduces the number of parameters we need to define. This in turn simplifies the stage of building a POMDP model in which quantitative cost estimates need to be found and assesed. Once these estimates are collected we use a transition model to compute the expected one step cost associated with action a and a state configuration s as:

$$\rho(s, a) = \sum_{s'} p(s'|s, a)\rho(s, a, s') = \rho(a) + \sum_{s'} p(s'|s, a)\rho(s')$$

6.2.6 Discretizing time

The POMDP framework, like many other frameworks, models continuous time through discretization. In the IHD problem it is assumed that every action is associated with a fixed time duration and that any change in state occurs between the discretized time points. The chosen duration of transitions strongly influences transition probabilities. For example, the probability that a patient will die as a consequence of not treating severe coronary obstruction is higher for a one year period than for a three month period.

In our IHD model we assume that transitions associated with invasive actions occur within a day, and transitions associated with non-invasive actions (such as *no-action* and *medication* treatment) are within 3 months. These durations are also reflected in the transition probabilities representing rates of state variable changes.

6.2.7 Modeling the objective function

The treatment objective is to find the action or the sequence of actions that minimize the expected cost with regard to the chosen decision model. The typical decision models one can use in the IHD case include both the finite horizon criterion in which one optimizes the treatment with regard to the next n time steps, and the infinite discounted horizon criterion which combines costs over an infinite number of time steps, with heavier discounting on the more distant future.

In our work we use the infinite discounted horizon model. This allows us to express longer term goals and not restrict the decision horizon to a finite number of steps. An interesting feature of this model is that we use discounting ($\gamma = 0.95$) only for long-term actions (*no-action* and *medication*). All other short-term actions are undiscounted and their costs are added fully within the model. Using two different discounts accounts for different action durations.

The important issue from the point of view of action selection is that the information-state at any point in time can be sufficiently modeled by a belief state that assigns a probability to every possible process state. The importance of this stems from the fact that the solution in this case is known to satisfy some nice properties, namely the value function is piecewise linear and concave. This knowledge allowed us to use better exact and approximation methods.

6.3 Acquisition of model parameters

One of the important problems associated with the ischemic heart disease model is to obtain a set of appropriate model parameters. The parameters define either probabilities or costs associated with state outcomes and actions. In general these can be obtained by:

- acquiring them directly from the domain expert or from the literature;
- infering them from the available data;
- or by using the combination of these two methods.

Although there are some studies with possibly useful datasets we were not able to obtain them for various proprietary and technical reasons. This left us with the choice of defining the parameters by hand using the published results or utilizing the experience of a cardiologist. We primarily relied on Wong [Wong et al. 90] that summarizes various studies in the area of chronic ischemic heart disease and compares outcomes for various interventions. In addition, one of our collegues, Dr. Hamish Fraser, helped us to interpret some of the available data.

6.3.1 Acquisition of transition and observation probabilities

To populate probabilistic transition and observation models we had to acquire parameters for all conditional distributions defined by parents-child variable combinations in the Bayesian network in figure 6-4. The total number of parameters one has to define for the model is 1171 (recall that some parameters can be infered because probability of all possible instances should sum to 1). This is significantly less compared to the case with flat state and observation spaces and

complete transition and observation matrices with 1127520 parameters (the number assumes that the process state space and the observation space are separate). Note that the number of parameters can be decreased by taking into account further structural features (e.g. partial conditional independencies).

The conditional probabilities for transitions can be obtained or infered from the results of clinical studies. For example the probability of the patient staying alive or dying as a result of a surgical intervention can be estimated from mortality rates for a specific treatment and specific patient condition. Similarly one can obtain numbers for other parameters. For example, numbers reflecting the rate of change of the coronary disease under different interventions can be obtained from the published success rates of revascularization for PTCA and CABG. Unfortunately, in many cases the results of studies are presented independently for one or a few conditioning variables, leaving open the problem of how to deal with various combinations. In such cases we either assumed independence, when it seemed reasonable, or adjusted probabilities by consulting Dr. Fraser. In general the process of defining probability parameters turned out to be very tedious and time consuming. We believe that the availability of datasets would simplify the acquisition process and would lead to more accurate parameter estimates.

Table 6-1 shows the parameters of the local probability table used to define the distribution of the coronary artery disease in the IHD model. The parameters represent transition rates for a 3 month period for *no-action* and *medication* choices. The parameters for other actions reflect the success rate of coronary artery disease repair. The probablity parameters shown were obtained and modified for our model based on results and success rates published in [Wong et al. 90]. The model at this stage does not distinguish between left main stem and multiple vessel coronary artery disease and combines them into the severe coronary artery disease category. This leads to similar success rates for CABG and PTCA procedures for severe coronary artery disease.

6.3.2 Acquiring cost model parameters

While probabilities can in principle be learned from an available dataset, costs reflect a combination of preferences of a physician, patient etc. This makes them more subjective and usually not mineable in the datasets. In order to acquire costs for the ischemic heart disease model we have designed an acquistion method based on the *cost distribution model*. The method can be applied directly to the hierarchically structured state variable set in the IHD model.

The main idea of the approach is to describe the distribution of costs among hierarchically structured components of the IHD model (like state variables, state variable values and actions). The cost model uses a local weighting scheme that describes the amount of cost a lower level component acquires from the higher level component. The cost associated with a lower level component is computed as:

$$Cost_i = w_i Cost,$$

where $Cost$ represents a cost quantity to be distributed and w_i is a weight associated with a lower level component that satisfies $0 \leq w_i \leq 1$ and that describes a share of the cost inherited by the component. There are two types of local models, that either restrict or do not restrict values of the component weights w_i:

- *and model*, where weights associated with lower level components (corresponding to the same higher level component) are complementary and must satisfy: $\sum_i w_i = 1$, where i ranges over all lower level components.
- *xor model*, where weights associated with lower level components are unrestricted and components are treated independently.

action	history of procedures	previous coronary artery disease	coronary artery disease		
			normal	mild-moderate	severe
no action	PTCA any CABG	normal	0.945	0.047	0.008
		mild-moderate	0.001	0.944	0.055
		severe	0.0	0.001	0.999
	CABG no PTCA	normal	0.955	0.037	0.008
		mild-moderate	0.001	0.957	0.042
		severe	0.0	0.001	0.999
	no CABG no PTCA	normal	0.99	0.0085	0.0015
		mild-moderate	0.0001	0.9799	0.02
		severe	0.0	0.002	0.998
medication	PTCA any CABG	normal	0.945	0.047	0.008
		mild-moderate	0.001	0.944	0.055
		severe	0.0	0.001	0.999
	CABG no PTCA	normal	0.955	0.037	0.008
		mild-moderate	0.001	0.957	0.042
		severe	0.0	0.001	0.999
	no CABG no PTCA	normal	0.99	0.0085	0.0015
		mild-moderate	0.0001	0.9799	0.02
		severe	0.0	0.002	0.998
angiogram	-	normal	1	0	0
		mild-moderate	0	1	0
		severe	0	0	1
stress test	-	normal	1	0	0
		mild-moderate	0	1	0
		severe	0	0	1
PTCA	-	normal	1	0	0
		mild-moderate	0.87	0.13	0
		severe	0.74	0.15	0.11
CABG	-	normal	0.9	0.05	0.05
		mild-moderate	0.82	0.12	0.06
		severe	0.75	0.15	0.1

Table 6-1: Local probability table for the severity of the coronary artery disease, given the action, history of previous procedures and severity of the coronary artery disease in the previous time step. The transition probabilities for long-term actions (*no-action*, *medication*) are defined for a 3 month period.

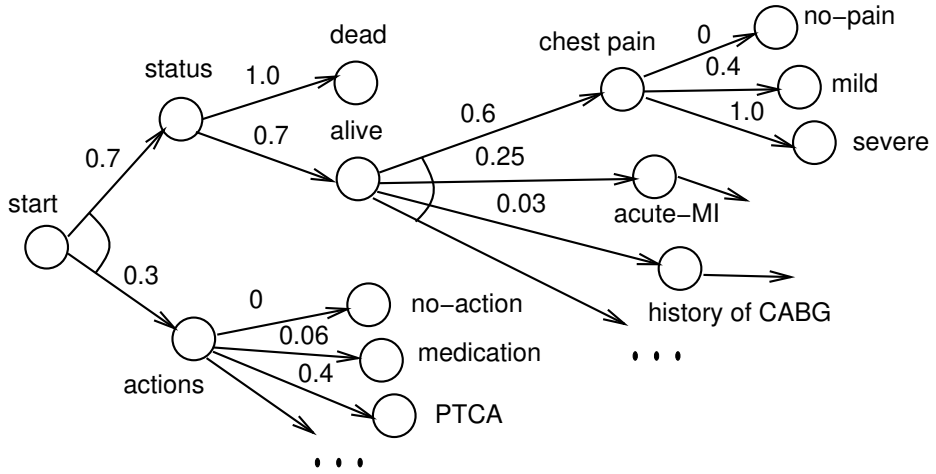


Figure 6-6: Model used for the acqusition of costs for the ischemic heart disease model. Links with arcs represent the *and* cost distribution model, links without arcs represent the *xor model*. Numbers represent weights assigned to the lower level components.

A graph representing a part of the cost distribution model for the IHD problem is shown in figure 6-6. The *and model* is represented by a parent-children combinations with an arc on the outgoing links and is used to distribute costs among complementary components. For example *status alive* is elaborated as lower level components *chest pain, acute-MI, history-CABG*, and so on, that are complementary and each is assigned a portion of the cost accounted for by *status alive*. On the other hand the *xor model* is represented by a parent-children subgraph without an arc and is used to distribute costs to components that are exclusive. For example, the patient's *status* can be either *dead* or *alive* and the cost model defines how much of the overall cost assigned to the patient state is accounted for by each alternative.

The advantage of the cost definition model is that it requires the physician to define only local cost distributions (weights). This simplifies significantly the whole acquisition process. For example we can ask the expert to quantify the cost associated with different severities of chest pain on scale $[0, 1]$ (or any scale as we can always renormalize the input), or ask the expert to quantify the importance of chest pain, acute MI, decreased ventricular function and so on with regard to the cost. The numbers shown in graph 6-6 corresponds to the weights we use for the IHD problem.

Once the cost distribution model is completely defined, we can compute the cost associated with a specific low level component. This is done by multiplying weights associated with the links one needs to traverse to get from the root of the distibution structure to the particular leaf node. For example, the cost for the severe chest pain is computed as:

$$Cost_{\text{chest-pain-severe}} = Cost_{\text{initial}} \times w_{\text{status}} \times w_{\text{alive}} \times w_{\text{chest-pain}} \times w_{\text{chest-pain-severe}}$$

where $Cost_{\text{initial}}$ is the cost we expect to distribute and the ws stand for weights associated with different links. The $Cost_{\text{initial}}$ value was set to 100 cost units for the IHD problem.

6.4 Finding control policy for the IHD domain

6.4.1 Representing information state

For the standard POMDP models, the information state space corresponds to the belief space that assigns the probability to every possible process state. However, this assumes that the process state is always hidden (partially observable). Contrary to this, the proposed IHD model uses process states that are heterogeneous and can have both perfectly and partially observable components (state variables). In fact the presence of perfectly observable process state variables simplifies the problem, as one can directly incorporate the observed state variable values into the information state. This reduces the size of the information belief space one needs to represent as it is defined only over all possible combinations of hidden variable values. Thus an information state for the factored state model with both perfectly and partially observable process variables can be represented using:

- a set of observable process state variable values;
- a belief over the combination of all hidden variable values.

The information state for the IHD problem consists of an assignment of values to observable process state variables, for example *status alive, acute-MI true, history-CABG false, history-PTCA false*, and a belief over all possible combinations of values for *ischemia* and *coronary artery disease*. Note that the information state for the case when the patient is dead is described only as *status dead*.

The information state space for the IHD model can be represented using a tree structure in figure 6-7. Internal nodes correspond to observable variables, subtrees of an internal node to assignments of values to the associated observable variable and leaf nodes to belief space over hidden variable values. Then, every branch of the tree represents one possible assignement of values to observable variables. Note the asymmetry due to hierarchical state variable space.

Savings from the additional structure

The information state space for the structured IHD model uses 17 possible combinations of observable variable values: one for *status dead* and 16 for the alive state (four binary variables). All *status alive* combinations require an additional 9-dimensional belief space (all possible combinations of values for *coronary artery disease* and *ischemia* variables).

The proposed factored and hierarchically structured IHD model reduces the complexity of the information state one needs to work with. To illustrate this, let us assume a flat process state space. Such a space does not allow to combine observable and hidden components and consists of 145 states (this figure counts only state variable combinations that are possible). As states are now assumed to be hidden, the information state space corresponds to a 145-dimensional belief space. The incorporation of observable variables thus reduces the complexity of the high dimensional belief state space to a set of belief spaces of small dimension. The overall number of observations is 1153 (all possible combinations of observable variable values) and it is same for both cases.

The above analysis illustrates savings from the model factorization and combination of observable and hidden process state variables. However, there are also savings that can be attributed to a hierarchy of state variables. Assume we use the flat state variable space, that is values for every state variable can be combined without restrictions. This would lead to the information state space with 32 combinations of observable variable values, compared to the hierarchical state variable space that allows only 17 different combinations.

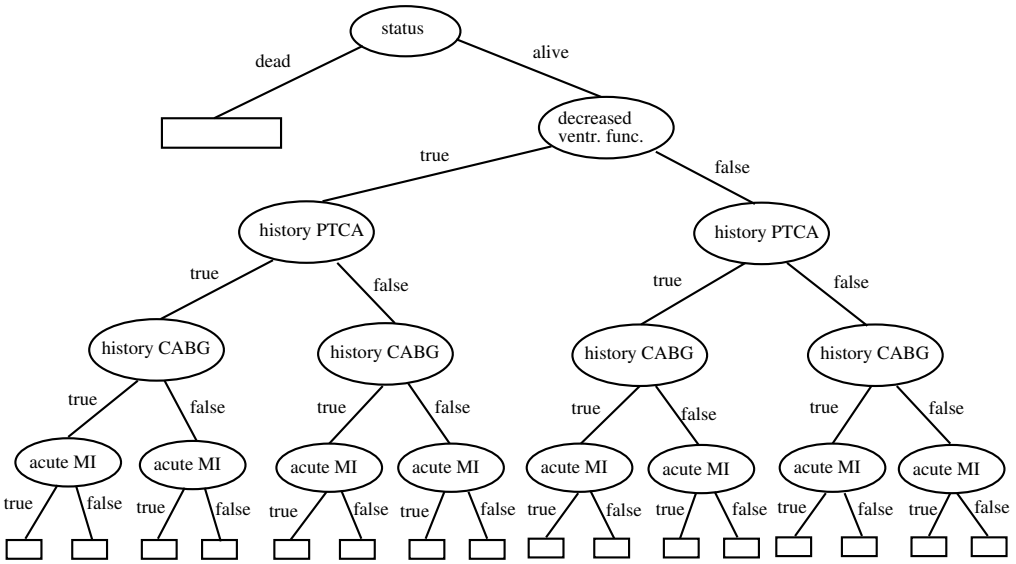


Figure 6-7: Information state space for the IHD problem represented by a tree structure. Internal nodes correspond to process state observables and leaf nodes to beliefs over all possible assignments to hidden process state variables.

6.4.2 Compilation of the model

Although the graphical model represenation in figure 6-4 allows us to represent the IHD model more compactly, it is not always useful when certain queries need to be computed repeatedly. Thus, instead of working with the original model we have decided to compile the model so that repeated queries can be obtained fast. The compiled model is represented as a decision tree with internal nodes corresponding to observable state variables or actions and leaf nodes containing conditional probability distributions over hidden variables. The constructed tree is optimized also by pruning branches corresponding to 0 probability situations. Such a decision tree model is then used to compute all probabilistic queries neccessary for the planning and control tasks.

6.4.3 Solving control problem

The objective of the control problem is to find the optimal policy over the information space. As this problem is often hard to compute one can turn to approximations that can produce good solutions with less computation. These were presented in previous chapters and can be modified and reimplemented to handle a hybrid two-component information state space.

From the methods we have implemented and tested on maze and the Shuttle docking problems we have reimplemented four value function approximation methods: MDP-based approximation, blind-policy method, the incremental linear vector method and the fast informed bound method. Incremental linear vector method and fast informed bound methods were also among the top three performers on the test problem set from the previous chapter.

New versions of approximation methods use compiled transition and observation models and

work with value functions that are defined over the hybrid information space. A value function consists of discrete and belief space components and can be represented using the same tree structure as shown in figure 6-7. The value function for each belief space component (one for each combination of the observable state variable values) is represented by a piecewise linear and concave function (concaveness is due to minimization). In other words the value functions consist of a set of piecewise linear and concave value functions (defined over belief spaces) that cover all possible combinations of observable process state variable values.

All methods described and tested for the standard POMDP can be modified with more or less effort to handle new information state space. To illustrate the idea of such modifications the new version of the incremental linear vector method will be presented and described next.

Incremental linear vector method

The incremental linear vector method from section 4.8.2 can be reimplemented for the new information state space using the following update procedure.

Incremental linear vector update $(\widehat{V}, k)$
  for every combination o of observable process state variable values
    do if no hidden variables are associated with o in the information state space
      then update value function $\widehat{V}$ for o using standard value function value update;
      else let B_o be a belief space associated with observable component o and
          Γ_o be a set of linear vectors defining $\widehat{V}$ for B_o;
        select $[k \dim(B_o)]$ belief points G from B_o;
        for every belief point $b \in G$
          compute new linear vector α^b for b using Sondik's update;
          add α^b to Γ_o in $\widehat{V}$;
return $\widehat{V}$;

Assuming that we want to solve the cost minimization problem, the procedure takes an upper bound value function $\widehat{V}$ defined over the hybrid information state space and parameter k that allows us to vary the number of belief points to be updated with Sondik's method in every component belief space. The procedure returns a new improved upper bound value function and thus can be repeatedly applied to tighten the upper bound.

We assume that $\dim(B_o)$ defines the dimension of the belief space B_o corresponding to the combination of observable process state variable values o. Then the number of belief points from B_o updated by the above procedure is $k\dim(B_o)$. Of course, other strategies to control the number of belief points updated in every belief space are possible as well.

Belief points to be updated can be selected using arbitrary point selection strategies, similarly to the standard POMDP case. Note that when a combination of observables o does not permit any hidden variables (e.g. for the *status dead*) the value function is updated for a given combination of process state observables o directly and only once.

Solutions used for testing the model

For the testing (see next section) we used solutions obtained by the incremental linear vector method and the fast informed bound method. The incremental linear vector method used 15 incremental linear vector updates (see above procedure) with parameter $k = 2$. A set of belief points updated in every belief space consisted of all critical points and the rest of points was

selected randomly. The computation took about 30 minutes on SPARC-10 in Lucid Common Lisp. The solution for the fast informed bound method was obtained in about 3 minutes.

6.5 Evaluating the model

In our work we constructed a prototype IHD model of significant complexity. Interestingly, despite model simplifications and the need to estimate a large number of parameters we were able to achieve the behavior that was for many cases clinically reasonable and justifiable. This is very promising for the future work and further extension and refinement of the model.

6.5.1 Testing obtained policy for the patient follow-up case

The construction of complex models is usually not a one shot activity and requires few iterations to clear various bugs and bad parameter assignments. However, we were surprised that we were able to acquire many clinically accurate recommendations for the approximate solution practically from the beginning. Thus, after a few iterations we were able to observe the reasonable decision behavior of the model in many instances for both initial and patient follow-up situations.

Table 6-2 illustrates a sequence of recommendations obtained for a single patient case (including follow-ups). The value function used to compute recommendations has been obtained by the incremental linear vector method (see previous section). For every stage, the table shows a list of actions, ordered with regard to the obtained cost score. The top (lowest cost) action is executed at each step. The second score represents a lower bound on the optimal expected cost computed by the fast informed bound method. Interestingly, if we use the second score as a basis for the recommendation the choices will be exactly the same. This is enouraging because these are methods that achieved the best control performance in the experiments presented in the previous chapter.

6.5.2 Alternative decision choices

More important than the simple ordering of actions based on absolute values is often a relative comparison of alternatives with regard to the leading choice. These differences turned out to be relatively small for the evaluated patient case, with the exception of both PTCA choices. However, comparing all candidate choices, it is clear that choices with similar scores are often not very far apart in terms of costs or similar effects. Thus, the action list does not look bad from the point of relative scores. The only action that is clearly suboptimal is the coronary bypass surgery (CABG).

Sensitivity of the model to parameter choices

The comparison of relative scores and small score differences between actions also opens the question of model sensitivity to parameter changes. It is clear that for some of the instances one should be able to cause the change of leading actions relatively easily by changing some of the cost or probabilistic parameters. Thus the model and policy for the tested region are sensitive to these parameters.

step	current patient status		actions	used cost score (upper bound)	cost score (lower bound)
0	chest pain:	mild-moderate	**stress-test**	285.22	248.53
	rest EKG ischemia:	negative	no action	285.62	249.82
	decreased ventr. func.:	false	medication	286.75	250.98
	acute MI:	false	PTCA	288.75	252.36
	coronary artery visual:	not available	angiogram	292.92	256.68
	stress test results:	not available	CABG	491.94	427.77
	history CABG:	false			
	history PTCA:	false			
1	chest pain:	mild-moderate	**PTCA**	298.47	262.54
	rest EKG ischemia:	negative	stress test	316.39	280.33
	decreased ventr. func.:	false	no action	321.92	288.24
	acute MI:	false	medication	322.72	289.12
	coronary artery visual:	not-available	angiogram	323.79	287.91
	stress test results:	positive	CABG	503.73	440.77
	history CABG:	false			
	history PTCA:	false			
2	chest pain:	no chest pain	**no action**	259.07	226.23
	rest EKG ischemia:	negative	medication	260.62	227.78
	decreased ventr. func.:	false	stress test	264.35	229.87
	acute MI:	false	angiogram	273.34	239.16
	coronary artery visual:	normal	PTCA	276.98	243.24
	stress test results:	not available	CABG	481.36	417.28
	history CABG:	false			
	history PTCA:	true			
3	chest pain:	mild-moderate	**medication**	451.50	418.07
	rest EKG ischemia:	negative	no action	452.81	419.47
	decreased ventr. func.:	false	PTCA	464.58	429.87
	acute MI:	true	angiogram	470.62	435.62
	coronary artery visual:	not available	stress-test	479.68	445.22
	stress test results:	not available	CABG	657.77	608.11
	history CABG:	false			
	history PTCA:	true			
4	chest pain:	mild-moderate	**PTCA**	471.16	433.98
	rest EKG ischemia:	negative	medication	483.11	447.85
	decreased ventr. func.:	true	no action	485.15	450.04
	acute MI:	false	stress-test	486.32	448.75
	coronary artery visual:	not available	angiogram	496.38	458.87
	stress test results:	not available	CABG	661.98	610.81
	history CABG:	false			
	history PTCA:	true			

Table 6-2: Patient case with followup. Recommendations are based on the value function approximation computed by the incremental linear vector method (upper bound). The lower bound cost score is obtained using the fast informed bound method.

6.5.3 Problems with the current model

The constructed ischemic heart disease model and computed policy solution demonstrated that they can be a source of clinically acceptable decisions for many situations. However, there were also situations in which decisions proposed seem to be unreasonable and did not match the standard clinical practice. These are mostly due to:

- model simplifications;
- subjective parameter estimates.

Model simplifications

An example of the situation when model is not sufficient is the following scenario:

- The patient presents with a mild-moderate chest pain. No other tests are positive. The recommended action is a stress test that is expected to produce more information about the underlying status of patient's coronary arteries.
- Unfortunately one of the outcomes of the stress test is non diagnostic. This corresponds to the situation when the patient fails the test due to his/her poor physical condition (the achieved level of exercise is not sufficient to make the positive or negative conclusion). Assuming that the patient failed the test the belief about the underlying *coronary artery disease* and *ischemia* level will not change very much.
- The action chosen is stress test again.

This is clearly an example of a case in which the model is oversimplified, as it is very likely that the patient will fail the test again. The problem is that the model does not represent and differentiate between circumstances when patient is more likely to pass or fail the test. The decision to recommend stress test again is based on the available stochastic model that models different test outcomes randomly with higher probability being assigned (based on population study) to the diagnostic outcome. Thus the repeated decision choice simply reflects the fact that it is worthwhile to flip the coin with larger probability of successful outcome again.

The simplest fix to the above problem is to add a new state variable *physical state* that would represent the physical state of the patient. The patient with a poor physical state is then likely to fail the stress test. The failure of the test in the first trial will lead to the assessment of the patient's poor physical state and prevents the stress test from being selected again in the next step.

Many current model simplifications could be fixed by adding new state variables and thus using more detailed process states. Unfortunately such changes make the model more complex and harder to solve. Therefore one needs to carefully decide how to refine the model and what details to elaborate more. It is also likely that in order to maintain practical solvability of the problem larger model refinements that represent more of the domain detail will require new approximation techniques (e.g. based on abstractions) and/or further exploitation of the underlying structure. We believe that with the current techniques we will be able to handle reasonably well the model with two or three additional observable state variables (binary variables) or one hidden process state variable (with two or three values).

Subjective parameter estimates

The other problem that complicates the matter in the ischemic heart disease domain is the problem of subjective cost estimates, that reflect the preferences of the physician and the

patient. In many cases it is really hard to say how to penalize e.g. death and how this compares to the heart attack in terms of the cost score. This uncertainty in preferences can lead to situations where cost quantities assigned to some scenario, although reasonable and justifiable on paper, do not lead to decisions seen in practice. This documents how hard it is to assign subjective preferences correctly.

On the other hand this may also mean that people in their decisions may be driven by observing specific patterns and applying learned (associated) actions rather than evaluating possible choices and their consequences appropriately. This can even lead to situations in which suboptimal decisions are considered to be standard. The use of decision analytic models and techniques that are based on well defined clinical studies could help us to acquire new insights and could potentially lead to change in standards.

6.6 Summary

POMDPs provide a suitable modelling framework for representing and solving complex treatment planning and decision problems in medicine. However, the application of the framework to medical or other real world applications also carries additional challenges one does not have to consider while solving toy world planning and control problems. These are related to:

- the representation of the model structure;
- the acquisition of model parameters (probabilities and costs)
- handling actions with different time durations.

Contributions

The major contribution of our work is in extending basic POMDP framework to model and exploit additional domain structure. The main new ideas include:

- combination of MDP and POMDP models (process state is described using both perfectly and partially observable components);
- hierarchical state variables space (cuts down the size of the process state space by excluding redundant or impossible state variable value combinations)

These extensions allow us to reduce the complexity of the information state space we use in computing the control task. The reductions are achieved by using a simpler two-component information state that consists of an assignment of values to observable process state variables and a belief over all possible assignments to hidden state variables. The information states can be heterogeneous, that is the size and the content of the information state can vary.

Other new ideas presented in this chapter include:

- Hierarchical Bayesian belief networks for representing transition and observation models. These capture more of the structure of the model, reduce the number of parameters the model uses and thus simplify their definition.
- Factored cost model that divides a cost into components: a cost associated with an action and a cost associated with state variable values that result from the action. Such a model requires fewer parameters to be defined. The definition of cost parameters was further simplified using the proposed cost distribution model.

- Actions of different durations with different discount factors.
- Compilation of the transition and observation model that speeds up probabilistic queries and eliminates zero probability transitions.

Using all new features discussed above we were able to define an IHD model of significant complexity and compute value function approximations for the treatment policy problem in reasonable time. These functions were in turn used to obtain treatment choices for different patient scenarios and follow-up situations. Although the model used needs to be further improved and refined, it demonstrated the capability to compute clinically correct choices in many situations. This helped us establish the link between models of system (disease) dynamics and goal preferences, and clinically correct decisions.

The current IHD model needs to be improved and refined in many places. It is also likely that in order to maintain practical solvability of the problem larger model changes and refinements that represent more of the domain detail will require alternative approximation techniques (e.g. based on model reductions) and/or further exploitation of the underlying structure.

Chapter 7

Conclusion

The POMDP framework is suitable for modelling dynamic decision or control proceses with stochastic behavior and with partial (imperfect) information about the underlying process state. The framework offers increased expressivity compared to the MDP that assumes perfect observability of the process state. Thus the main distinguishing features of the POMDP framework are: process states are observed indirectly through a set of observations, and observations can be conditioned on investigative actions.

7.1 Solving the POMDP problem

The price paid for the increased modelling power of POMDP framework is high, and causes the significant increase in the computational complexity of exact algorithms producing optimal or near optimal solutions. This makes the framework and associated algorithms often practically applicable only in domains with a relatively small number of states, actions and observations.

7.1.1 POMDP exact methods

The partial observability hits policy problems, that require one to find the control for all possible information state especially hard. Problem solving methods are based on the dynamic programming or value iteration but are subject to the exponential growth of the value function description. Moreover existing algorithms are inefficient also with regard to the computation of the value function update. This causes only problems of small complexity, that include not more than 10 states, to be practically solvable. Moreover all of the known optimization algorithms exploit the feature of piecewise linearity and convexness of the value function that holds only for belief space POMDPs and thus we do not know how to compute exactly the policy problem for more complex POMDP models, e.g. with delayed observations.

Relatively faster, but still subject to the exponential growth are algorithms that generate optimal or near-optimal response for the current information state in the forward fashion using decision trees. As the decision tree needed to make the decision can grow large (infinite for infinite horizon problems) "intelligent" methods that attempt to build (expand) the tree gradually and prune suboptimal branches whenever possible can be designed. The pruning can be performed based on value function bounds. The advantage of the decision tree method is that it can be applied also for finding the best action not only for belief space POMDPs but also for POMDPs with delayed observations. Incremental forward methods for finding optimal

or near-optimal decisions can be turned into anytime procedures generating control responses that improve gradually over time.

7.1.2 POMDP approximations

The natural solution for the problem of computational complexity is to trade off the solution accuracy for the speed. This leads to methods that try to come up with a good solution efficiently. Most of the approximation methods are based on the approximation of value functions or model reduction techniques. Possible value function approximation methods are: the MDP based approximation, blind policies, fast informed bound update, grid-based interpolation-extrapolation, grid-based linear vector method, curve fitting. On the other hand model reduction techniques are based mostly on the feature-based approaches that reduce the information-state MDP corresponding to the POMDP.

Although there is a relatively large spectrum of approximation methods that alllow us to solve the optimization problem efficiently, there is not a very good understanding of what makes various approximation methods better or helps us determine what methods are more promising and what are inferior. This is caused to a great extent by the lack of larger scale experimental studies that would give us a ground for the larger evaluation and comparison.

7.1.3 Extensions of the POMDP framework

The main advantage of the POMDP framework over alternatives is in its capability to model stochastic partially observable control processes. However this does not mean that we will be able to capture all features of real-world domains using the basic POMDP formalism. In fact, dealing with real world domains, one can often take advantage of additional problem structure that is not expressed in the basic POMDP model and use it to speed-up the problem-solving routines. Thus, the exploitation of the additional problem structure offers another solution for the problem of computational complexity of the exact POMDP methods. For example, dynamic processes are not often completely hidden and what occurs is usually a combination of perfectly and partially observable state components. Then a framework that combines and exploits advanatges of both MDPs and POMDPs offers better solution. This was shown for example on the ischemic heart disease problem in the previous chapter.

7.2 Contributions

Our research work has focused on the following goals:

- the design of new exact and approximation methods;
- the comparison, test and analysis of value function approximation methods;
- extensions of the basic POMDP framework, exploitation of the additional problem structure.

Although the main contributions of our work fall into the above categories, we believe that the text as a whole can serve as a good reference for people exploring the area of planning under uncertainty. Also the work describes some of the new and promising ideas we were not able to pursue or describe in depth due to time constraints, and thus it provides a source for interesting research topics. In the following we will summarize the contributions of the thesis along the outlined main objectives.

7.2.1 Exact POMDP methods

Belief space POMDPs

The standard POMDP model assumes that observations always depend on the actual process states and previous actions. Such a model can be converted to information-state MDP with sufficient information states that correspond to belief states, and with value functions that have been shown to be piecewise linear and convex [Smallwood, Sondik 73]. However, there are other models (e.g. model with backward triggered observations or combination of backward and forward triggered observations) that can be converted to information-state MDPs with belief states. We have shown that the Sondik's result of piecewise linearity and convexness not only applies to the standard model but can be extended to the set of belief space POMDPs. This allows us to use exact algorithms developed for the standard model for any belief space POMDP.

Gauss-Seidel speedup of value iteration

Probably the most important contribution of our work for exact methods is the idea of Gauss-Seidel speedups of the value iteration algorithm for the belief space POMDPs and infinite discounted horizon problems. The method uses lower bound piecewise linear value functions and improves value function incrementally by computing and adding new linear vectors obtained for points of the belief space to the previous solution. Every new linear vector obtained is immediately used to compute further updates. The main advantage of the incremental scheme is that it avoids the recomputation of the complete value function from scratch.

Speedups of exact updates

Another interesting part, is the work on the improvement of exact Monahan's algorithm using incremental schemes that enable us to interleave the construction and test phases of the useful linear vector set and employ an early pruning of redundant partially built linear vectors. This topic has been investigated recently by [Cassandra et al. 97]. However, the methods developed there can be applied to build action-value functions (Q-functions) and do not allow one to do early pruning accross actions. We have suggested an extension that makes it possible to apply the idea of early pruning across actions as well. The extension is based on computing bounds.

Forward decision methods

Most of the attention of researchers in the area has been devoted to the problem of finding the optimal policy. However, in many cases a far simpler decision problem that tries to select a control response for a single initial state can be sufficient for implementing the control agent. Such problems can be solved in the forward fashion by a process that incrementally expands the decision tree. In our work we have proposed, designed and implemented various incremental algorithms for solving such problems: breadth first, bound span heuristic, randomized heuristic, and linear space. These methods reduce the growth of the decision tree via pruning based on value function bounds.

In general the quality of bounds computed by the incremental forward algorithms depend both on the depth of the decision tree and on the quality of value function bounds used at the leaves of the tree. Thus one can tighten the bounds by either further expansion of the tree or by the improvement of bounds used at leaves. We have suggested a new decision method that

combines advantages of both forward decision methods and bound improvement steps using a metalevel decision procedure.

7.2.2 POMDP approximation methods

The fact that POMDP problems cannot be solved efficiently naturally leads to the usage of various approximation methods, that trade off precision for speed. Although many of the approximation methods have been known for some time, it is still possible to find new ones or suggest promising modifications of the existing ones. In our work we have suggested a few of these.

Fast informed bound

The fast informed bound method is a newly-designed method, that uses an efficient update scheme and upper bounds the exact update rule. The rule approximates a value function using piecewise linear and convex approximation with at most $|A|$ linear vectors. The main advantages of the method are its simplicity (it updates linear vectors directly), bound property and convergence. This is unlike the Q-function least square fit method that also uses $|A|$ linear vectors, but must update the value function at some number of sample belief points first; it does not bound the exact update, and it is not guaranteed to converge.

Variable grid point interpolation scheme

One of the existing methods for approximating value functions uses a grid of points, their values and the interpolation-extrapolation rule for approximating values at nongrid points. Interesting interpolation-extrapolation rules are based on point interpolation techniques. These lead to solutions that guarantee the upper bound as well as convergence for the belief state MDPs. The main problem with point interpolation rules is the selection of grid points relevant for interpolation. The techniques used to deal with this are based on regular grids that uniquely partition a belief space. In our work we propose a simple and efficient point interpolation scheme that can use aribtrary (variable) grids and preserves the upper bound property. This flexibility makes it possible to combine the method with various grid selection strategies, including various heuristics.

In connection with a new point interpolation method we have also proposed a new heuristic approach for constructing grids. The method uses a stochastic simulation idea to find grid points that are likely to maximize the improvement of the upper bound. Versions of the same method can also be applied together with other grid-based interpolation-extrapolation strategies, e.g. the nearest neighbor approach.

Incremental linear vector method

Yet another method for the value function approximation uses the refinement of the exact point-based linear vector updates to grids. The method computes a lower bound value function update, but does not guarantee the convergence. In our work we have proposed a new incremental linear vector method that updates linear vectors for a set of grid points and guarantees the convergence. The method starts from the initial piecewise linear and convex lower bound and gradually adds new linear vectors found for the grid points to the original function. The methods avoids costly rebuilding of the complete value function for every update and can be used to speed up the exact value iteration (see above).

7.2.3 Comparison and tests of approximations methods

There is a spectrum of approximations methods researchers have developed over the years. However, these were very often left uncompared and there is a lack of understanding of how various methods compare to each other and/or how various modifications can help to improve the basic methods. The methods can be compared theoretically and one can in many instances show that some value function method gives a better bound result than the other method, or that the method converges for the infinite discounted horizon case. However, it is very hard to say in general what the impact is of various heuristic improvements or how various approximation methods will perform with regard to control. These properties often need to be explored experimentally, and the lack of experimental studies that compare performances of a large number methods does not help in further endeavour.

Experimental study

New and existing value function approximation methods were tested and their results were compared using a set of three different infinite discounted horizon problems of various complexities. The experiments covered a large spectrum of possible value function approximation methods and their modifications that ranged from simple MDP-based approximations to least square fit methods and heuristic grid-based linear vector methods. The results thus provide the ground for their comparison and evaluation.

The experiments were conducted to explore the quality of value function bounds that are guaranteed by some of the methods, and the quality of control, where methods were judged solely based on the control performance on test problems. The results confirmed that for the purpose of control the best performance was achieved by methods that tend to approximate better the shape of the optimal value function. The best methods update value function derivatives and attempt to preserve the shape of the functions over many updates. Contrary to this, methods that used value functions that deviated from the piecewise linear and convex shape, like the grid-based nearest neighbour method, achieved inferior results and thus their usage is not warranted for the belief space POMDPs.

7.2.4 Extensions of the basic POMDP framework

The basic POMDP framework can be extended in many ways to better fit the features of the real world domains. For example the basic framework can be extended to deal with observation delays that are very important in modelling time critical control problems. Unfortunately in this case the original POMDP does not reduce to the belief state MDP and thus it remains closed to various exact and approximation methods that assume belief information states.

Although in some cases the extensions can make the control problem more complex, the basic framework can be modified and extended to take advantage of the additional problem structure and to use it to improve the problem solving routines. We have explored these ideas in connection with the POMDP application to the management of a patient with ischemic heart disease (IHD). The work on the IHD model lead to many new and very interesting extensions of the basic framework and we plan to explore them further in the future.

Combining MDPs and POMDPs using factored models

The basic POMDP framework assumes that process states are always hidden and information about the state can be acquired only through observations. However this is not always true, and one often works with process states that consist of both observable and hidden components. In

order to deal with this issue we propose to represent the POMDP model in the factored form with process states and observations represented using a set of state variables. Such variables can then be modelled as either observable or hidden. Moreover, probabilistic relations (transition and observation probabilities) can be expressed with regard to variables using graphical models and thus take advantage of independencies (conditional or unconditional) that hold among them. The factored model representation effectively allows us to combine MDP and POMDP formalisms into one frame work, and take advantage of both of them.

The factored model, with both observable and hidden process states, can be converted into the information space MDP, with information states that are composed of two components: a set of value assignements to observable state variables, and a belief state over all possible combinations of hidden state variable values. The usage and idea of two component information states that combine MDP and POMDPs frameworks is new and it has not been reported in the literature.

Heterogeneous information space

Although factored models can help to simplify the information state description, the information space they define can include information states that cannot occur in practice (contradictory variable value combinations, etc.). In order to reduce the complexity of the information state as much as possible, we have proposed the hierarchical version of the factored model in which some of the state variable values describe higher level concepts (abstractions) and subsume sets of other lower level state variables. The structuring allows one to describe possible states using descriptions of different complexity and size. Moreover the idea of hierarchical subsumption can be used to simplify the definition of the probabilistic relations by exploiting independencies that emerge on different levels of abstraction. With a hierarchical model, information states can be described using varying size components and thus information state space is heterogeneous. The idea of hierachical state spaces is also new and has not been used in the POMDP literature.

Other model improvements and extensions

Factored and hierarchically structured transition and observation models allow us to reduce the number of parameters defining the POMDP model. This is very important for the process of acquisition of the parameters both from the human expert or from the avaliable datasets using machine learning techniques. Similarly we proposed and used the factored cost model that is easy to define and uses small number of parameters.

Other model extensions and improvements we have proposed and used in our work include: handling actions with different time durations using different discount factors and compilation of the transition and observation models. The purpose of compilation was to acquire a model that would allow us to compute relevant probabilistic queries faster. To do this we have converted the model to the decision tree structure with internal nodes corresponding to observable variables and leaves corresponding to probabilities over hidden variables. Such a decision structure was further optimized by excluding zero probability contingencies.

7.2.5 Application of the POMDP framework

The newly extended POMDP framework has been applied to the problem of management of patients with chronic ischemic heart disease. The parameters of the underlying model were acquired based on published study results and subjective estimates. The model and solutions have been tested on few inital and follow-up scenarios. Despite some of the deficiencies (mostly

due to the model simplifications) we were able to observe reasonable therapeutical choices in many instances and these were in concordance with clinical practice.

The acquired result is important both from the perspective of the application area, and the framework. In the first case it gives us hope that we might be able to solve and analyse complex medical decision problems. In the second case it represents an example of a real-world application domain and thus helps to prove the relevance and place of the methodology in solving real-world problems.

7.3 Challenges and further research directions

There are many interesting problems that are crucial for further developments in POMDPs and their applications. Two of the most important are related to applications of the POMDP framework to large size domains and to learning of POMDP models from temporal datasets.

7.3.1 Attacking large problem domains

Despite efficient value function approximations, the standard POMDP framework (with hidden process states) is still suitable to handle problems of relatively small size (our best guess on the size of the problems would be around 100 states, but this can also vary with applications and their specificities). The problem of having large process states can be resolved when the underlying process state space consists of both observable and hidden components. Then new ideas and techniques developed in Chapter 6 that combine MDPs and POMDPs frameworks and use two component information states can be applied. However these techniques does not help to reduce the complexity introduced by hidden components (e.g. one still needs to work with belief states over all possible combinations of hidden variable values). Thus solutions for reducing the complexity associated with hidden process states are of our main interest.

Limitations of factored POMDPs

One of the approaches suggested for dealing with large models in the fully observable MDP framework was to rely more on the structure of the model. The approach works with a factored MDP model that captures independencies and regularities that hold among model components (represented using graphical models) and use these directly to find optimal or approximate solutions. Unfortunately, the planning methods that exploit factored models and underlying structural dependencies work fine for the MDP case mostly thanks to perfect observability. This is because all process state variables, once observed, make all past and future states independent of each other. Contrary to this in the POMDP case one works with information states, that are hard to break along the factored components. This is illustrated in the following: Assuming that all state variable values at current time are given (MDP case), all future instances of state variables (and their values) become independent of past state variables instances and their values. In graphical models language all future state variable values are d-separated from past state variables. However, not knowing values of the current state variables with certainty, future and past state variables are not independent of each other. Then for example, two observation variables in the future can become dependent, whenever both of them share a common hidden state variable in the past. Or in other words two observation variables that are d-separated by some hidden state variables of the Markov chain in the past can be dependent.

The major consequence of this is that the Markov property of information state process can be violated when one would use "factored" information states that are blindly related to

the underlying factored state representation. Of course, one can always apply the idea to the observable component of the process state whenever two component information states are used.

Model reduction techniques

The approach well suited for attacking large size POMDPs is based on model reduction techniques. We described it in section 4.10 but we did not explore it to the depth. The model reduction methods can target either information state MDPs or directly original POMDP models. The main idea is to merge states, observations or actions to aggregate entities and work with such aggregates. The typical representatives of such an approach are various feature extraction mappings [Tsitsiklis, Van Roy 96], or methods that work with truncated information histories [White, Scherer 94]. In the ideal case one would like to have techniques that can automatically select components of the model that can be aggregated and influence the solution to the smallest extent.

Open and challenging problems related to the model reduction idea include:

- techniques for finding appropriate aggregation methods (or feature extraction mappings) with the smallest effect on the resulting approximate control;
- the exploration of relations between reductions of the original POMDP model and an associated information-state MDP;
- the tradoff between model approximation and value function approximation approachess.

Speeding up high dimensional belief state updates

There are other approaches that can be used to speedup computations for the large POMDP problems. The fact that one needs to work with POMDPs with a large number of partially observable states causes a significant slow down of information state updates. This is also because one needs to work with high dimensional belief states that need to be updated often. One approach aimed to reduce the computational and space complexity associated with belief state updates is based on the idea of stochastic simulation (see [Schachter, Peot 89] [Kanazawa et al. 95]). The idea of the method is: assuming that one knows the current belief state then the next belief state approximation under action a and observation o can be computed using the following steps:

1. select k random world states based on the current belief state distribution;
2. simulate the transition for the selected state, action a and observation o via Monte Carlo method;
3. merge simulated results (from frequency count) and produce a new belief state.

The approximation of a belief state update is suitable when there is a small number of regions with higher probability (weight). Then one can approximate the probability distribution over belief space by considering and remembering only higher probability regions. There are other options for making the simulation work. In the outlined approach one needs to compute $P(s'|s,o,a)$ for all possible s' first, then to select next state via Monte Carlo simulation and after that to compute frequencies for all outcomes. However it is possible also to use the fact that $P(s'|s,o,a) = P(s'|s,a)P(o|s',a)$. Then one can select s' from $P(s'|a,s)$ by the simulation, give it weight of $P(o|s',a)$, and sum all results for the same s' to acquire the overall weight for s' (after normalization).

7.3.2 Learning in partially observable stochastic control domains

Most of the discussion related to the control in partially observable stochastic domains assumed that a POMDP model was always available, so the control agent or compiler could use it to compute the optimal or approximate control. This completely ignores problems associated with the acquisition of POMDP models, that can turn out to be hard task itself. For example the assignment of rewards or other parameters of the model must be done consistently and reflect intended preferences and/or objective frequencies. Therefore the possibility of learning underlying controller knowledge directly from observed control sequences and/or temporal datasets is often of high importance. In the following we will briefly go over the main ideas one can pursue to achieve these objectives. However there are still many opened problems that need to be investigated.

The basic learning scenario in the control domain is that the learner observes sequences of control actions, observations and reinforcements. Reinforcements represent either costs or rewards and quantify the goodness of the transitions that occurred with regard to the control goal. The learner can either be combined with the controller with the capability to perform actions or it may be only a passive observer. Using active learning can often lead to shorter learning times due to the fact that the controller can explore those control sequences it considers more relevant. On the other hand passive learning assumes that the learner is given information about a control case without any active intervention, which can be crucial in some domains like medicine.

In general, depending on what we want to learn, we can speak about two main learning approaches in control domains :

- learning of POMDP models
- learning of control policies

Learning of the model

The first approach is trying to learn the underlying domain model from observed data and reinforcements. Such a model is then used to compute the optimal or approximate control in the obvious way. Learning of the model can consist of learning the complete model (both structure and parameters) or learning model parameters only. The problem of learning model parameters is far easier and methods for learning parameters of probabilistic networks with hidden variables, like EM [Rabiner, Juang 86] [Spiegelhalter et al. 93][Lauritzen 94] or gradient descent methods [Russell et al. 95], can be applied.

The agent with a built-in parameter learning mechanism can be the basis of an adaptive control agent that adapts its behavior with regard to specificities of the control cases that have been solved. The adaptation of the model parameters can be important, e.g., when there is a natural variation in cases the control agent repeatedly solves, and when one can incorporate in the model initially only population estimates.

The learning of a complete POMDP model is a far harder task, as one is supposed to go beyond learning of parameter values and also derive the underlying hidden structure. The learning of POMDP models has not been explored to a sufficient depth so far and much work need to be done here. Two approaches published are the predictive distinction approach [Chrisman 92] and the utile distinction approach [McCallum 93]. Both of these operate under various simplifying assumptions (restricted value function form) and gradually increase the number of states needed to fit the observed control data.

Learning of control policies

The second approach is based on the assumption that one can build a good controller without the detailed underlying model by building control policies directly based on action-observation sequences. This is in many respects related to the approach of feature-based approximation with truncated histories [Platzman 77] [White, Scherer 94]. Control policies that use truncated histories can be learned, e.g., using reinforcement learning techniques (see, e.g., [Watkins 89], [Barto et al. 91], [Hauskrecht 94], [Kaelbling et al. 96]). The problems with this are that the number of items in the history is not known in advance, and that not all observations and actions are equally relevant to control. An approach that attempts to dynamically identify the relevant history items to be used in the control policy definition was presented in [McCallum 95].

Appendix A

Test problems

A.1 Maze20 problem

Problem: infinite discounted horizon
Optimization: MAX
Discount: 0.9

States correspond to rooms in the maze. They are numbered from 0 to 19 (see figure A-1).

Actions: numbered from 0 to 5.

0	move north;	3	move west;
1	move south;	4	make observation (north-south);
2	move east;	5	make observation (east-west).

Observations: numbered from 0 to 7.

0	no-observation (unknown);	4	both north and south walls;
1	no wall;	5	east wall;
2	north wall;	6	west wall;
3	south wall;	7	both east and west walls.

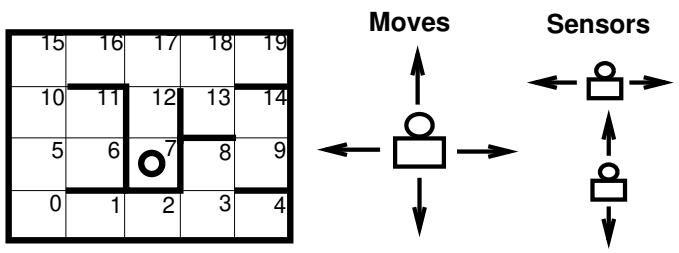


Figure A-1: Maze20 robot navigation problem.

Transition model (action, previous state, next state)

Action: 0

0.15	0.15	0	0	0	0.7	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0.15	0.7	0.15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0.15	0.7	0.15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0.15	0	0.15	0	0	0	0.7	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0.15	0.85	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0.15	0.15	0	0	0	0.7	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0.15	0.15	0	0	0	0	0.7	0	0	0	0	0	0	0	0
0.3	0	0	0	0.3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.4
0	0	0	0	0	0	0	0	0.85	0.15	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0.15	0.15	0	0	0	0	0.7	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0.15	0.15	0	0	0	0.7	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0.15	0.85	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0.3	0	0	0	0	0.7	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0.15	0.15	0	0	0	0.7	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0.15	0.85	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.85	0.15	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.15	0.7	0.15	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.15	0.7	0.15	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.15	0.7	0.15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.15	0.85

Action: 1

0.85	0.15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0.15	0.7	0.15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0.15	0.7	0.15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0.15	0.7	0.15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0.15	0.85	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0.7	0	0	0	0	0.15	0.15	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0.15	0.85	0	0	0	0	0	0	0	0	0	0	0	0	0
0.3	0	0	0	0.3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.4
0	0	0	0.7	0	0	0	0	0.15	0.15	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0.15	0.85	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0.7	0	0	0	0	0.15	0.15	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0.7	0	0	0	0.15	0.15	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0.7	0	0	0	0	0.3	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0.85	0.15	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0.7	0	0	0	0.15	0.15	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0.7	0	0	0	0	0.15	0.15	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.15	0.7	0.15	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0.7	0	0	0	0.15	0	0.15	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0.7	0	0	0	0.15	0	0.15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.15	0.85

Action: 2

0.15	0.7	0	0	0	0.15	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0.3	0.7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0.3	0.7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0.15	0.7	0	0	0	0.15	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0.15	0	0	0	0	0	0.7	0	0	0	0.15	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0.85	0	0	0	0	0.15	0	0	0	0	0	0	0	0
0.3	0	0	0	0.3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.4
0	0	0	0.15	0	0	0	0	0.15	0.7	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0.85	0	0	0	0	0.15	0	0	0	0	0
0	0	0	0	0	0.15	0	0	0	0	0	0.7	0	0	0	0.15	0	0	0	0
0	0	0	0	0	0	0.15	0	0	0	0	0	0.85	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0.15	0	0	0	0	0	0.7	0	0	0	0.15	0	0
0	0	0	0	0	0	0	0	0.15	0	0	0	0	0	0.15	0.7	0	0	0.15	0
0	0	0	0	0	0	0	0	0	0.15	0	0	0	0	0	0.85	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0.15	0	0	0	0	0	0.15	0.7	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.3	0.7	0
0	0	0	0	0	0	0	0	0	0	0	0	0.15	0	0	0	0	0	0.15	0.7
0	0	0	0	0	0	0	0	0	0	0	0	0	0.15	0	0	0	0	0.15	0.7
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.0

Action: 3

0.85	0	0	0	0	0.15	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0.7	0.3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0.7	0.3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0.7	0.15	0	0	0	0	0.15	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0.7	0.3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0.15	0	0	0	0	0.7	0	0	0	0	0.15	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0.7	0.15	0	0	0	0	0.15	0	0	0	0	0	0	0	0
0.3	0	0	0	0.3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.4
0	0	0	0.15	0	0	0	0	0.85	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0.7	0.15	0	0	0	0	0.15	0	0	0	0	0
0	0	0	0	0	0.15	0	0	0	0.7	0	0	0	0	0	0.15	0	0	0	0
0	0	0	0	0	0	0.15	0	0	0	0.7	0.15	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0.15	0	0	0	0	0.7	0	0	0	0	0.15	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0.7	0	0	0	0	0.15	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.85	0	0	0	0	0.15
0	0	0	0	0	0	0	0	0	0.15	0	0	0	0	0.7	0.15	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0.15	0	0	0	0	0	0.85	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.7	0.3	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0.15	0	0	0	0	0.7	0.15	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0.15	0	0	0	0	0.7	0.3

Action: 4

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Action: 5

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Observation model (action, state, observation)

Action: 0

1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0

Action: 1

1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0

Action: 2

1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0

Action: 3

1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0

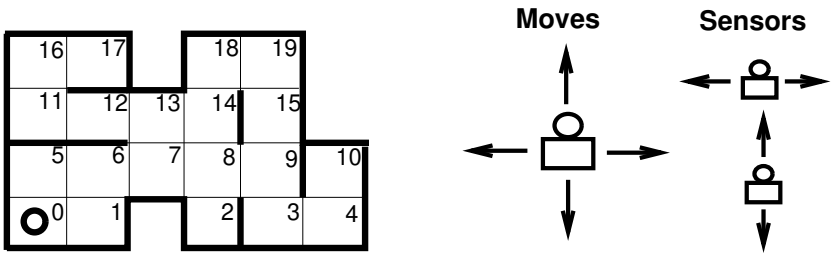


Figure A-2: Maze20B problem

Action: 4

0	0.14	0.01	0.8	0.05	0	0	0
0	0.05	0.1	0.1	0.75	0	0	0
0	0.05	0.1	0.1	0.75	0	0	0
0	0.14	0.01	0.8	0.05	0	0	0
0	0.05	0.1	0.1	0.75	0	0	0
0	0.89	0.05	0.05	0.01	0	0	0
0	0.14	0.01	0.8	0.05	0	0	0
0	0.14	0.01	0.8	0.05	0	0	0
0	0.14	0.8	0.01	0.05	0	0	0
0	0.14	0.01	0.8	0.05	0	0	0
0	0.89	0.05	0.05	0.01	0	0	0
0	0.14	0.8	0.01	0.05	0	0	0
0	0.89	0.05	0.05	0.01	0	0	0
0	0.14	0.01	0.8	0.05	0	0	0
0	0.14	0.8	0.01	0.05	0	0	0
0	0.14	0.8	0.01	0.05	0	0	0
0	0.05	0.1	0.1	0.75	0	0	0
0	0.14	0.8	0.01	0.05	0	0	0
0	0.14	0.8	0.01	0.05	0	0	0
0	0.05	0.1	0.1	0.75	0	0	0

Action: 5

0	0.14	0	0	0	0.01	0.8	0.05
0	0.89	0	0	0	0.05	0.05	0.01
0	0.89	0	0	0	0.05	0.05	0.01
0	0.89	0	0	0	0.05	0.05	0.01
0	0.14	0	0	0	0.8	0.01	0.05
0	0.14	0	0	0	0.01	0.8	0.05
0	0.14	0	0	0	0.8	0.01	0.05
0	0.05	0	0	0	0.1	0.1	0.75
0	0.14	0	0	0	0.01	0.8	0.05
0	0.14	0	0	0	0.8	0.01	0.05
0	0.14	0	0	0	0.01	0.8	0.05
0	0.14	0	0	0	0.8	0.01	0.05
0	0.05	0	0	0	0.1	0.1	0.75
0	0.14	0	0	0	0.01	0.8	0.05
0	0.14	0	0	0	0.8	0.01	0.05
0	0.14	0	0	0	0.01	0.8	0.05
0	0.89	0	0	0	0.05	0.05	0.01
0	0.89	0	0	0	0.05	0.05	0.01
0	0.14	0	0	0	0.8	0.01	0.05

Expected one step reward (action, state)

3.4	1.2	1.2	4.0	0.6	3.4	3.4	150.0	0.6	3.4	3.4	0.6	2.8	3.4	0.6	0.6	1.2	1.2	1.2	0.6
0.6	1.2	1.2	1.2	0.6	3.4	0.6	150.0	3.4	0.6	3.4	3.4	2.8	0.6	3.4	3.4	1.2	4.0	4.0	0.6
3.4	2.8	2.8	3.4	0.0	4.0	0.6	150.0	3.4	0.6	4.0	0.6	1.2	3.4	0.6	3.4	2.8	3.4	3.4	0.0
0.6	2.8	2.8	3.4	2.8	1.2	3.4	150.0	0.6	3.4	1.2	3.4	1.2	0.6	3.4	0.6	2.8	3.4	3.4	2.8
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2

A.2 Maze20B problem

Problem: infinite discounted horizon
Optimization: MIN
Discount: 0.95

States: rooms in the maze, numbered from 0 to 19 (see figure A-2).

Actions and observations: same as for the Maze20 problem.

Transition model (action, previous state, next state)

Action: 0

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0.15	0.15	0	0	0	0	0.7	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0.3	0	0	0	0	0	0.7	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0.15	0.15	0	0	0	0	0.7	0	0	0	0	0	0	0	0	0	0
0	0	0	0.15	0.15	0	0	0	0	0	0.7	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0.85	0.15	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0.15	0.7	0.15	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0.15	0	0.15	0	0	0	0.7	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0.15	0	0.15	0	0	0	0	0.7	0	0	0	0	0
0	0	0	0	0	0	0	0	0.15	0.15	0	0	0	0	0	0.7	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1.0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0.15	0.15	0	0	0	0.7	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0.15	0.7	0.15	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0.15	0.7	0.15	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0.15	0.15	0	0	0	0.7	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.3	0	0	0	0.7
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.85	0.15	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.15	0.85	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.85	0.15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.15	0.85

Action: 1

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0.15	0.85	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0.85	0.15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0.15	0.85	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0.7	0	0	0	0	0.15	0.15	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0.7	0	0	0	0.15	0	0.15	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0.15	0.7	0.15	0	0	0	0	0	0	0	0	0	0	0
0	0	0.7	0	0	0	0	0.15	0	0.15	0	0	0	0	0	0	0	0	0	0
0	0	0	0.7	0	0	0	0	0.15	0.15	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0.7	0	0	0	0	0	0.3	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0.85	0.15	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0.15	0.7	0.15	0	0	0	0	0	0
0	0	0	0	0	0	0	0.7	0	0	0	0	0.15	0	0.15	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0.7	0	0	0	0	0.15	0.15	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0.7	0	0	0	0	0.3	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0.7	0	0	0	0	0.15	0.15	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.15	0.85	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.7	0	0	0.15	0.15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.7	0	0.15	0.15

Action: 2

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0.85	0	0	0	0	0.15	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0.85	0	0	0	0	0	0.15	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0.15	0.7	0	0	0	0	0.15	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0.85	0	0	0	0	0	0.15	0	0	0	0	0	0	0	0	0
0.15	0	0	0	0	0.15	0.7	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0.15	0	0	0	0	0.15	0.7	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0.15	0.7	0	0	0	0	0.15	0	0	0	0	0	0
0	0	0.15	0	0	0	0	0	0	0.7	0	0	0	0	0.15	0	0	0	0	0
0	0	0	0.15	0	0	0	0	0	0.7	0	0	0	0	0	0.15	0	0	0	0
0	0	0	0	0.15	0	0	0	0	0	0.85	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0.15	0.7	0	0	0	0.15	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0.3	0.7	0	0	0	0	0	0
0	0	0	0	0	0	0	0.15	0	0	0	0	0	0.15	0.7	0	0	0	0	0
0	0	0	0	0	0	0	0	0.15	0	0	0	0	0	0.7	0	0	0	0.15	0
0	0	0	0	0	0	0	0	0	0.15	0	0	0	0	0	0.7	0	0	0	0.15
0	0	0	0	0	0	0	0	0	0	0	0.15	0	0	0	0	0.15	0.7	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.15	0	0	0	0.15	0.7
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.15	0	0	0	0.85

Action: 3

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0.7	0.15	0	0	0	0	0.15	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0.85	0	0	0	0	0	0.15	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0.85	0	0	0	0	0	0.15	0	0	0	0	0	0	0	0	0	0
0	0	0	0.7	0.15	0	0	0	0	0	0.15	0	0	0	0	0	0	0	0	0
0.15	0	0	0	0	0.85	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0.15	0	0	0	0.7	0.15	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0.7	0.15	0	0	0	0	0	0.15	0	0	0	0	0	0
0	0	0.15	0	0	0	0	0.7	0	0	0	0	0	0	0.15	0	0	0	0	0
0	0	0	0.15	0	0	0	0	0.7	0	0	0	0	0	0	0.15	0	0	0	0
0	0	0	0	0.15	0	0	0	0	0	0.85	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0.85	0	0	0	0	0.15	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0.7	0.3	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0.15	0	0	0	0	0.7	0.15	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0.15	0	0	0	0	0.7	0	0	0	0	0.15	0
0	0	0	0	0	0	0	0	0	0.15	0	0	0	0	0	0.7	0	0	0	0.15
0	0	0	0	0	0	0	0	0	0	0	0.15	0	0	0	0	0.85	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.7	0.3	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.15	0	0	0	0.85	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.15	0	0	0.7	0.15

Action: 4

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Action: 5

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Observation model (action, state, observation)

Action: 0

1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0

Action: 1

1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0

Action: 2

1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0

Action: 3

1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0

Action: 4

0	0.14	0.01	0.8	0.05	0	0	0
0	0.14	0.01	0.8	0.05	0	0	0
0	0.14	0.01	0.8	0.05	0	0	0
0	0.14	0.01	0.8	0.05	0	0	0
0	0.14	0.01	0.8	0.05	0	0	0
0	0.14	0.8	0.01	0.05	0	0	0
0	0.14	0.8	0.01	0.05	0	0	0
0	0.14	0.01	0.8	0.05	0	0	0
0	0.89	0.05	0.05	0.01	0	0	0
0	0.89	0.05	0.05	0.01	0	0	0
0	0.14	0.8	0.01	0.05	0	0	0
0	0.14	0.01	0.8	0.05	0	0	0
0	0.05	0.1	0.1	0.75	0	0	0
0	0.14	0.8	0.01	0.05	0	0	0
0	0.89	0.05	0.05	0.01	0	0	0
0	0.89	0.05	0.05	0.01	0	0	0
0	0.14	0.8	0.01	0.05	0	0	0
0	0.05	0.1	0.1	0.75	0	0	0
0	0.14	0.8	0.01	0.05	0	0	0
0	0.14	0.8	0.01	0.05	0	0	0

Action: 5

0	0.14	0	0	0	0.01	0.8	0.05
0	0.14	0	0	0	0.8	0.01	0.05
0	0.05	0	0	0	0.1	0.1	0.75
0	0.14	0	0	0	0.01	0.8	0.05
0	0.14	0	0	0	0.8	0.01	0.05
0	0.14	0	0	0	0.01	0.8	0.05
0	0.89	0	0	0	0.05	0.05	0.01
0	0.89	0	0	0	0.05	0.05	0.01
0	0.89	0	0	0	0.05	0.05	0.01
0	0.14	0	0	0	0.8	0.01	0.05
0	0.05	0	0	0	0.1	0.1	0.75
0	0.14	0	0	0	0.01	0.8	0.05
0	0.89	0	0	0	0.05	0.05	0.01
0	0.89	0	0	0	0.05	0.05	0.01
0	0.14	0	0	0	0.8	0.01	0.05
0	0.05	0	0	0	0.1	0.1	0.75
0	0.14	0	0	0	0.01	0.8	0.05
0	0.14	0	0	0	0.8	0.01	0.05
0	0.14	0	0	0	0.01	0.8	0.05
0	0.14	0	0	0	0.8	0.01	0.05

Expected one step cost (action, state)

0	21.5	23.0	21.5	21.5	28.5	27.0	20.0	20.0	21.5	30.0	21.5	27.0	27.0	21.5	23.0	28.5	28.5	28.5	28.5
0	28.5	30.0	28.5	28.5	21.5	20.0	27.0	20.0	21.5	23.0	28.5	27.0	20.0	21.5	23.0	21.5	28.5	21.5	21.5
0	28.5	28.5	21.5	28.5	21.5	21.5	21.5	20.0	27.0	28.5	21.5	23.0	21.5	27.0	27.0	21.5	30.0	21.5	28.5
0	21.5	28.5	28.5	21.5	28.5	21.5	21.5	20.0	20.0	28.5	28.5	23.0	21.5	20.0	27.0	28.5	23.0	28.5	21.5
0	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
0	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20

A.3 Shuttle docking problem

Due to Chrisman [Chrisman 92].

Problem: infinite discounted horizon
Optimization: MAX
Discount: 0.95

States: numbered from 0 to 7. Represent positions of the Shuttle with regard to the least recently visited (LRV) and the most recently visited (MRV) station.

0 Docked in LRV;
1 Just outside space station MRV, front of ship facing station;
2 Space facing MRV;
3 Just outside space station LRV, back of ship facing station;
4 Just outside space station MRV, back of ship facing station;
5 Space facing LRV;
6 Just outside space station LRV, front of ship facing station;
7 Docked in MRV.

Actions: numbered from 0 to 2.

0 TurnAround; 1 GoForward; 2 Backup.

Observations: numbered from 0 to 4.

0 see LRV forward;
1 see MRV forward;
2 docked at MRV;
3 see nothing;
4 docked at LRV.

Transition model (action, previous state, next state)

Action: 0

0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0

Action: 1

0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0

Action: 2

0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
0.0	0.4	0.3	0.0	0.3	0.0	0.0	0.0
0.0	0.0	0.1	0.8	0.0	0.0	0.1	0.0
0.7	0.0	0.0	0.3	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.3	0.0	0.0	0.7
0.0	0.1	0.0	0.0	0.8	0.1	0.0	0.0
0.0	0.0	0.0	0.3	0.0	0.3	0.4	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0

Observation model (action, state, observation)

Action: 0, 1, 2

0.0	0.0	0.0	0.0	1.0
0.0	1.0	0.0	0.0	0.0
0.0	0.7	0.0	0.3	0.0
0.0	0.0	0.0	1.0	0.0
0.0	0.0	0.0	1.0	0.0
0.7	0.0	0.0	0.3	0.0
1.0	0.0	0.0	0.0	0.0
0.0	0.0	1.0	0.0	0.0

Expected one step reward (action, state)

0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	-3.0	0.0	0.0	0.0	0.0	-3.0	0.0
0.0	0.0	0.0	7.0	0.0	0.0	0.0	0.0

Bibliography

[Albus 71] J. Albus. A theory of cerebellar functions. Mathematical biology. 10, pp. 26-61, 1971.

[Anderson et al. 90] K.M. Anderson, P.M. Odell, P.W.F. Wilson, W.B. Kannel. Cardiovascular Disease Risk Profiles. American Heart Journal, 121:293-298, 1990.

[Astrom 65] K.J. Astrom. Optimal control of Markov decision processes with incomplete state estimation. Journal of Mathematical Analysis and Applications, 10, pp. 174-205, 1965.

[Baird 95] L. Baird. Residual algorithms: Reinforcement learning with function approximation. In Proceedings of 12-th International Machine Learning confernce, pp. 30-37, 1995.

[Barto et al. 83] A.G. Barto, R.S. Sutton, C.W. Anderson. Neuronlike adaptive elements that can solve dificult learning control problems. IEEE Transactions on Systems, Man, and Cybernetics, 13, pp. 835 -846, 1983

[Barto et al. 91] A.G. Barto, S.J. Bradtke, S.P. Singh. Real-time learning and control using asynchronous dynamic programming. University of Massachusetts, TR-91-57, 1991.

[Bellman 57] R. Bellman. Adaptive Control Processes. Princeton University Press. Princeton, New Jersey, 1961.

[Bellman, Dreyfus 62] R. Bellman, S. Dreyfus. Applied Dynamic Programming. Princeton University Press, New Jersey, 1962.

[Bertsekas 95] D.P. Bertsekas. Dynamic programming and optimal control. Athena Scientific, 1995.

[Bertsekas, Tsitsiklis 96] D. Bertsekas, J.N. Tsitsiklis. Neuro-dynamic programming. Athena Scientific, 1996.

[Boutillier, Dearden 94] C. Boutilier, R. Dearden. Using abstractions for decision-theoretic planning with time constraints. In Proceedings of AAAI-94, pp. 1016-1022, 1994.

[Boutillier et al. 95] C. Boutilier, R. Dearden, M. Goldszmidt. Exploiting structure in policy construction. In Proceedings of IJCAI-95, pp. 1104-1111, 1995.

[Boutillier, Puterman 95] C. Boutilier, M. Puterman. Process oriented planning and avarage-reward optimality. In Proceedings of IJCAI-95, pp. 1096-1103, 1995.

[Brafman 97] R.I. Brafman. A heuristic variable grid solution method for POMDPs. In Proceedings of AAAI-97, pp. 727-233, 1997.

[Buntine 94] W.L. Buntine. Operations for learning with graphical models. Journal of Artificial Intelligence Research, 2, pp. 159-225, 1994.

[Cassandra et al. 94] A.R. Cassandra, L.P. Kaebling, M.L. Littman. Acting optimally in partially observable stochastic domains. In Proceedings of AAAI-94, pp. 1023-1028, 1994.

[Cassandra 94] A.R. Cassandra. Optimal policies for partially observable Markov decision processes. Brown University, Technical report CS-94-14, 1994.

[Cassandra et.al 96] A. Cassandra, L. Kaelbling, J. Kurien. Acting under uncertainty: discrete Bayesian models for mobile-robot navigation. CS-96-17, Brown University, 1996.

[Cassandra et al. 97] A. Cassandra, M.L. Littman, N.L. Zhang. Incremental pruning: A simple, fast, exact algorithm for Partially observable Markov decision processes. In Proceedings of UAI-97, pp. 54-61, 1997.

[Castillo et al. 97] E. Castillo, J.M. Gutierrez, A.S. Hadi. Expert systems and probabilistic network models. Springer-Verlag, 1997.

[Cheng 88] H.-T. Cheng. Algorithms for partially observable Markov decision processes. PhD thesis, University of British Columbia, 1988.

[Chrisman 92] L. Chrisman. Reinforcement learning with Perceptual Aliasing: The perceptual distinction approach. In Proceeding of AAAI-92, pp. 183-188, 1992.

[D'Ambrosio 96] B. D'Ambrosio. POMDP learning using qualitative belief spaces. unpublished, http://www.cs.orst.edu/ dambrosi/, 1996.

[Dean 91] T. Dean. Decision-theoretic control of inference for time-critical applications. International journal of Intelligent Systems, vol. 6., pp. 417-441, 1991.

[Dean, Wellman 91] T.L.Dean, M.P. Wellman. Planning and control. Morgan Kaufmann, San Mateo, 1991

[Dean et al. 93] T. Dean, L.P. Kaelbling, J. Kirman, A. Nicholson. Planning with deadlines in stochastic domains. In Proceedings of AAAI-93, pp. 574-579, 1993.

[Dean, Lin 95] T. Dean, S.-H. Lin. Decomposition techniques for planning in stochastic domains. In Proceedings of IJCAI-95, pp. 1121-1127, 1995.

[Dean, Lin 95] T. Dean, S.-H. Lin. Decomposition techniques for planning in stochastic domains. Brown univesrtity, TR CS-95-10, 1995.

[Dean, Givan 97] T. Dean, R. Givan. Model minimization in Markov decision processes. In Proceedings of the AAAI-97, pp. 106-111, 1997.

[Dean et al. 97] T. Dean, R. Givan, S. Leach. Model reduction techniques for computing approximately optimal solutions for Markov decision processes. In Proceedings of UAI-97, pp. 124-131, 1997.

[Deng, Moore 95] K. Deng, A.W. Moore. Multiresolution Instance-based learning. In Proceedings of the IJCAI-95, pp. 1233-1239, 1995.

[Eagle 84] J.N. Eagle. The optimal search for a moving target when search path is constrained. Operations Research, 32:5, pp. 1107-1115, 1984.

[Goldszmidt 95] M. Goldszmidt. Fast belief updating using order of magnitude probabilities. In Proceedings of UAI-95, pp. 208-216, 1995.

[Gordon 95a] G.J. Gordon. Stable function approximation in Dynamic Programming. Technical report, CMU-CS-95-103, 1995.

[Hansen 94] E.A. Hansen. Cost-effective sensing during plan execution. In Proceedings of AAAI-94, pp. 1029-1035, 1994.

[Heckerman et al. 89] D.E. Heckerman, J.S. Breese, E.J. Horvitz. The compilation of decisions models. Knowledge Systems Laboratory, KSL-89-58, 1989.

[Hauskrecht 94] M. Hauskrecht. Reinforcement learning of control policies. Term paper, Machine learning, 1994.

[Hauskrecht 95] M. Hauskrecht. Learning Bayesian belief networks from data. MIT EECS Area Exam paper, 36 pages, 1995.

[Hauskrecht 96a] M. Hauskrecht. Dynamic decision making in stochastic partially observable domains. In Procedings of AAAI symposium on AI in Medicine, Stanford University, pp. 69-72, 1996.

[Hauskrecht 96b] M. Hauskrecht. Planning and control in stochastic domains with imperfect information. PhD thesis proposal, EECS, MIT, 1996.

[Hauskrecht 97a] M. Hauskrecht. Dynamic decision making in stochastic partially observable medical domains: Ischemic heart disease example. In Procedings of AIME-97, Grenoble, France, pp. 296-299, 1997.

[Hauskrecht 97b] M. Hauskrecht. Incremental methods for computing bounds in partially observable Markov decision processes. In Proceedings of AAAI-97, pp. 734-739, 1997.

[Hauskrecht 97c] Hauskrecht, M. 1997c. Approximation methods for solving control problems in partially observable Markov decision processes. Technical Memo, MIT-LCS-TM-565, 1997.

[Heyman, Sobel 84] D.P. Heyman, M.J. Sobel. Stochastic methods in Operations research: Stochastic optimization. Volume 2, McGraw-Hill, 1984.

[Hovorka et al. 92] R. Hovorka et.al. Causal probabilistic network modelling - An illustration of its role in the management of chronic diseases. IBM Systems Journal, 31:4, pp. 635-648, 1992.

[Howard 60] R.A. Howard. Dynamic programming and Markov processes. MIT press, Cambridge, 1960.

[Howard, Matheson 84] R.A. Howard, J.E. Matheson. Influence diagrams. In Principles and applications of decision analysis, Vol.2, 1984.

[Jaakkola el.al. 93] T. Jaakkola, M.I. Jordan, S.P. Singh. On the convergence of stochastic iterative dynamic algorithms. AI memo 1441, Massachusetts Institute of Technology, 1993.

[Jensen 96] F.V. Jensen. An introduction to Bayesian networks. Springer, 1996.

[Kanazawa et al. 95] K. Kanazawa, D. Koller, S. Russell. Stochastic simulation algorithms for dynamic probabilistic networks. In Proceeding of UAI-95, pp. 346-351, 1995.

[Kaelbling et al.95] L.P. Kaelbling, M.L. Littman and A.R. Cassandra. Planning and Acting in Partially Observable Stochastic Domains, Unpublished, 1995.

[Kaelbling et al. 96] L.P. Kaelbling, M.P. Littman, A.W. Moore. Reinforcement learning: a survey. Journal of Artificial Intelligence Research, 4:237-285, 1996.

[Koenig, Simmons 95] S. Koenig, R.G Simmons. Real-Time search in non-deterministic domains. In Proceedings of IJCAI-95, pp. 1660-1667, 1995.

[Korf 85] R. Korf. Depth-first iterative deepening: an optimal addmissible tree search. Artificial Intelligence, 27:1, 97-109, 1985.

[Kushmerick et al. 94] N. Kushmerick, S. Hanks, D. Weld. An algorithm for probabilistic least commitment planning. In Proceedings of AAAI-94, pp. 1073-1078, 1994.

[Lauritzen 94] S.L. Lauritzen. The EM algorithm for graphical association models with missing data. Computational Statistics and Data Analysis, 19:2, pp. 191-201, 1994.

[Lauritzen 96] S. L. Lauritzen. Graphical models. Clarendon Press, 1996.

[Leong 94] T.-Y. Leong. An integrated approach to dynamic decision making under uncertainty. MIT/LCS/TR-631, 1994.

[Littman 94] M.L. Littman. The Wittness algorithm: solving partially observable Markov decision processes. Brown University, Technical report CS-94-40, 1994.

[Littman et al. 95a] M.L. Littman, A.R. Cassandra, L.P. Kaelbling. Learning policies for partially observable environmnets: scaling up. In Proceedings of the 12-th international conference on Machine Learning, pp. 362-370, 1995

[Littman et al. 95b] M.L. Littman, T.L. Dean, L.P. Kaelbling. On the complexity of solving Markov decision problems. In Proceedings of UAI-95, pp. 394-402, 1995

[Littman et al. 95c] M.L. Littman, A.R. Cassandra, L.P. Kaelbling. Efficient dynamic programming updates in partially observable Markov decision processes. submitted to Operations Research, 1995.

[Littman 96] M.L. Littman. Algorithms for Sequential decision making. PhD thesis, CS-96-09, Brown University, 1996.

[Lovejoy 91a] W.S. Lovejoy. A survey of algorithmic methods for partially observed Markov decision processes. Annals of Operations Research, 28, pp. 47-66, 1991.

[Lovejoy 91b] W.S. Lovejoy. Computationally feasible bounds for partially observed Markov decision processes. Operations Research, 39:1, pp. 192-175, 1991.

[Lovejoy 93] W.S. Lovejoy. Suboptimal policies with bounds for parameter adaptive decision processes. Operations Research, 41:3, pp. 583-599, 1993.

[McCallum 93] R.A. McCallum. Overcoming Incomplete Perceptions with Utile Distinction Memory. In the Proceedings of the 10-th Machine Learning conference, pp. 190-196, 1993.

[McCallum 95] R.A. McCallum. Instance-based utile distinctions for reinforcement learning with hidden state. In Proceedings of the 12-th International conference on Machine Learning, pp. 387-395, 1995.

[Monahan 82] G.E. Monahan. A survey of partially observable Markov decision processes: theory, models, and algorithms. Management Science, 28:1, pp. 1-16, 1982.

[Moore, Atkenson 93] A.W. Moore, C.G. Atkenson. Prioritized Sweeping: Reinforcement Learning With Less Data and Less Time. Machine Learning, 10, pp. 103-130, 1993

[Papadimitriou, Tsitsiklis 87] C.H. Papadimitriou, J.N. Tsitsiklis. The complexity of Markov decision processes. Mathematics of Operations Research, 12:3, pp. 441-450, 1987.

[Parr, Russell 95] R. Parr, S. Russell. Approximating optimal policies for partially observable Stochastic domains. In Proceedings of IJCAI-95, pp. 1088-1094, 1995.

[Pearl 89] J. Pearl. Probabilistic reasoning in intelligent systems. Morgan Kaufmann, 1989.

[Platzman 77] L.K. Platzman. Finite memory estimation and control of finite probabilistic systems. MIT EECS Department, PhD. thesis, 1977.

[Puterman 94] M.L. Puterman. Markov Decision Processes: discrete stochastic dynamic programming. John Wiley and Sons, 1994.

[Rabiner, Juang 86] L.R. Rabiner, B.H. Juang. An introduction to Hidden Markov Models. In IEEE ASSP magazine 3(1), pp. 4-16, 1986.

[Rumelhart et.al 86] D.E. Rumelhart, G.E.Hinton, R.J. Williams. Learning internal representations by error propagation. In Parallel Distributed Processing, chapter 8, pp. 318-362, 1986.

[Russell, Wefald 91] S. Russell, E. Wefald. Principles of metareasoning. Artificial Intelligence, 49, 1991, pp. 361-395, 1991.

[Russell, Norvig 95] S. Russell, P. Norvig. Artifician Intelligence. A modern approach. Prentice Hall, 1995.

[Russell et al. 95] S. Russell, J. Binder, D. Koller, K. Kanazawa. Local learning in probabilistic networks with hidden variables. In Proceedings of IJCAI-95, pp. 1146-1152, 1995.

[Sachs 84] L. Sachs. Applied statistics: A handbook of techniques. Springer Verlag, 1984.

[Schachter 86] R.D. Schachter. Evaluating influence diagrams. Operations Research, 34:6, pp. 871-882, 1986.

[Schachter, Peot 89] R.D. Schachter, M.A. Peot. Simulation approaches to general probabilistic inference on belief networks. In Proceedings of UAI-89, pp. 311-318, 1989.

[Schachter, Peot 92] R.D. Schachter, M.A. Peot. Decision making using probabilistic inference methods. In Proceedings of UAI-92, pp. 276-283, 1992.

[Singh et al. 94] S.P. Singh, T. Jaakkola, M.I. Jordan. Learning Without State-Estimation in Partially observable Markovian Decision Processes. In the Proceedings of the 11-th conference on Machine Learning, pp. 284-292, 1994.

[Smallwood, Sondik 73] R.D. Smallwood, E.J. Sondik. The optimal control of Partially observable processes over a finite horizon. Operations Research, 21, pp. 1071-1088, 1973.

[Spiegelhalter et al. 93] D.J. Spiegelhalter, A.P. Dawid, S.L. Lauritzen, R.G. Cowell. Bayesian Analysis in Expert Systems. Statistical Science, 8:3, pp. 219-247, 1993.

[Sondik 78] E.J. Sondik. The optimal control of partially observable Markov processes over the infinite horizon: discounted cost. Operations Research, 26, 1978.

[Strang 86] G. Strang. Introduction to applied mathematics. Wellesley-Cambridge Press, 1986.

[Sutton 88] R.S. Sutton. Learning to Predict by the Methods of Temporal Differences. Machine Learning, 3, pp. 9-44, 1988.

[Sutton 90] R.S. Sutton. Integrated architecture for Learning, Planning, and Reacting Based on Approximating Dynamic Programming. Proceedings of the Seventh International Conference on Machine Learning, Morgan Kaufmann, pp. 216-224, 1990.

[Tash, Russell 94] J. Tash, S. Russell. Control strategies for a stochastic planner. In Proceedings of AAAI-94, pp. 1079-1085, 1994.

[Tsitsiklis, Van Roy 96] J.N. Tsitsiklis, B. Van Roy. Feature-based methods for large scale Dynamic programming. Machine Learning, 22, pp.59-94, 1996.

[Washington 96] R. Washington. Incremental Markov model planning. In Proceedings of ICTAI-96, 1996.

[Watkins 89] C.J.C.H. Watkins. Learning from Delayed Rewards. PhD thesis, Cambridge University, 1989.

[Webber et al. 92] B.L. Webber, R. Rymon, J.R. Clarke. Flexible support for Trauma management through goal directed reasoning and planning. Artificial Intelligence in Medicine, 4:2, 1992.

[White, Scherer 89] C.C. White III, W.T. Scherer. Solution procedures for partially observed Markov decision processes. Operations Research, 37:5, pp. 791-797, 1989.

[White, Scherer 94] C.C. White III, W.T. Scherer. Finite memory suboptimal design for partially observed Markov decision processes. Operations Research, 42:3, pp. 439-455, 1994.

[Whitehead, Ballard 90] S.D. Whitehead, D.H, Ballard. Active Perception and Reinfocement learning. In the Proceedings of the 7-th conference on Machine Learning, pp. 179-188, 1990.

[Wong et al. 90] J.B. Wong et al. Myocardial revascularization for chronic stable angina. Annals of Internal Medicine, 113 (1), pp. 852-871, 1990.

[Zhang, Liu 96] N.L. Zhang, W. Liu. Planning in stochastic domains: Problem characteristics and approximations. Technical report HKUST-CS96-31, 1996.