

Decision Making Under Uncertainty

MIT Lincoln Laboratory Series

Perspectives on Defense Systems Analysis: The What, the Why, and the Who, but Mostly the How of Broad Defense Systems Analysis, William P. Delaney

Ultrawideband Phased Array Antenna Technology for Sensing and Communications Systems, Alan J. Fenn and Peter T. Hurst

Decision Making Under Uncertainty: Theory and Application, Mykel J. Kochenderfer

Applied State Estimation and Association, Chaw-Bing Chang and Keh-Ping Dunn

MIT Lincoln Laboratory is a federally funded research and development center that applies advanced technology to problems of national security. The books in the *MIT Lincoln Laboratory Series* cover a broad range of technology areas in which Lincoln Laboratory has made leading contributions. The books listed above and future volumes in this series renew the knowledge-sharing tradition established by the seminal *MIT Radiation Laboratory Series* published between 1947 and 1953.

Decision Making Under Uncertainty

Theory and Application

Mykel J. Kochenderfer

with contributions from

Christopher Amato

Girish Chowdhary

Jonathan P. How

Hayley J. Davison Reynolds

Jason R. Thornton

Pedro A. Torres-Carrasquillo

N. Kemal Üre

John Vian

The MIT Press

Cambridge, Massachusetts

London, England

© 2015 Massachusetts Institute of Technology

All rights reserved. No part of this book may be reproduced in any form by any electronic or mechanical means (including photocopying, recording or information storage and retrieval) without permission in writing from the publisher.

MIT Press books may be purchased at special quantity discounts for business or sales promotional use. For information, please email special_sales@mitpress.mit.edu.

This book was set in Adobe Garamond Pro by the author in $\text{\LaTeX}$.
Printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Data

Kochenderfer, Mykel J., 1980–

Decision making under uncertainty : theory and application / Mykel J. Kochenderfer ; with Christopher Amato, Girish Chowdhary, Jonathan P. How, Hayley J. Davison Reynolds, Jason R. Thornton, Pedro A. Torres-Carrasquillo, N. Kemal Üre, and John Vian.

p. cm — (Lincoln Laboratory series)

Includes bibliographical references and index.

ISBN 978-0-262-02925-4 (hardcover : alk. paper)

1. Intelligent control systems. 2. Automatic machinery. 3. Decision making—Mathematical models. I. Title.

TJ217.5.K63 2015

003'.56—dc23

2014048127

10 9 8 7 6

Dedication

To my family.

Table of Contents

Preface	xix
About the Authors	xxi
Acknowledgments	xxv
1 Introduction	1
1.1 Decision Making	1
1.2 Example Applications	2
1.2.1 Traffic Alert and Collision Avoidance System	2
1.2.2 Unmanned Aircraft Persistent Surveillance	3
1.3 Methods for Designing Decision Agents	4
1.3.1 Explicit Programming	4
1.3.2 Supervised Learning	4
1.3.3 Optimization	5
1.3.4 Planning	5
1.3.5 Reinforcement Learning	5
1.4 Overview	5
1.5 Further Reading	7
References	7
I THEORY	9
2 Probabilistic Models	11
2.1 Representation	11
2.1.1 Degrees of Belief and Probability	12
2.1.2 Probability Distributions	13
2.1.3 Joint Distributions	16
2.1.4 Bayesian Network Representation	17

2.1.5	Conditional Independence	19
2.1.6	Hybrid Bayesian Networks	21
2.1.7	Temporal Models	23
2.2	Inference	25
2.2.1	Inference for Classification	26
2.2.2	Inference in Temporal Models	29
2.2.3	Exact Inference	30
2.2.4	Complexity of Exact Inference	33
2.2.5	Approximate Inference	35
2.3	Parameter Learning	40
2.3.1	Maximum Likelihood Parameter Learning	40
2.3.2	Bayesian Parameter Learning	42
2.3.3	Nonparametric Learning	45
2.4	Structure Learning	46
2.4.1	Bayesian Structure Scoring	46
2.4.2	Directed Graph Search	48
2.4.3	Markov Equivalence Classes	51
2.4.4	Partially Directed Graph Search	52
2.5	Summary	52
2.6	Further Reading	54
	References	54
3	Decision Problems	57
3.1	Utility Theory	57
3.1.1	Constraints on Rational Preferences	58
3.1.2	Utility Functions	58

3.1.3	Maximum Expected Utility Principle	59
3.1.4	Utility Elicitation	60
3.1.5	Utility of Money	60
3.1.6	Multiple Variable Utility Functions	61
3.1.7	Irrationality	63
3.2	Decision Networks	64
3.2.1	Evaluating Decision Networks	65
3.2.2	Value of Information	66
3.2.3	Creating Decision Networks	67
3.3	Games	68
3.3.1	Dominant Strategy Equilibrium	69
3.3.2	Nash Equilibrium	70
3.3.3	Behavioral Game Theory	71
3.4	Summary	72
3.5	Further Reading	72
	References	74
4	Sequential Problems	77
4.1	Formulation	77
4.1.1	Markov Decision Processes	77
4.1.2	Utility and Reward	78
4.2	Dynamic Programming	79
4.2.1	Policies and Utilities	79
4.2.2	Policy Evaluation	80
4.2.3	Policy Iteration	81
4.2.4	Value Iteration	81

4.2.5	Grid World Example	83
4.2.6	Asynchronous Value Iteration	84
4.2.7	Closed- and Open-Loop Planning	84
4.3	Structured Representations	89
4.3.1	Factored Markov Decision Processes	89
4.3.2	Structured Dynamic Programming	89
4.4	Linear Representations	91
4.5	Approximate Dynamic Programming	93
4.5.1	Local Approximation	93
4.5.2	Global Approximation	96
4.6	Online Methods	99
4.6.1	Forward Search	99
4.6.2	Branch and Bound Search	100
4.6.3	Sparse Sampling	101
4.6.4	Monte Carlo Tree Search	102
4.7	Direct Policy Search	103
4.7.1	Objective Function	104
4.7.2	Local Search Methods	104
4.7.3	Cross Entropy Methods	105
4.7.4	Evolutionary Methods	106
4.8	Summary	108
4.9	Further Reading	108
	References	110
5	Model Uncertainty	113
5.1	Exploration and Exploitation	113

5.1.1	Multi-Armed Bandit Problems	113
5.1.2	Bayesian Model Estimation	114
5.1.3	Ad Hoc Exploration Strategies	115
5.1.4	Optimal Exploration Strategies	115
5.2	Maximum Likelihood Model-Based Methods	116
5.2.1	Randomized Updates	117
5.2.2	Prioritized Updates	118
5.3	Bayesian Model-Based Methods	118
5.3.1	Problem Structure	119
5.3.2	Beliefs over Model Parameters	119
5.3.3	Bayes-Adaptive Markov Decision Processes	120
5.3.4	Solution Methods	121
5.4	Model-Free Methods	121
5.4.1	Incremental Estimation	121
5.4.2	Q-Learning	122
5.4.3	Sarsa	123
5.4.4	Eligibility Traces	123
5.5	Generalization	124
5.5.1	Local Approximation	125
5.5.2	Global Approximation	126
5.5.3	Abstraction Methods	128
5.6	Summary	129
5.7	Further Reading	129
	References	130
6	State Uncertainty	133

6.1	Formulation	133
6.1.1	Example Problem	133
6.1.2	Partially Observable Markov Decision Processes	134
6.1.3	Policy Execution	134
6.1.4	Belief-State Markov Decision Processes	134
6.2	Belief Updating	136
6.2.1	Discrete State Filter	136
6.2.2	Linear-Gaussian Filter	138
6.2.3	Particle Filter	138
6.3	Exact Solution Methods	140
6.3.1	Alpha Vectors	140
6.3.2	Conditional Plans	141
6.3.3	Value Iteration	143
6.4	Offline Methods	144
6.4.1	Fully Observable Value Approximation	144
6.4.2	Fast Informed Bound	144
6.4.3	Point-Based Value Iteration	145
6.4.4	Randomized Point-Based Value Iteration	146
6.4.5	Point Selection	147
6.4.6	Linear Policies	149
6.5	Online Methods	149
6.5.1	Lookahead with Approximate Value Function	149
6.5.2	Forward Search	150
6.5.3	Branch and Bound	151
6.5.4	Monte Carlo Tree Search	152

6.6	Summary	155
6.7	Further Reading	155
	References	156
7	Cooperative Decision Making	159
7.1	Formulation	159
7.1.1	Decentralized POMDPs	159
7.1.2	Example Problem	161
7.1.3	Solution Representations	162
7.2	Properties	164
7.2.1	Differences with POMDPs	164
7.2.2	Dec-POMDP Complexity	165
7.2.3	Generalized Belief States	165
7.3	Notable Subclasses	166
7.3.1	Dec-MDPs	166
7.3.2	ND-POMDPs	168
7.3.3	MMDPs	169
7.4	Exact Solution Methods	170
7.4.1	Dynamic Programming	170
7.4.2	Heuristic Search	172
7.4.3	Policy Iteration	175
7.5	Approximate Solution Methods	177
7.5.1	Memory-Bounded Dynamic Programming	177
7.5.2	Joint Equilibrium Search	178
7.6	Communication	178
7.7	Summary	180

7.8	Further Reading	180
	References	182
II	APPLICATION	189
8	Probabilistic Surveillance Video Search	191
8.1	Attribute-Based Person Search	191
8.1.1	Applications	192
8.1.2	Person Detection	193
8.1.3	Retrieval and Scoring	194
8.2	Probabilistic Appearance Model	195
8.2.1	Observed States	195
8.2.2	Basic Model Structure	197
8.2.3	Model Extensions	202
8.3	Learning and Inference Techniques	206
8.3.1	Parameter Learning	207
8.3.2	Hidden State Inference	211
8.3.3	Scoring Algorithm	214
8.4	Performance	217
8.4.1	Search Accuracy	217
8.4.2	Search Timing	220
8.5	Interactive Search Tool	223
8.6	Summary	225
	References	227
9	Dynamic Models for Speech Applications	229
9.1	Modeling Speech Signals	229

9.1.1	Feature Extraction	230
9.1.2	Hidden Markov Models	230
9.1.3	Gaussian Mixture Models	231
9.1.4	Expectation-Maximization Algorithm	232
9.2	Speech Recognition	232
9.3	Topic Identification	235
9.4	Language Recognition	236
9.5	Speaker Identification	238
9.5.1	Forensic Speaker Recognition	240
9.6	Machine Translation	242
9.7	Summary	243
	References	243
10	Optimized Airborne Collision Avoidance	249
10.1	Airborne Collision Avoidance Systems	249
10.1.1	Traffic Alert and Collision Avoidance System	250
10.1.2	Limitations of Existing System	251
10.1.3	Unmanned Aircraft Sense and Avoid	252
10.1.4	Airborne Collision Avoidance System X	253
10.2	Collision Avoidance Problem Formulation	253
10.2.1	Resolution Advisories	253
10.2.2	Dynamic Model	255
10.2.3	Reward Function	256
10.2.4	Dynamic Programming	258
10.3	State Estimation	259
10.3.1	Sensor Error	259

10.3.2	Pilot Response	260
10.3.3	Time to Potential Collision	260
10.4	Real-Time Execution	261
10.4.1	Online Costs	261
10.4.2	Multiple Threats	262
10.4.3	Traffic Alerts	263
10.5	Evaluation	265
10.5.1	Safety Analysis	265
10.5.2	Operational Suitability and Acceptability	267
10.5.3	Parameter Tuning	271
10.5.4	Flight Test	272
10.6	Summary	273
	References	274
11	Multiagent Planning for Persistent Surveillance	277
11.1	Mission Description	277
11.2	Centralized Problem Formulation	278
11.2.1	State Space	278
11.2.2	Action Space	279
11.2.3	State Transition Model	279
11.2.4	Reward Function	280
11.3	Decentralized Approximate Formulations	280
11.3.1	Factored Decomposition	280
11.3.2	Group Aggregate Decomposition	281
11.3.3	Planning	281
11.4	Model Learning	282

11.5	Flight Test	285
11.6	Summary	286
	References	289
12	Integrating Automation with Humans	291
12.1	Human Capabilities and Coping	291
12.1.1	Perceptual and Cognitive Capabilities	291
12.1.2	Naturalistic Decision Making	294
12.2	Considering the Human in Design	296
12.2.1	Trust and Value of Decision Logic Transparency	296
12.2.2	Designing for Different Levels of Certainty	300
12.2.3	Supporting Decisions over Long Timescales	305
12.3	A Systems View of Implementation	308
12.3.1	Interface, Training, and Procedures	308
12.3.2	Measuring Decision Support Effectiveness	311
12.3.3	Organization Influences on System Effectiveness	313
12.4	Summary	313
	References	314
	Index	317

Preface

This book provides an introduction to decision making under uncertainty from a computational perspective. The aim of the first part of the book is to familiarize the reader with the foundations of probabilistic models and decision theory. The second part of the book discusses the application of the theory to problems relevant to a variety of mission areas. The subject of decision making under uncertainty is quite broad and has its origins in several different fields. The text aims to be as concise as possible, providing references to additional material that may be relevant to a wide set of applications.

The target audience for this book includes advanced engineering undergraduate and graduate students as well as professionals. Disciplines for which the book would be especially useful include computer science, aerospace, electrical engineering, and operations research. The text is intended to be introductory in nature. Although algorithms are outlined in the text, proofs are omitted. The book requires some mathematical maturity and assumes some prior exposure to probability theory and calculus. The first five chapters can be used as the basis of an undergraduate or graduate course. The topics in Chapters 6 and 7 are more appropriate for the graduate level.

The book was written over the course of two years while I was at Lincoln Laboratory, a federally funded research and development center of the Massachusetts Institute of Technology. While teaching a course on decision making under uncertainty, I was invited by a member of the Lincoln Laboratory book series to prepare a volume. Much of the material in this book originated from the course. The later part of the course consisted of guest lectures from researchers from Lincoln Laboratory and campus with the aim to show how the principles and techniques discussed in the first part of the course can be applied to problems of national interest. Some of these guest lectures have become chapters in this book.

MYKEL J. KOCHENDERFER
Stanford, Calif.
February 6, 2015

Ancillary material is available on the book's webpage:
<http://mitpress.mit.edu/decision-making-under-uncertainty>

About the Authors



Mykel J. Kochenderfer is an assistant professor in the Department of Aeronautics and Astronautics at Stanford University. Prior to joining the faculty at Stanford, he was a staff member at MIT Lincoln Laboratory. He received BS and MS degrees in computer science from Stanford University and a doctorate from the University of Edinburgh. His current research activities include airspace modeling and aircraft collision avoidance. In 2011, Prof. Kochenderfer was awarded the Lincoln Laboratory Early Career Technical Achievement Award for recognition of his development of a new collision avoidance system and advanced techniques for improving air traffic safety. He was involved in artificial intelligence research at Rockwell Scientific, the Honda Research Institute, and Microsoft Research. He is a third-generation pilot.



Christopher Amato is an assistant professor at Northeastern University. He received a BA from Tufts University and an MS and a PhD from the University of Massachusetts, Amherst. Before joining Northeastern, he was a research scientist at Aptima, Inc. and a postdoc and research scientist at MIT, and an assistant professor at the University of New Hampshire. His research interests include decision making under uncertainty, machine learning, and multi-agent systems.



Girish Chowdhary is an assistant professor at the University of Illinois at Urbana Champaign. He received his MS and PhD degrees from Georgia Institute of Technology. He was a postdoctoral researcher at Georgia Institute of Technology and MIT. He also has research experience at the German Aerospace Center's Institute of Flight Systems. His ongoing research interest is in creating provable algorithms to enable intelligent adaptive autonomy for large-scale and long-duration operation involving collaborating mobile agents. He has authored more than 90 peer reviewed papers in the areas of adaptive control, system identification, distributed sensing and inference, and mission planning, and is the winner of the Air Force Young Investigator award and ACGSC Dave Ward memorial award.



Jonathan P. How is a professor in the Department of Aeronautics and Astronautics at MIT. He received a BA Sc from the University of Toronto in 1987 and his SM and PhD in aeronautics and astronautics from MIT in 1990 and 1993, respectively. How studied for two years as a postdoctoral associate for the Middeck Active Control Experiment, which flew on board the Space Shuttle Endeavour. Prior to joining the faculty at MIT in 2000, he was an assistant professor in the Department of Aeronautics and Astronautics at Stanford University. Among his other achievements, Prof. How was the planning and control lead for the MIT DARPA Urban Challenge team, was the recipient of the 2002 Institute of Navigation Burka Award, and received a Boeing Special Invention award in 2008.



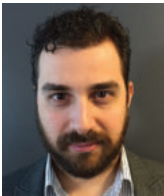
Hayley J. Davison Reynolds has been a technical staff member at MIT Lincoln Laboratory since 2009. She previously worked for Instrata, Ltd. in Cambridge, UK, as a human-computer interaction consultant for Microsoft, Yahoo, and the London 2012 Olympic Committee. She received her PhD in aeronautical systems and applied psychology and SM in aeronautics and astronautics from MIT in 2006 and 2001, respectively. She received her BS in psychology from the University of Illinois in 1999. Her research interests include human-systems integration of complex systems in aviation and biosurveillance, and any messy human-systems problems in general.



Jason R. Thornton is an assistant group leader in the Informatics and Decision Support Group at MIT Lincoln Laboratory. He received a BS degree in computer science from the University of California at Irvine and MS and PhD degrees in electrical and computer engineering from Carnegie Mellon University, where he was awarded the A.G. Milnes Award for an electrical and computer engineering thesis of the highest quality in 2007. His work at Lincoln Laboratory includes research related to sensor fusion, image and video processing, and probabilistic models. In 2012, he received the Lincoln Laboratory Early Career Technical Achievement Award for his development of novel video processing techniques.



Pedro A. Torres-Carrasquillo joined the Information Systems Technology Group, currently Human Language Technology Group, at MIT Lincoln Laboratory in 2002 as a technical staff member. At Lincoln Laboratory, he has been involved in a number of areas related to information extraction from speech, including language, dialect, and speaker recognition. His current areas of interest include automatic dialect recognition by combining multiple knowledge sources and speaker recognition in low-data conditions focusing on forensic speaker recognition. Recently, he has been overseeing the development of software tools for speaker recognition. Dr. Torres-Carrasquillo received a BS degree from the University of Puerto Rico at Mayaguez in 1992, an MS degree from Ohio State University in 1995, and a PhD degree from Michigan State University in 2002, all in electrical engineering.



N. Kemal Üre is an assistant professor in the Department of Aeronautical Engineering at Istanbul Technical University. He received his BSc and MS degrees from Istanbul Technical University and a doctoral degree in Aeronautics and Astronautics from the MIT. His research interests include multiagent learning and control of agile autonomous vehicles.



John Vian is a technical fellow at Boeing Research and Technology, responsible for leading multivehicle autonomous systems research. He has 30 years of experience in flight controls, autonomous systems, and vehicle health management. Dr. Vian received a BS from Purdue and MS and PhD degrees from Wichita State University. He has taught at Embry-Riddle Aeronautical University and Cogswell College.

Acknowledgments

Throughout the process of preparing this manuscript for the Lincoln Laboratory book series, Jim Ward and Dave Martinez have been very generous with their time in helping me. The final draft benefited greatly from Dorothy Ryan's careful copyediting.

Over the past two years, a number of colleagues and friends have made comments and suggestions on early drafts, including Jonathan Christensen, James Chryssanthacopoulos, Louis Dressel, Ann Drumm, Ryan Gardner, Lucas Hansen, Jeremy Kepner, Youngjun Kim, Mary Anne Kochenderfer, Jim Kuchar, Robert Moss, Wes Olson, Carl Quillen, Dorothy Ryan, Josh Silberman, Tan Trinh, Michael Watson, and Chulhee Yun.

Chapter 7 is extended from a series of tutorials developed with Shlomo Zilberstein and Matthijs Spaan. Chapter 8 discusses research activities funded by the Assistant Secretary of Defense for Research and Engineering. Chapter 10 describes research funded by the Federal Aviation Administration under the direction of Neal Suchy. Chapter 11 discusses research supported by Boeing Research & Technology.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. government.

1

Introduction

Mykel J. Kochenderfer

Many important problems involve decision making under uncertainty, including aircraft collision avoidance, wildfire management, and disaster response. When one is designing automated decision support systems, it is important to account for the various sources of uncertainty when making or recommending decisions. Accounting for these sources of uncertainty and carefully balancing the multiple objectives of the system can be very challenging. We will discuss these challenges from a computational perspective, aiming to provide the theory behind decision-making models and algorithms and then illustrating the theory on a collection of real problems. This chapter introduces the problem of decision making under uncertainty, discusses the space of possible approaches to the problem, and overviews the remainder of the book.

1.1 Decision Making

An agent is something that acts based on observations of its environment. Agents may be physical entities, like humans or robots, or they may be nonphysical entities, such as decision support systems that are implemented entirely in software. As shown in Figure 1.1, the interaction between the agent and the world follows an observe-act cycle.

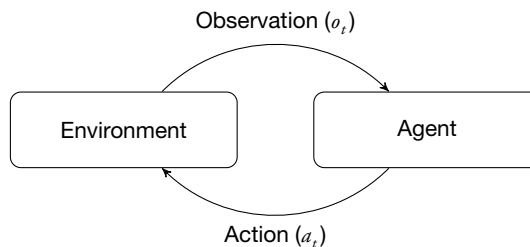


Figure 1.1 Interaction between the environment and the agent.

The agent at time t receives an observation of the world, denoted o_t . Observations may be made, for example, through a biological sensory process as in humans or by a sensor system like radar in an air traffic control system. Observations are often incomplete or noisy; humans may not see an approaching aircraft or a radar system might miss a detection through electromagnetic interference. The agent then chooses an action a_t through some decision-making process to be discussed later. This action, such as sounding an alert, may have a nondeterministic effect on the world.

Our focus is on intelligent agents that interact intelligently in the world to achieve their objectives over time. Given the past sequence of observations $o_0, \dots, o_t$ and knowledge about the environment, the agent must choose an action a_t that best achieves its objectives.

1.2 Example Applications

There are many examples of problems in which accounting for uncertainty is important. This section outlines two of them, both of which are revisited in the latter part of this book.

1.2.1 Traffic Alert and Collision Avoidance System

An example of a decision support system that has significantly improved the safety of air travelers worldwide is the Traffic Alert and Collision Avoidance System (TCAS). TCAS is an onboard collision avoidance system that has been mandated on all aircraft with a maximum takeoff mass of more than 5700 kg or authorized to carry more than 19 passengers. The system provides resolution advisories to the pilots, instructing them to adjust their climb or descent rate to avoid collision. The advisory is announced aurally in the cockpit as well as visually on the instrument display.

Figure 1.2 shows an example of a resolution advisory displayed on the vertical speed indicator. The white arrow, pointing at 0 ft/min, indicates the current vertical rate. The vertical speed dial is in 1000s of feet per minute. The green arc, shown ranging from 1500 ft/min to 2000 ft/min in the figure, informs the pilots to begin climbing at a rate within that range. The red square and white diamond indicate the relative lateral positions of the intruding aircraft, and the numbers below the shapes indicate their relative altitude in 100s of feet.

The TCAS surveillance system sends out interrogations over the radio and listens for replies from the beacons onboard the other aircraft. The distance to the other aircraft can be inferred from measurements of the reply delay. Because TCAS has multiple antennas, small differences in the reply delay permit the inference of the bearing angle to the intruder. Replies over the radio also include the altitude of the aircraft. The TCAS

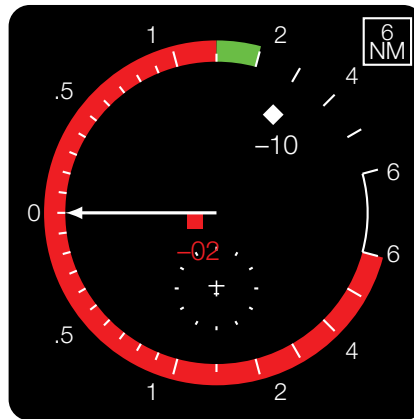


Figure 1.2 TCAS display.

logic determines which resolution advisory to issue based on estimates of the range, bearing, and altitude.

In this example, TCAS is the agent, and the environment is composed of both the aircraft and pilots involved in the encounter. The observations consist of range, bearing, and altitude. The actions available to TCAS consist of climb or descent rate commands. The actions taken by the system do not have a deterministic effect on the environment. Radar data have shown that there is significant variability in the response of the pilots to their advisories.

Although TCAS may appear to be a rather simple decision support system, it has required decades of careful design. Given the uncertainty in the observations resulting from imperfect sensors and the uncertainty in the future trajectories of the aircraft, it is far from straightforward whether to delay an advisory or change the commanded rate part way through an encounter. The consequence of a poor choice could cost the lives of hundreds of passengers. The system must provide an exceptional guarantee of safety while staying operationally acceptable and not disrupting normal air traffic procedures.

1.2.2 Unmanned Aircraft Persistent Surveillance

Unmanned aircraft can provide persistent surveillance over an area of interest, such as a forest fire or battle theater. One approach for enabling surveillance over long durations is to use a team of geographically distributed, low-cost aircraft. It is important that the algorithms that drive the aircraft account for communication constraints and the health of the aircraft.

There are several challenges for building autonomous systems for such a scenario. Some aircraft will need to be allocated to a communication relay area to route commu-

nications between the control base and the mission area. Without the relay, important mission data might not be collected and acted upon. The aircraft also have fuel constraints and can only operate for a limited amount of time in the communication or surveillance areas before returning to base. The fuel depletion rate is stochastic.

The aircraft must be robust to failures in the sensors and actuators. At any point during the mission, a sensor or actuator may fail unexpectedly. An aircraft with a sensor failure becomes useless in the surveillance area, but it can serve as a communication relay. However, if an aircraft has an actuator failure, then the aircraft is not useful for any of the tasks and must be repaired at the base. Building a team of agents to fulfill this mission with a high degree of reliability is extremely challenging.

1.3 Methods for Designing Decision Agents

There are many different methods for designing decision agents. Depending on the application, some methods may be more appropriate than others. The methods differ in the responsibilities of the designer and the tasks left to automation. This section briefly overviews a collection of these methods. The book will focus primarily on the last two methods, planning and reinforcement learning, but some of the techniques will involve elements of supervised learning and optimization.

1.3.1 Explicit Programming

The most direct method for designing a decision agent is to anticipate all the different scenarios the agent might find itself in and then explicitly program the agent to do what is desired. The explicit programming approach may work well for simple problems, but it places a large burden on the designer to provide a complete strategy. Various agent programming languages and frameworks have been proposed to make programming agents easier.

1.3.2 Supervised Learning

In some problems, it may be easier to show an agent what to do rather than to write a program for the agent to follow. The designer provides a set of training examples, and an automated learning algorithm must generalize from these examples. This approach is known as supervised learning and has been widely applied to classification problems. This technique is sometimes called behavioral cloning when applied to learning mappings from observations to actions. Behavioral cloning works well when an expert designer actually knows the best course of action for a representative collection of example situations. Although there exists a wide variety of different learning algorithms, they generally cannot perform better than human designers in new situations.

1.3.3 Optimization

Another approach is for the designer to specify the space of possible decision strategies and a performance measure to be maximized. Evaluating the performance of a decision strategy generally involves running a batch of simulations with the decision strategy. The optimization algorithm then performs a search in this space for the optimal strategy. If the space of possible strategies is relatively low dimensional and the performance measure does not have many local optima, then various local or global search strategies may be appropriate. Although knowledge of a dynamic model is generally assumed in order to run the simulations, it is not otherwise used to guide the search for the optimal strategy, which can be important in complex problems.

1.3.4 Planning

Planning is a form of optimization, but it uses a model of the problem dynamics to help guide the search. A broad literature has arisen on planning problems, much of it focused on deterministic problems. For some problems, it may be acceptable to approximate the dynamics with a deterministic model. Assuming a deterministic model permits the use of methods that more easily scale to high-dimensional problems. For other problems, accounting for future uncertainty is absolutely critical. This book focuses entirely on problems in which accounting for uncertainty is important.

1.3.5 Reinforcement Learning

Reinforcement learning relaxes the assumption in planning that a model is known ahead of time. Instead, the decision-making strategy is learned while the agent interacts with the world. The designer only has to provide a performance measure; it is up to a learning algorithm to optimize the behavior of the agent. One of the interesting complexities that arises in reinforcement learning is that the choice of action impacts not only the immediate success of the agent in achieving its objectives but also the agent's ability to learn about the environment and identify the characteristics of the problem that it can exploit.

1.4 Overview

This book is organized into two parts: theory and application. The theory part is organized as follows:

- Chapter 2: *Probabilistic Models* discusses how uncertainty is represented. It introduces Bayesian networks as a graphical model that captures probabilistic relationships between variables. The chapter presents algorithms for making inferences from these representations and explains how to learn the structure and parameters from data.
- Chapter 3: *Decision Problems* presents utility theory as a framework for understanding optimal decision making under uncertainty. This chapter focuses entirely on single shot decisions. It presents decision networks as a generalization of Bayesian networks with the introduction of decision and utility nodes. The chapter also discusses decision making in the context of multiple, potentially competing, agents.
- Chapter 4: *Sequential Problems* discusses the problem of making decisions over time when the outcomes of the actions are probabilistic. It introduces Markov decision processes as a way to model such problems. The chapter shows how to compute optimal solutions using a process known as dynamic programming. Because many problems are too complex to solve exactly, this chapter also discusses a variety of different approximation methods, such as online methods and direct policy search.
- Chapter 5: *Model Uncertainty* introduces the challenges that arise in solving sequential problems when the dynamic model is not known exactly. It presents a variety of methods for balancing exploration with exploitation and overviews both model-based and model-free approaches. The chapter concludes with a discussion of how to generalize from limited interaction with the environment.
- Chapter 6: *State Uncertainty* presents a formulation known as a partially observable Markov decision process that accounts for uncertainty resulting from imperfect observations. Such a formulation requires updating beliefs about the current state of the system. This chapter presents both offline and online methods for solving these problems.
- Chapter 7: *Cooperative Decision Making* introduces decision making in a collaborative environment where there are multiple interacting agents. This chapter presents some of the properties of such problems, notable subclasses, and algorithms for computing exact and approximate solutions.

The application part of the book shows how the concepts introduced in the theory part can be applied to real problems. The application chapters are organized as follows:

- Chapter 8: *Probabilistic Surveillance Video Search* discusses a probabilistic approach to attribute-based person search. This chapter describes the probabilistic appearance model used in this application and how the model can be used for learning and inference.
- Chapter 9: *Dynamic Models for Speech Applications* provides a broad overview of how the probabilistic methods introduced in the theory chapters have led to major

advances in speech recognition, topic identification, language recognition, speaker identification, and machine translation.

- Chapter 10: *Optimized Airborne Collision Avoidance* explains how to represent the problem of collision avoidance as a partially observable Markov decision process. The chapter explains how to use dynamic programming to produce safer collision avoidance systems with fewer disruptions to the airspace.
- Chapter 11: *Multiagent Planning for Persistent Surveillance* describes how the algorithms presented earlier can be adapted to problems involving a team of unmanned aircraft monitoring a region of interest.
- Chapter 12: *Integrating Automation with Humans* concludes the book with a summary of various challenges of integrating decision support systems with human operators and provides strategies for effective implementation.

Each chapter has its own bibliographical section, and the end of the book contains an index of important terms and acronyms.

1.5 Further Reading

The artificial intelligence textbook from Russell and Norvig titled *Artificial Intelligence: A Modern Approach* provides an excellent introductory overview of different methods for building intelligent agents [1]. This book focuses primarily on planning and reinforcement learning, and the theory chapters provide additional references for further reading in those areas. Of course, if the problem is simple enough to be solved without resorting to such advanced methods, then explicitly programming the decision-making agents using some agent-oriented programming language might be best [2]. If an expert can show the system how to behave in different situations, then supervised learning may be appropriate. Many recent books cover supervised learning in great depth from a probabilistic perspective [3]–[5]. Generic optimization methods are covered by several textbooks [6]–[9]. Deterministic planning approaches are summarized in the books *Automated Planning: Theory and Practice* [10] and *Planning Algorithms* [11]. The two example applications discussed in this chapter will be revisited in Chapters 10 and 11.

References

1. S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Upper Saddle River, NJ: Pearson, 2010.
2. Y. Shoham, “Agent-Oriented Programming,” *Artificial Intelligence*, vol. 60, no. 1, pp. 51–92, 1993. doi: 10.1016/0004-3702(93)90034-9.
3. D. Barber, *Bayesian Reasoning and Machine Learning*. New York: Cambridge University Press, 2012.

4. K.P. Murphy, *Machine Learning: A Probabilistic Perspective*. Cambridge, MA: MIT Press, 2012.
5. C.M. Bishop, *Pattern Recognition and Machine Learning*. New York: Springer, 2006.
6. A.D. Belegundu and T.R. Chandrupatla, *Optimization Concepts and Applications in Engineering*, 2nd ed. New York: Cambridge University Press, 2011.
7. S. Boyd and L. Vandenberghe, *Convex Optimization*. New York: Cambridge University Press, 2004.
8. D. Bertsimas and J.N. Tsitsiklis, *Introduction to Linear Optimization*. Belmont, MA: Athena Scientific, 1997.
9. E.K.P. Chong and S.H. Zak, *An Introduction to Optimization*, 4th ed. Hoboken, NJ: Wiley, 2013.
10. M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: Theory and Practice*. San Francisco: Morgan Kaufmann, 2004.
11. S.M. LaValle, *Planning Algorithms*. New York: Cambridge University Press, 2006.

I

THEORY

2

Probabilistic Models

Mykel J. Kochenderfer

Rational decision making requires reasoning about one's uncertainty and objectives. This chapter focuses on the representation of uncertainty as a probability distribution. Real-world problems require reasoning about distributions over many different variables. We will discuss how to construct these models and how to use them to make inferences.

2.1 Representation

Uncertainty can arise from incomplete information about the state of the world. Suppose, for example, we are monitoring a satellite orbiting the earth thousands of kilometers away. Our satellite has been sending mission and telemetry data down to us reliably for months, but all of a sudden we lose our communication feed. Many different events could lead to a communication loss, such as an electrical power system or communications system failure on board the satellite, or it could be due to a failure of a system on the ground used for monitoring the satellite. Given the information that we have at hand, it is impossible to make a diagnosis with complete certainty.

Uncertainty can also arise from practical and theoretical limitations in our ability to predict future events. For example, predicting exactly how a human operator will respond to advice from a decision support system would require, among other things, a detailed model of how the human brain works. Even the paths of satellites can be difficult to predict. Although Newtonian physics permit highly precise predictions of satellite trajectories, spontaneous failures in the attitude thrusters can result in large deviations from the nominal path.

A robust decision-making system must account for these sources of uncertainty in the current state of the world and the future outcomes of events. Accounting for uncertainty computationally requires a formal representation.

2.1.1 Degrees of Belief and Probability

In problems of uncertainty, it is essential to be able to compare the plausibility of different statements. We would like to be able to represent, for example, that our belief that “there is an electrical anomaly on our satellite” is stronger than our belief that “there is a thruster anomaly on our satellite.” If E represents the proposition “there is an electrical anomaly on our satellite” and T represents the proposition “there is a thruster system anomaly on our satellite,” then we would write $E \succ T$. If we hold E and T with the same degree of belief, then we write $E \sim T$.

It is also useful to be able to compare our beliefs about statements given some information. For example, we would like to say, “there is an electrical anomaly given a communication loss” is more likely than “there is a thruster anomaly given a communication loss.” If C represents communication loss, then we would write $(E | C) \succ (T | C)$.

We want to make certain assumptions about the relationships induced by the operators $\succ$ and $\sim$. The assumption of *universal comparability* requires exactly one of the following to hold: $(A | C) \succ (B | C)$, $(A | C) \sim (B | C)$, or $(A | C) \prec (B | C)$. *Transitivity* requires that if $(A | D) \succeq (B | D)$ and $(B | D) \succeq (C | D)$ then $(A | D) \succeq (C | D)$. Universal comparability and transitivity assumptions lead to an ability to represent degrees of belief by a real-valued function [1]. In other words, we can use a function P that has the following two properties:

$$\begin{aligned} P(A | C) &> P(B | C) \text{ if and only if } (A | C) \succ (B | C) \\ P(A | C) &= P(B | C) \text{ if and only if } (A | C) \sim (B | C). \end{aligned}$$

If we make a set of additional assumptions about the form of P , then we can show that P must satisfy the basic axioms of probability. Hence, $0 \leq P(A | B) \leq 1$. If we are certain of $(A | B)$, then $P(A | B) = 1$. If we believe $(A | B)$ is impossible, then $P(A | B) = 0$. Uncertainty in the truth of $(A | B)$ is represented by values in between the two extrema.

This book does not provide a comprehensive review of probability theory, but we will restate two important properties of probabilities. The first is the definition of *conditional probability*, which states that

$$P(A | B) = \frac{P(A, B)}{P(B)}, \quad (2.1)$$

where $P(A, B)$ represents the probability of A and B both being true. The second important property is the *law of total probability*, which requires that if $\mathcal{B}$ is a set of mutually exclusive and exhaustive propositions, then

$$P(A | C) = \sum_{B \in \mathcal{B}} P(A | B, C) P(B | C). \quad (2.2)$$

It is easy to show from the definition of conditional probability that the following holds:

$$P(A | B) = \frac{P(B | A)P(A)}{P(B)}. \quad (2.3)$$

This equation is known as *Bayes' rule*, and it will play an important role in this book.

2.1.2 Probability Distributions

Suppose we have a binary random variable A that can assume one of two different values: 0 or 1. The probability distribution associated with A specifies the probabilities of the various values that can be assigned to A , in particular $P(A = 0)$ and $P(A = 1)$. We will use lowercase letters and superscripts as shorthand when discussing the assignment of values to random variables. For example, $P(a^0)$ is shorthand for $P(A = 0)$. The distribution $P(A)$ is defined by the values of $P(a^0)$ and $P(a^1)$, but the distribution can be represented with only a single independent parameter, $P(a^0)$, because $P(a^1) = 1 - P(a^0)$. If A is a discrete random variable that can assume one of n different values, then $P(A)$ can be defined by $n - 1$ parameters, namely, $P(a^1), \dots, P(a^{n-1})$, because $P(a^n) = 1 - (P(a^1) + \dots + P(a^{n-1}))$.

If A is a continuous random variable, then representing the distribution over A is a little less straightforward. The probability of A taking on any particular value is infinitesimally small. Consider the *uniform distribution* $\mathcal{U}(0, 10)$, which assigns equal probability to all values in the range $(0, 10)$. The probability that a random sample from this distribution is equal to the constant π is essentially zero. However, we can define non-zero probabilities for samples being within some interval, say $(3, 4)$. If $P(A) = \mathcal{U}(0, 10)$, then the probability that a sample a lies between 3 and 4 is $1/10$.

Distributions over continuous variables can be represented using either a *cumulative distribution function* or a *probability density function*. A cumulative distribution function specifies the probability mass associated with values below some threshold. If $p(a)$ is a probability density function over A , then $p(a)da$ is the probability A falls within the interval $(a, a + da)$ as $da \rightarrow 0$. A cumulative distribution function P can be defined in terms of a probability density as follows:

$$P(a) = \int_{-\infty}^a p(a) da. \quad (2.4)$$

Suppose we wanted to represent the distribution over the altitude of aircraft in the terminal region around New York City's JFK airport using a density function $p(a)$. We would first have to choose a form of the distribution and then specify its parameters. A common distribution for continuous variables is the *Gaussian distribution* (also called the *normal distribution*). The Gaussian distribution is parameterized by the mean μ and

variance σ^2 :

$$p(w) = \mathcal{N}(w \mid \mu, \sigma^2). \quad (2.5)$$

We use $\mathcal{N}(\mu, \sigma^2)$ to represent the Gaussian distribution with parameters μ and σ^2 and $\mathcal{N}(w \mid \mu, \sigma^2)$ to represent the density at w as given by

$$\mathcal{N}(w \mid \mu, \sigma^2) = \frac{1}{\sigma} \phi\left(\frac{w - \mu}{\sigma}\right). \quad (2.6)$$

The function ϕ above is the *standard normal density function*:

$$\phi(x) = \frac{1}{\sqrt{2\pi}} \exp(-x^2/2). \quad (2.7)$$

Although a Gaussian distribution is often convenient because it is defined by only two parameters and makes computation easy, it has some limitations, especially for representing altitude distributions. It assigns non-zero probability to negative altitudes, which of course must be positive. It also assigns non-zero probabilities to aircraft flying at unrealistically high altitudes. These problems can be remedied by bounding the *support*, that is, the range of values assigned non-zero probabilities, resulting in a *truncated Gaussian distribution* with density function given by

$$\mathcal{N}(w \mid \mu, \sigma^2, a, b) = \frac{\frac{1}{\sigma} \phi\left(\frac{w - \mu}{\sigma}\right)}{\Phi\left(\frac{b - \mu}{\sigma}\right) - \Phi\left(\frac{a - \mu}{\sigma}\right)}, \quad (2.8)$$

when w is within the interval $[a, b]$. The function Φ is the *standard normal cumulative distribution function* as given by

$$\Phi(x) = \int_{-\infty}^x \phi(x) dx. \quad (2.9)$$

In our aircraft altitude example, we might truncate below 0 ft and above 65,000 ft.

Another limitation of using a Gaussian distribution is that it is *unimodal*, meaning that there is a point in the distribution at which the density increases monotonically on one side and decreases monotonically on the other side. The distribution over altitudes in the JFK region is not unimodal. The top-left plot in Figure 2.1 shows the probability distribution of aircraft between 2000 ft and 10,000 ft, as estimated from 18 million radar reports during August 2011 in the JFK terminal region. Peaks every 1000 ft are due to the airspace structure. Clearly, using a Gaussian distribution would not be appropriate.

There are different ways to represent continuous distributions that are multimodal. One way is to mix together a collection of unimodal distributions. A *Gaussian mixture*

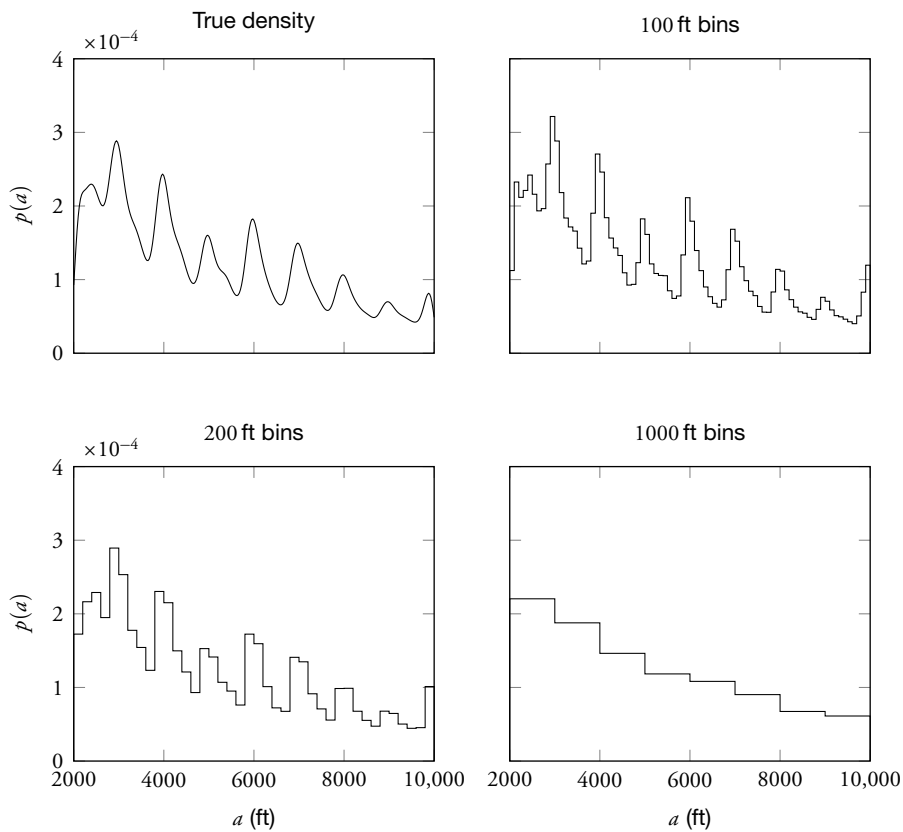


Figure 2.1 Modeling altitude distribution using different bin sizes.

Table 2.1 Example joint distribution.

A	B	C	$P(A, B, C)$
0	0	0	0.08
0	0	1	0.15
0	1	0	0.05
0	1	1	0.10
1	0	0	0.14
1	0	1	0.18
1	1	0	0.19
1	1	1	0.11

model (GMM) is simply a weighted average of different Gaussian distributions. The parameters of a Gaussian mixture model include the parameters of the Gaussian distribution components $\mu_1, \sigma_1^2, \dots, \mu_n, \sigma_n^2$ as well as their weights $\rho_1, \dots, \rho_n$. The density is given by

$$p(x \mid \mu_1, \sigma_1^2, \dots, \mu_n, \sigma_n^2, \rho_1, \dots, \rho_n) = \sum_{i=1}^n \rho_i \mathcal{N}(x \mid \mu_i, \sigma_i^2), \quad (2.10)$$

with the constraint that the weights sum to 1. If we were to use a Gaussian mixture model to represent the altitude distribution, then we might use Gaussian components centered at the peaks and assign appropriate weights.

Another approach to representing multimodal continuous distributions is through discretization. For example, we could create altitude bins every 100 ft and represent the distribution as a *piecewise-uniform density*, as shown in Figure 2.1. The density is specified by the bin edges, and the probability mass is associated with each bin. Figure 2.1 also shows the impact that different discretization schemes have on the representation of the altitude distribution. Although 200 ft bins may be acceptable for representing the altitude distribution, 1000 ft bins lose important features of the distribution.

2.1.3 Joint Distributions

One of the challenges when representing uncertainty in real-world problems is handling joint distributions over many variables. For now, let us assume we want to model the joint distribution over binary variables A , B , and C . An example distribution is shown in Table 2.1.

Table 2.1 contains $2^3 = 8$ entries specifying probabilities for every possible assignment of values to the three variables. Because we enumerated every possible assignment, the probabilities in the table sum to 1. Although there are eight entries in the table, only

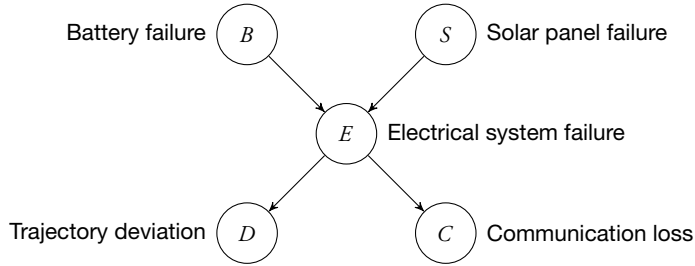


Figure 2.2 Example Bayesian network structure.

seven of them are *independent*. If θ_i represents the probability in the i th row in the table, then we only need the parameters $\theta_1, \dots, \theta_7$ to represent the distribution because we know $\theta_8 = 1 - (\theta_1 + \dots + \theta_7)$.

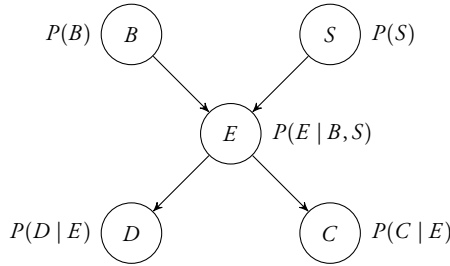
If we have n binary variables, then we need as many as $2^n - 1$ independent parameters to specify the joint distribution. This exponential growth in the number of parameters makes representing uncertainty and learning probabilistic models difficult.

2.1.4 Bayesian Network Representation

A *Bayesian network* is a compact representation of a joint distribution. The structure of the network is represented as a graph consisting of nodes and directed edges. Each node corresponds to a random variable. Directed edges (sometimes called arrows) connect pairs of nodes, with cycles in the graph being prohibited. The arrows indicate direct probabilistic relationships. Associated with each node X_i is a conditional distribution $P(X_i \mid \text{Pa}_{X_i})$, where Pa_{X_i} represents the parents of X_i in the graph.

Figure 2.2 shows an example Bayesian network for a satellite-monitoring problem involving five binary variables. Fortunately, battery failure and solar panel failures are both rare, although solar panel failures are somewhat more likely than battery failures. Failures in either can lead to an electrical system failure. There may be causes of electrical system failure other than battery or solar panel failure, such as a problem with the power management unit. An electrical system failure can result in trajectory deviation, which can be observed from the earth by telescope, as well as a communication loss that interrupts the transmission of telemetry and mission data down to various ground stations. Other anomalies not involving the electrical system can result in trajectory deviation and communication loss.

Associated with each of the five variables are five conditional probability distributions, as shown in Figure 2.3. Because B and S do not have any parents, we only need to specify $P(B)$ and $P(S)$. The distribution $P(B)$ can be specified by using a single

**Figure 2.3** Example Bayesian network conditional probabilities.**Table 2.2** Example conditional distribution.

E	B	S	$P(E B, S)$
0	0	0	0.90
0	0	1	0.05
0	1	0	0.03
0	1	1	0.01
1	0	0	0.10
1	0	1	0.95
1	1	0	0.97
1	1	1	0.99

independent parameter $P(b^0)$, and the distribution $P(S)$ can be specified by using a single independent parameter $P(s^0)$.

The node associated with E has two parents, B and S . Table 2.2 represents $P(E | B, S)$ and has 2^3 rows. Only half of these rows are needed to specify the distribution due to the constraint that $P(e^1 | b, s) = 1 - P(e^0 | b, s)$, where b and s represent any assignment to B and S . The other two conditional probability tables $P(D | E)$ and $P(C | E)$ can each be represented by two independent parameters. When the variables are binary, $P(X | \text{Pa}_X)$ can be represented by 2^n independent parameters, where n is the number of parents of X .

The *chain rule* for Bayesian networks specifies how to construct a joint distribution from the *local* conditional probability distributions. Suppose we have the variables $X_1, \dots, X_n$ and want to compute the probability of a particular assignment of all these variables to values $P(x_1, \dots, x_n)$. We let pa_{x_i} represent the particular assignment of the

parents of X_i to their values. The chain rule says

$$P(x_1, \dots, x_n) = \prod_{i=1}^n P(x_i \mid \text{pa}_{x_i}). \quad (2.11)$$

In the satellite example, suppose we want to compute the probability that nothing is going wrong, that is, $P(b^0, s^0, e^0, d^0, c^0)$. From the chain rule,

$$P(b^0, s^0, e^0, d^0, c^0) = P(b^0)P(s^0)P(e^0 \mid b^0, s^0)P(d^0 \mid e^0)P(c^0 \mid e^0). \quad (2.12)$$

If we had fully specified a joint distribution over the five variables B , S , E , D , and C , then we would need $2^5 - 1 = 31$ independent parameters. The structure assumed in our Bayesian network allows us to specify the joint distribution using only $1 + 1 + 4 + 2 + 2 = 10$ independent parameters. The difference between 10 and 31 does not represent an especially significant savings in the number of parameters, but the savings can become enormous in larger Bayesian networks. The power of Bayesian networks comes from their ability to reduce the number of parameters required to specify a joint probability distribution.

2.1.5 Conditional Independence

The reason that a Bayesian network can represent joint distributions with fewer independent parameters than would normally be required is due to the *conditional independence* assumptions encoded in its graphical structure. If the conditional independence assumptions made by the Bayesian network are invalid, then we run the risk of not properly modeling the joint distribution, as will be discussed in Section 2.4.

We say that variables A and B are *independent* if and only if $P(A, B) = P(A)P(B)$. The assertion that A and B are independent is written $A \perp B$. From Equation (2.1), we see that $A \perp B$ if and only if $P(A) = P(A \mid B)$. In other words, information about B does not give us any additional information about A , and vice versa. For example, let us assume that a battery failure on our satellite (B) is independent of a solar panel failure (S). Hence, knowing that we have had a battery failure does not increase or decrease our belief about whether there has been a solar panel failure. We can specify the joint distribution $P(B, S)$ using just two parameters $P(b^0)$ and $P(s^0)$, as shown in Table 2.3. In fact, if we have n independent binary variables, then we can specify the joint distribution using only n independent parameters, as opposed to $2^n - 1$ independent parameters if we could not make the independence assumption.

Variables A and B are *conditionally independent* given C if and only if $P(A, B \mid C) = P(A \mid C)P(B \mid C)$. The assertion that A and B are conditionally independent given C is written $(A \perp B \mid C)$. It is possible to show from this definition that $(A \perp B \mid C)$ if and only if $P(A \mid C) = P(A \mid B, C)$. Given C , information about B provides no additional

Table 2.3 Example joint distribution over independent variables.

B	S	$P(B, S)$
0	0	$P(b^0)P(s^0)$
0	1	$P(b^0)(1 - P(s^0))$
1	0	$(1 - P(b^0))P(s^0)$
1	1	$(1 - P(b^0))(1 - P(s^0))$

information about A , and vice versa. For example, let us assume that the presence of satellite trajectory deviation (D) is conditionally independent of whether we have a communication loss (C) given whether we have an electrical system failure (E). We would write this ($D \perp C \mid E$). If we know that we have an electrical system failure, then the fact that we observe a loss of communication has no impact on our belief that there is a trajectory deviation. We may have an elevated expectation that there is a trajectory deviation, but that is only because we know that an electrical system failure has occurred.

We can use a set of rules to determine whether nodes A and B are conditionally independent given a set of nodes $\mathcal{C}$. If ($A \perp B \mid \mathcal{C}$), then we say that $\mathcal{C}$ *d-separates* A and B (where the “d” stands for directional). We also say that a path between A and B is d-separated by $\mathcal{C}$ if any of the following are true:

1. The path contains a *chain* of nodes, $X \rightarrow Y \rightarrow Z$, such that Y is in $\mathcal{C}$.
2. The path contains a *fork*, $X \leftarrow Y \rightarrow Z$, such that Y is in $\mathcal{C}$.
3. The path contains an *inverted fork* (also called a *v-structure*), $X \rightarrow Y \leftarrow Z$, such that Y is *not* in $\mathcal{C}$ and no descendant of Y is in $\mathcal{C}$.

If all paths between A and B are d-separated by $\mathcal{C}$, then ($A \perp B \mid \mathcal{C}$). Sometimes the term *Markov blanket* is used to refer to the minimal set of nodes that d-separates a node from all other nodes.

In the network in Figure 2.2, there is only one v-structure, $B \rightarrow E \leftarrow S$. In the absence of information about E , D , or C , then B and S are independent. Given E , D , or C , however, B and S are no longer independent; influence can flow from B to S . For example, if we know that we have had an electrical system failure, then knowing that we have had a battery failure reduces our belief that there has been a solar panel failure. This kind of influence through a v-structure is sometimes called *explaining away* because the presence of a battery failure explains away the cause of the electrical system failure.

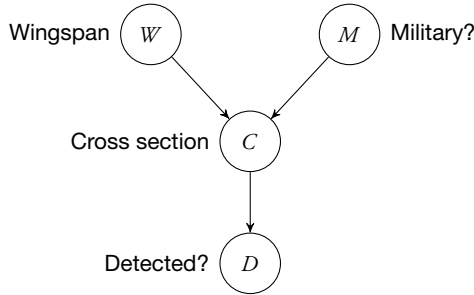


Figure 2.4 Example hybrid Bayesian network.

2.1.6 Hybrid Bayesian Networks

The examples in this chapter so far have involved binary variables, but Bayesian networks can contain a mixture of both discrete and continuous variables. Bayesian networks with discrete and continuous variables are often called *hybrid Bayesian networks*. Figure 2.4 shows an example hybrid Bayesian network representing the relationship among characteristics of aircraft, their radar cross section, and the ability of a radar to detect a target. Aircraft with larger wingspans tend to have larger radar cross sections, where cross section is measured using a decibel measure relative to one square meter (dBsm). Military aircraft are sometimes designed to have lower radar cross sections (below 0 dBsm) to escape detection. Targets that have larger cross sections are more likely to be detected, although other factors might influence detection. In Figure 2.4, the variables M and D are naturally binary, and the variables W and C are naturally continuous.

As with any Bayesian network, we need to specify the conditional distributions for each node. The node W has no parents, so we just need to specify a distribution over W . We will use a Gaussian distribution defined by the parameters μ and σ^2 as discussed in Section 2.1.2, although it may assign small probabilities to negative wingspans and unrealistically large wingspans. The variable M is binary, so we can define that distribution using a single parameter specifying $P(m^0)$.

The cross section C depends on both the continuous variable W and the binary variable M . For the moment, we will ignore the dependence on M and define a density $p(c | w)$. A common approach for defining a distribution over a continuous variable, given another continuous variable, is to use a *linear Gaussian* distribution. For example,

$$p(c | w) = \mathcal{N}(c | \theta_1 w + \theta_2, \theta_3). \quad (2.13)$$

As can be seen above, the mean is a linear function of w defined by parameters θ_1 and θ_2 . The variance is defined by θ_3 . Because we want larger wingspans to result in larger cross sections, we should be sure to make θ_1 positive. Aircraft with infinitesimally small

wingspans will also have infinitesimally small cross sections, and so θ_2 should probably be 0. The parameter θ_3 controls the amount of variance in the linear relationship between c and w .

In reality, C depends on both W and M . We can simply make the parameters used in the linear Gaussian distribution dependent on M :

$$P(c \mid w, m) = \begin{cases} \mathcal{N}(c \mid \theta_1 w + \theta_2, \theta_3) & \text{if } m^0 \\ \mathcal{N}(c \mid \theta_4 w + \theta_5, \theta_6) & \text{if } m^1 \end{cases}. \quad (2.14)$$

This kind of distribution is known as *conditional linear Gaussian*. In this example, we require six parameters to represent $p(c \mid w, m)$. Because military aircraft are more likely to be designed to have lower radar cross section than nonmilitary aircraft, we would want θ_4 to be smaller than θ_1 .

Finally, we need to define the conditional distribution $P(D \mid C)$. We want to capture the property that our radar is more likely to detect aircraft with larger cross sections. Of course, we could just set a threshold θ and say $P(d^1 \mid c) = 0$ if $c < \theta$ and $P(d^1 \mid c) = 1$ otherwise. However, such a model would potentially assign zero probability to detections that may actually occur.

Instead of a hard threshold to define $P(D \mid C)$, we could use a soft threshold that assigns low probabilities when below a threshold and high probabilities above a threshold. One way to represent a soft threshold is to use a *logit model*, which produces a *sigmoid* curve having an “S” shape:

$$P(d^1 \mid c) = \frac{1}{1 + \exp\left(-2 \frac{c - \theta_1}{\theta_2}\right)}. \quad (2.15)$$

The parameter θ_1 governs the location of the threshold, and θ_2 controls the “softness” or spread of the probabilities. Figure 2.5 shows $P(d^1 \mid c)$ with $\theta_1 = 0$ and $\theta_2 = 1$ as a solid line.

An alternative to the logit model is the *probit model*:

$$P(d^1 \mid c) = \Phi(c - \theta_1)/\theta_2, \quad (2.16)$$

where Φ is the standard normal cumulative distribution function as introduced in Section 2.1.2. The logit model corresponds closely to the probit model, as shown in Figure 2.5.

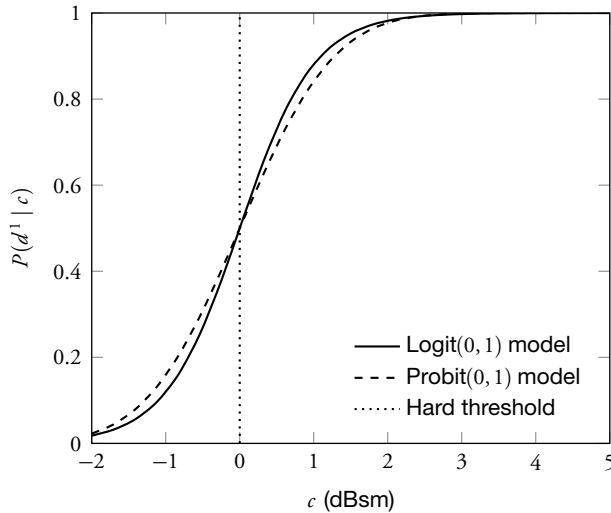


Figure 2.5 Hard and soft thresholds.

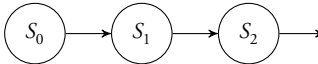


Figure 2.6 Markov chain.

2.1.7 Temporal Models

A *temporal model* represents how a set of variables evolves over time. A simple temporal model is a *Markov chain*, where the state at time t is denoted S_t . A Markov chain can represent, for example, the position and velocity of an aircraft over time. Figure 2.6 shows the structure of a Bayesian network representing a Markov chain. Only the first three states are shown in the figure, but a Markov chain can continue indefinitely. The initial distribution is given by $P(S_0)$. The conditional distribution $P(S_t | S_{t-1})$ is often referred to as the *state transition model*. If the state transition distribution does not vary with t , then the model is called *stationary*.

The state in a Markov chain does not have to be scalar. For example, if we want to model the random behavior of an aircraft over time, the state might be a vector $\mathbf{s} = (h, \dot{h})$, where h is the altitude of the aircraft and $\dot{h}$ is the vertical rate. The initial distribution $P(S_0)$ may be represented by a *multivariate Gaussian distribution* parameterized by mean

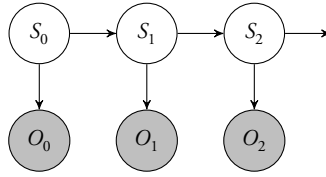


Figure 2.7 Hidden Markov model.

vector μ and covariance matrix Σ with density given by

$$p(\mathbf{s}) = \mathcal{N}(\mathbf{s} \mid \mu, \Sigma), \quad (2.17)$$

where $\mathcal{N}(\mathbf{s} \mid \mu, \Sigma)$ is a k -dimensional generalization of the Gaussian distribution in Equation (2.6):

$$\mathcal{N}(\mathbf{s} \mid \mu, \Sigma) = \frac{1}{(2\pi)^{k/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{s} - \mu)^\top \Sigma^{-1}(\mathbf{s} - \mu)\right). \quad (2.18)$$

For our aircraft model, k is two, μ is a vector with two elements, and Σ is a two-by-two matrix.

The state transition distribution for our aircraft model can be represented as a linear Gaussian as follows:

$$p(\mathbf{s}_t \mid \mathbf{s}_{t-1}) = \mathcal{N}(\mathbf{s}_t \mid \mathbf{M}\mathbf{s}_{t-1} + \mathbf{b}, \Sigma). \quad (2.19)$$

The mean is simply a linear function of the previous state. If aircraft continue straight on average, then a sensible choice for the mean is

$$\mathbf{M}\mathbf{s}_{t-1} + \mathbf{b} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \mathbf{s}_{t-1}. \quad (2.20)$$

The covariance matrix Σ controls the amount of randomness applied to the altitude and vertical rate of the aircraft.

Markov chains can be extended by adding observation nodes, as shown in Figure 2.7. The observation at time t is denoted O_t . The observation nodes are shaded to indicate that the values at those nodes are known. If the states correspond to the position and velocity of an aircraft, then the observations may be noisy radar measurements of the range and azimuth. If the state variables are discrete, then the model is called a *hidden Markov model* (HMM). If the state variables are continuous and the conditional distributions are linear Gaussian, then the model is called a *linear dynamical system*.

An example of a linear dynamical system is an extension of the aircraft model whose true state at time t is represented by the vector $(h_t, \dot{h}_t)$. Our observations are

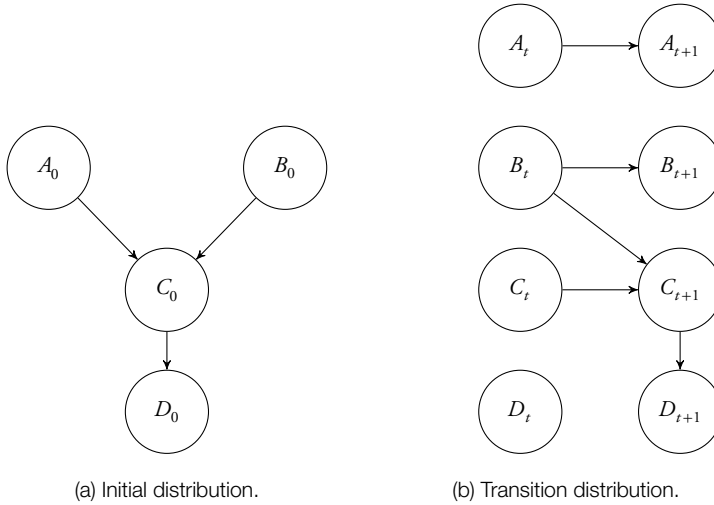


Figure 2.8 Dynamic Bayesian network.

noisy measurements of the altitude; the vertical rate cannot be observed directly. The observation at time t is modeled as coming from a linear Gaussian distribution:

$$p(\mathbf{o}_t \mid \mathbf{s}_t) = \mathcal{N}\left(\mathbf{o}_t \mid \begin{bmatrix} 1 & 0 \end{bmatrix} \mathbf{s}_t, \Sigma\right). \quad (2.21)$$

The covariance matrix Σ , in this case a single-element matrix, controls the measurement noise.

Stationary temporal models involving multiple state variables can be compactly represented using a *dynamic Bayesian network*. A dynamic Bayesian network is composed of two Bayesian networks, one representing the initial distribution and the other representing the transition distribution. The transition distribution is represented by a Bayesian network with two slices. The first slice represents the variables at time t , and the second slice represents the variables at time $t + 1$. Figure 2.8 shows an example dynamic Bayesian network with four state variables.

2.2 Inference

The previous section explained how to represent probability distributions. We now discuss how to use these probabilistic representations to perform *inference*. Inference involves determining the distribution over one or more unobserved variables given the values associated with a set of observed variables. For example, suppose we want to

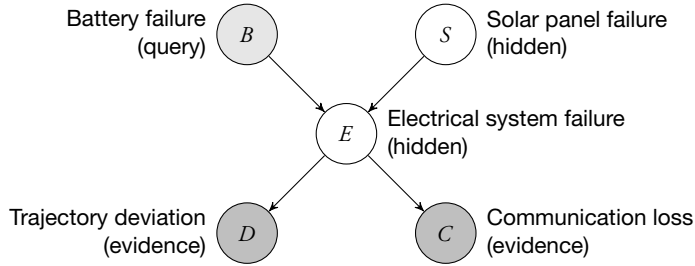


Figure 2.9 Query, evidence, and hidden variables in a Bayesian network.

infer the distribution $P(B \mid d^1, c^1)$ using the satellite Bayesian network introduced in Section 2.1.4. In this case, B is the *query* variable, D and C are the *evidence* variables, and S and E are the *hidden* variables. After a discussion of a couple examples in which inference can be helpful, this section explains how to leverage the structure inherent in a Bayesian network to make efficient inferences.

2.2.1 Inference for Classification

Inference can be used for *classification* tasks, where we want to infer the class given a set of observations or features. For example, suppose we want to determine whether a radar target is either a bird or an aircraft given properties of the radar track. In this case, the class is either bird or aircraft, and the observations might include measurements of the velocity and the amount of fluctuation in the heading over the duration of the track. Most aircraft travel faster than most birds, but there is some overlap, especially with smaller, lower performance aircraft. Migrating birds tend to maintain their heading, in contrast to maneuvering aircraft.

A simple probabilistic model often used in classification tasks is the *naive Bayes* model, which has the structure shown in Figure 2.10. An equivalent but more compact representation is shown in Figure 2.11 using a *plate*, shown as a rounded box. The $i = 1 : n$ in the bottom of the box specifies that the i in the subscript of the variable name is repeated from 1 to n .

In the naive Bayes model, the class C is the query variable, and the observed features $O_1, \dots, O_n$ are the evidence variables. For compactness throughout this book, we will use colon notation occasionally in subscripts. For example, $O_{1:n}$ is a compact way to write $O_1, \dots, O_n$. The naive Bayes model is called naive because it assumes conditional independence between the evidence variables given the class. Using the notation introduced in Section 2.1.5, we can say $(O_i \perp O_j \mid C)$ for all $i \neq j$. Of course, if these conditional independence assumptions do not hold, then we can add the necessary directed edges between the observed features.

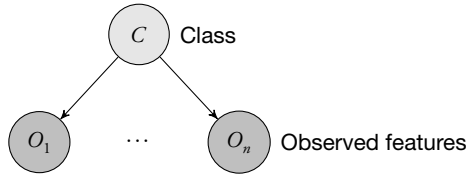


Figure 2.10 Inference for classification using a naive Bayes model.

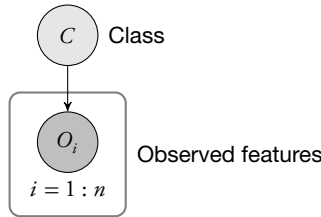


Figure 2.11 Plate representation of a naive Bayes model.

In the naive Bayes model, we have to specify the *prior* $P(C)$ and the *class-conditional distribution* $P(O_i | C)$. In the radar target classification problem, the prior represents our belief about whether a target is a bird or an aircraft in the absence of any information about the track. Figure 2.12 shows example class-conditional distributions for airspeed as estimated from radar data.

We can apply the chain rule in Equation (2.11) to infer the joint distribution in a naive Bayes model:

$$P(c, o_{1:n}) = P(c) \prod_{i=1}^n P(o_i | c). \quad (2.22)$$

What we really want for our classification task is the conditional probability $P(c | o_1, \dots, o_n)$. From the definition of conditional probability in Equation (2.1), we have

$$P(c | o_{1:n}) = \frac{P(c, o_{1:n})}{P(o_{1:n})}. \quad (2.23)$$

We can easily compute the denominator using the joint distribution and the law of total probability in Equation (2.2):

$$P(o_{1:n}) = \sum_c P(c, o_{1:n}). \quad (2.24)$$

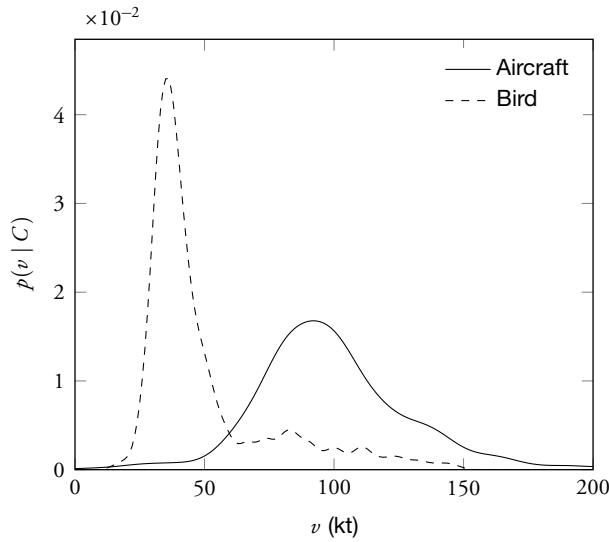


Figure 2.12 Airspeed probability density given target class.

The denominator in Equation (2.23) is not a function of C and can therefore be treated as a constant. Hence, we can write

$$P(c \mid o_{1:n}) = \kappa P(c, o_{1:n}), \quad (2.25)$$

where κ is the *normalization constant* such that $\sum_c P(c \mid o_{1:n}) = 1$. We often drop the κ and simply write

$$P(c \mid o_{1:n}) \propto P(c, o_{1:n}), \quad (2.26)$$

where the symbol “ $\propto$ ” is used to represent that the left-hand side is “proportional to” the right-hand side. For example, suppose from the chain rule we determine:

$$P(\text{bird, slow, little heading fluctuation}) = 0.03 \quad (2.27)$$

$$P(\text{aircraft, slow, little heading fluctuation}) = 0.01. \quad (2.28)$$

Of course, these probabilities do not sum to 1. If we want to determine the probability that a target is a bird given the evidence, then we would make the following calculation:

$$P(\text{bird} \mid \text{slow, little heading fluctuation}) = \frac{0.03}{0.03 + 0.01} = 0.75. \quad (2.29)$$

Using our model and applying the laws of probability, we have determined that the probability of our target being a bird is 0.75 and the probability of it being an aircraft is 0.25, but for many applications, we have to commit to a particular class. A common way to determine a classification is to select the class that has the highest *posterior probability*, that is, the one with the highest probability after taking the evidence into account. However, choosing a class is really a decision problem that often should take into account the consequences of misclassification. For example, if we are interested in using our classifier to filter out targets that are not aircraft for the purpose of air traffic control, then we can afford to occasionally let a few birds and other clutter tracks through our filter. However, we would want to avoid filtering out any real aircraft because that could lead to a collision. In this case, we would probably only want to classify a track as a bird if the posterior probability were close to 1. Decision problems will be discussed in Chapter 3.

2.2.2 Inference in Temporal Models

Many important applications, such as speech recognition, aircraft tracking, and cryptanalysis, involve performing inference in temporal models. Four common inference tasks include:

- *Filtering*: $P(S_t \mid O_{0:t})$
- *Prediction*: $P(S_{t'} \mid O_{0:t})$ where $t' > t$
- *Smoothing*: $P(S_{t'} \mid O_{0:t})$ where $t' < t$
- *Most likely explanation*: $\arg \max_{S_{0:t}} P(S_{0:t} \mid O_{0:t})$

The items above use the hidden Markov structure and notation shown in Figure 2.7. The variable t represents the current time.

To illustrate inference in temporal models, we will focus on filtering in a hidden Markov model with discrete state and observation variables. By Bayes' rule,

$$P(s_t \mid o_{0:t}) \propto P(o_t \mid s_t, o_{0:t-1})P(s_t \mid o_{0:t-1}). \quad (2.30)$$

The structure of the Bayesian network representing a hidden Markov model allows us to make the conditional independence assumption ($O_t \perp O_{0:t-1} \mid S_t$), which implies that $P(o_t \mid s_t, o_{0:t-1})$ in the equation above is equal to $P(o_t \mid s_t)$. Rewriting Equation (2.30) and applying the law of total probability to the second term, we get

$$P(s_t \mid o_{0:t}) \propto P(o_t \mid s_t) \sum_{s_{t-1}} P(s_t, s_{t-1} \mid o_{0:t-1}). \quad (2.31)$$

Applying the definition of conditional probability to $P(s_t, s_{t-1} \mid o_{0:t-1})$, we get

$$P(s_t \mid o_{0:t}) \propto P(o_t \mid s_t) \sum_{s_{t-1}} P(s_t \mid s_{t-1}, o_{0:t-1})P(s_{t-1} \mid o_{0:t-1}). \quad (2.32)$$

The structure of the model has that $(s_t \perp o_{0:t-1} \mid s_{t-1})$, and so we can simplify the equation above:

$$P(s_t \mid o_{0:t}) \propto P(o_t \mid s_t) \sum_{s_{t-1}} P(s_t \mid s_{t-1}) P(s_{t-1} \mid o_{0:t-1}). \quad (2.33)$$

We know $P(o_t \mid s_t)$ and $P(s_t \mid s_{t-1})$ directly from the model. The probability $P(s_{t-1} \mid o_{0:t-1})$ on the right-hand side suggests how one would go about recursively updating the state distribution with the progression of time and new observations. Algorithm 2.1 shows how this process, called *recursive Bayesian estimation*, is done. The posterior state distribution at time t is denoted b_t . To reduce the number of subscripts, Algorithm 2.1 assumes that the state transition distribution $P(S_t \mid S_{t-1})$ and observation distribution $P(O_t \mid S_t)$ are stationary; that is, they do not vary with time.

Algorithm 2.1 Recursive Bayesian estimation

```

1: function RECURSIVEBAYESIANESTIMATION
2:    $b_0(s) \leftarrow P(o_0 \mid s)P(s_0)$  for all  $s$ 
3:   Normalize  $b_0$ 
4:   for  $t \leftarrow 1$  to  $\infty$ 
5:      $b_t(s) \leftarrow P(o_t \mid s) \sum_{s'} P(s \mid s') b_{t-1}(s')$  for all  $s$ 
6:     Normalize  $b_t$ 

```

If observations are continuous instead of discrete, then $P(o \mid s)$ will be a probability density instead of a probability mass. If states are continuous instead of discrete, then the state transition distribution and b become density functions and the summation on Line 5 becomes an integral, which in general can be difficult to evaluate exactly.

Filtering in linear dynamical systems, which assume the state transition and observation distributions are linear Gaussian, can actually be done exactly. If b_{t-1} is represented as a normal distribution, then it can be shown that the integration on Line 5 results in the posterior b_t being Gaussian. The *Kalman filter* is a well-known filter for linear dynamical systems that simply updates the mean and covariance of b_t appropriately.

2.2.3 Exact Inference

Let us revisit the network in Figure 2.9 and attempt to exactly infer $P(b^1 \mid d^1, c^1)$. By the definition of conditional probability, we know that

$$P(b^1 \mid d^1, c^1) = \frac{P(b^1, d^1, c^1)}{P(d^1, c^1)}. \quad (2.34)$$

We will focus on the numerator in this discussion because the process of computing the numerator can be applied to computing the denominator as well. By the law of total probability,

$$P(b^1, d^1, c^1) = \sum_s \sum_e P(b^1, s, e, d^1, c^1). \quad (2.35)$$

This process of summing out the hidden variables is called *marginalization*. By the chain rule,

$$P(b^1, d^1, c^1) = \sum_s \sum_e P(b^1)P(s)P(e | b^1, s)P(d^1 | e)P(c^1 | e). \quad (2.36)$$

The trouble with exact inference is that we have to sum out the hidden variables. In Equation (2.36), we only have to sum out two variables, but for larger networks, this summing may be infeasible. The number of terms to be added together can grow exponentially with the number of hidden variables, although for many Bayesian networks, we can take advantage of the structure of the model to make inference efficient.

As an extreme example of how a particular network representation can result in efficient inference, suppose we had a Bayesian network with binary variables, $X_1, \dots, X_n$, and there are no arrows in the network. We want to compute the following:

$$P(x_1^0) = \sum_{x_2} \cdots \sum_{x_n} P(x_1^0)P(x_2) \cdots P(x_n). \quad (2.37)$$

Here, there are 2^{n-1} terms, with each term consisting of the product of n factors. Of course, there is no reason to go to the effort of applying the chain rule to get the joint distribution and then applying the law of total probability to sum out the hidden variables. We know the probability $P(x_1^0)$ directly from the table specifying $P(X_1)$.

A variety of methods can be used to perform efficient inference in more complicated Bayesian networks. One method is known as *variable elimination*, which eliminates the hidden variables in sequence. We will illustrate the variable elimination algorithm by computing the distribution $P(B | d^1, c^1)$ for the Bayesian network in Figure 2.9. The conditional probability distributions associated with the nodes in the network can be represented by the following tables:

$$T_1(B), T_2(S), T_3(E, B, S), T_4(D, E), T_5(C, E). \quad (2.38)$$

Because D and C are observed variables, the last two tables are replaced with $T_6(E)$ and $T_7(E)$, keeping only the rows in which $D = 1$ and $C = 1$. We then proceed by eliminating the hidden variables in sequence. Different strategies can be used for choosing an ordering, but for this example, we use the ordering E and then S . To eliminate E , we gather all the tables involving E :

$$T_3(E, B, S), T_6(E), T_7(E). \quad (2.39)$$

We then sum out E from the product of these tables to get a new table:

$$T_8(B, S) = \sum_e T_3(e, B, S) T_6(e) T_7(e). \quad (2.40)$$

We can then discard T_3, T_6, T_7 because all the information we need from them is contained in T_8 . Now, we eliminate S . Again, we gather all the remaining tables that involve S and sum out S from the product of these tables:

$$T_9(B) = \sum_s T_2(s) T_8(B, s). \quad (2.41)$$

We discard T_2 and T_8 , and now we are left with $T_1(B)$ and $T_9(B)$. We have to normalize the product of these two tables to obtain $P(B \mid d^1, c^1)$.

Algorithm 2.2 outlines the variable elimination algorithm for a Bayesian network B , a set of query variables $\mathcal{Q}$, and observed values $\mathbf{o}$. For many networks, variable elimination allows inference to be done in an amount of time that scales linearly with the size of the network, but it has exponential time complexity in the worst case. What influences the amount of computation is the variable elimination order. Choosing the optimal elimination order, it turns out, is *NP-hard*, meaning that it cannot be done in polynomial time in the worst case (Section 2.2.4). Even if we found the optimal elimination order, variable elimination can still require an exponential number of computations. Variable elimination heuristics generally try to minimize the number of variables involved in the intermediate tables generated in Line 6.

Algorithm 2.2 Variable elimination in Bayesian networks

```

1: function VARIABLEELIMINATION( $B, \mathcal{Q}, \mathbf{o}$ )
2:    $\mathcal{T} \leftarrow$  set of conditional probability tables associated with nodes in  $B$ 
3:   Remove rows that are inconsistent with  $\mathbf{o}$  from all the tables in  $\mathcal{T}$ 
4:   for  $i \leftarrow 1$  to  $n$ 
5:      $\mathcal{T}' \leftarrow$  all the tables in  $\mathcal{T}$  that involve  $X_i$ 
6:      $T \leftarrow$  the product of the tables in  $\mathcal{T}'$  with  $X_i$  summed out
7:     Remove  $\mathcal{T}'$  from  $\mathcal{T}$  and add  $T$ 
8:    $T \leftarrow$  product of the tables remaining in  $\mathcal{T}$ 
9:    $P(\mathcal{Q} \mid \mathbf{o}) \leftarrow$  normalize  $T$ 
10:  return  $P(\mathcal{Q} \mid \mathbf{o})$ 

```

An approach to inference known as *belief propagation* works by propagating “messages” through the network. Belief propagation requires linear time but only provides an exact answer if the network does not have undirected cycles. If the network has undirected cycles, then it can be converted into a tree by combining multiple variables into single nodes by using what is known as the *junction tree algorithm*. If the number of variables

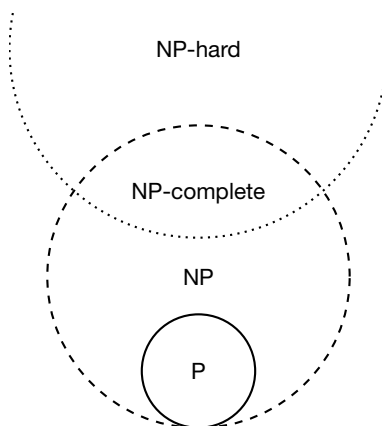


Figure 2.13 Complexity classes.

that have to be combined into any one node in the resulting network is small, then inference can be done efficiently.

2.2.4 Complexity of Exact Inference

The difficulty of solving certain problems can be grouped into certain complexity classes. Important classes that will appear frequently throughout this book include:

- *P*: problems that can be solved in polynomial time,
- *NP*: problems whose solutions can be verified in polynomial time,
- *NP-hard*: problems that are at least as hard as the hardest problems in NP, and
- *NP-complete*: problems that are both NP-hard and in NP.

Formal definitions of these complexity classes are rather involved. It is generally believed that $P \neq NP$, but it has not been proven and remains one of the most important open problems in mathematics. In fact, modern cryptography depends on the fact that there are no known efficient (i.e., polynomial time) algorithms for solving NP-hard problems. Figure 2.13 illustrates the relationship between the complexity classes under the assumption that $P \neq NP$.

A common approach to proving whether a particular problem Q is NP-hard is to come up with a polynomial transformation from a known NP-complete problem Q' to an instance of Q . We can show that inference in Bayesian networks is NP-hard by using an NP-complete problem called 3SAT. The 3SAT problem is the first known NP-complete problem. It involves determining whether a Boolean formula is *satisfiable*. The Boolean formula consists of conjunctions ($\wedge$), disjunctions ($\vee$), and negations ($\neg$) involving n Boolean variables $x_1, \dots, x_n$. A literal is a variable x_i or its negation $\neg x_i$. A 3SAT clause is a disjunction of up to three literals; for example, $x_3 \vee \neg x_5 \vee x_6$. A 3SAT

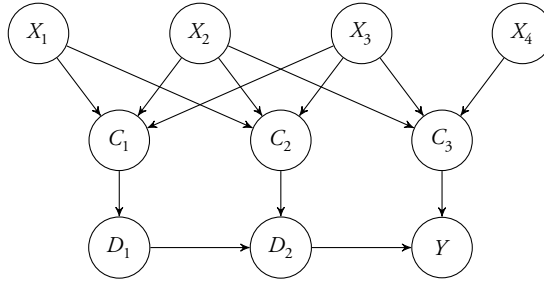


Figure 2.14 Bayesian network representing a 3SAT problem.

formula is a conjunction of 3SAT clauses like

$$F(x_1, x_2, x_3, x_4) = \left(\begin{array}{cccc} x_1 & \vee & x_2 & \vee & x_3 \\ \neg x_1 & \vee & \neg x_2 & \vee & x_3 \\ x_2 & \vee & \neg x_3 & \vee & x_4 \end{array} \right) \wedge \quad (2.42)$$

The challenge in 3SAT is to determine whether a possible assignment of truth values to variables exists that makes the formula true. In Equation (2.42),

$$F(\text{true}, \text{false}, \text{false}, \text{true}) = \text{true}. \quad (2.43)$$

Hence, the formula is satisfiable. Although a satisfying assignment can be easily found for some 3SAT problems, sometimes just by quick inspection, they are difficult to solve in general. One way to determine whether a satisfying assignment can be made is to enumerate the 2^n possible truth values of all the variables. Although determining whether a satisfying truth assignment exists is difficult, verification of whether a truth assignment leads to satisfaction can be done in linear time.

It is easy to construct a Bayesian network from an arbitrary 3SAT problem. Figure 2.14 is a Bayesian network representation of Equation (2.42). The variables are represented by $X_{1:4}$, and the clauses are represented by $C_{1:3}$. The distributions over the variables are uniform. The nodes representing clauses have as parents the participating variables. Because this is a 3SAT problem, each clause node has exactly three parents. Each clause node assigns probability 0 to assignments that do not satisfy the clause and probability 1 to all satisfying assignments. The remaining nodes assign probability 1 to true if all their parents are true. The original 3SAT problem is satisfiable if and only if $P(y^1) > 0$. Hence, inference in Bayesian networks is at least as hard as 3SAT.

The reason to go to the effort of showing that inference in Bayesian networks is NP-hard is so that we know to avoid wasting time looking for an efficient, exact inference algorithm that works on all Bayesian networks. Therefore, research over the past couple decades has focused on approximate inference methods, which are discussed next.

2.2.5 Approximate Inference

One of the simplest approaches to approximate inference involves sampling from the joint distribution represented by the Bayesian network. The first step involves finding a *topological sort* of the nodes in the Bayesian network. A topological sort of nodes in a directed acyclic graph is an ordered list such that if there is an edge $A \rightarrow B$, then A comes before B in the list. For example, a topological sort for the network in Figure 2.9 is B, S, E, D, C . A topological sort always exists, but it may not be unique. Another topological sort for the network in Figure 2.9 is S, B, E, C, D . Algorithm 2.3 provides an algorithm for finding a topological sort of a graph G .

Algorithm 2.3 Topological sort

```

1: function TOPOLOGICALSORT( $G$ )
2:    $n \leftarrow$  number of nodes in  $G$ 
3:    $L \leftarrow$  empty list
4:   for  $i \leftarrow 1$  to  $n$ 
5:      $X \leftarrow$  any node not in  $L$  but all of whose parents are in  $L$ 
6:     Add  $X$  to end of  $L$ 
7:   return  $L$ 

```

Once we have a topological sort, we can begin sampling from the conditional probability distributions. Suppose our topological sort results in the ordering $X_{1:n}$. Algorithm 2.4 shows how to sample from a Bayesian network B . In Line 4, we draw a sample from the conditional distribution associated with X_i given the values of the parents that have already been assigned. Because $X_{1:n}$ is a topological sort, we know that all the parents of X_i have already been instantiated, allowing this sampling to be done.

Algorithm 2.4 Direct sampling from a Bayesian network

```

1: function DIRECTSAMPLE( $B$ )
2:    $X_{1:n} \leftarrow$  a topological sort of nodes in  $B$ 
3:   for  $i \leftarrow 1$  to  $n$ 
4:      $x_i \leftarrow$  a random sample from  $P(X_i \mid \text{pa}_{x_i})$ 
5:   return  $x_{1:n}$ 

```

Table 2.4 shows ten random samples from the network in Figure 2.9. We are interested in inferring $P(b^1 \mid d^1, c^1)$. Only two of the ten samples (pointed to in the table) are consistent with the observations d^1 and c^1 . One sample has $B = 1$ and the other sample has $B = 0$. From these samples, we infer that $P(b^1 \mid d^1, c^1) = 0.5$. Of course, we would want to use more than just two samples to accurately estimate $P(b^1 \mid d^1, c^1)$.

Table 2.4 Direct samples from a Bayesian network.

B	S	E	D	C	
0	0	1	1	0	
0	0	0	0	0	
1	0	1	0	0	
1	0	1	1	1	←
0	0	0	0	0	
0	0	0	1	0	
0	0	0	0	1	
0	1	1	1	1	←
0	0	0	0	0	
0	0	0	1	0	

The problem with direct sampling is that we may waste a lot of time generating samples that are inconsistent with the observations, especially if the observations are unlikely. An alternative approach is called *likelihood weighting*, which involves generating weighted samples that are consistent with the observations. We begin with a topological sort and sample from the conditional distributions in sequence. The only difference in likelihood weighting is how we handle observed variables. Instead of sampling their values from a conditional distribution, we assign variables to their observed values and adjust the weight of the sample appropriately. The weight of a sample is simply the product of the conditional probabilities at the observed nodes. Algorithm 2.5 summarizes this process for a Bayesian network B and observations $o_{1:n}$. If o_i is not observed, then $o_i \leftarrow \text{NIL}$.

Algorithm 2.5 Likelihood-weighted sampling from a Bayesian network

```

1: function LIKELIHOODWEIGHTEDSAMPLE( $B, o_{1:n}$ )
2:    $X_{1:n} \leftarrow$  a topological sort of nodes in  $B$ 
3:    $w \leftarrow 1$ 
4:   for  $i \leftarrow 1$  to  $n$ 
5:     if  $o_i = \text{NIL}$ 
6:        $x_i \leftarrow$  a random sample from  $P(X_i \mid \text{pa}_{x_i})$ 
7:     else
8:        $x_i \leftarrow o_i$ 
9:        $w \leftarrow w \times P(x_i \mid \text{pa}_{x_i})$ 
10:  return  $(x_{1:n}, w)$ 

```

Table 2.5 Likelihood weighted samples from a Bayesian network.

B	S	E	D	C	Weight
1	0	1	1	1	$P(d^1 e^1)P(c^1 e^1)$
0	1	1	1	1	$P(d^1 e^1)P(c^1 e^1)$
0	0	0	1	1	$P(d^1 e^0)P(c^1 e^0)$
0	0	0	1	1	$P(d^1 e^0)P(c^1 e^0)$
0	0	1	1	1	$P(d^1 e^1)P(c^1 e^1)$

Table 2.5 shows five likelihood-weighted samples from the network in Figure 2.9. We sample from $P(B)$, $P(S)$, and $P(E | B, S)$, as we would with direct sampling. When we come to D and C , we assign $D = 1$ and $C = 1$. If the sample has $E = 1$, then the weight is $P(d^1 | e^1)P(c^1 | e^1)$; otherwise, the weight is $P(d^1 | e^0)P(c^1 | e^0)$. If we assume

$$P(d^1 | e^1)P(c^1 | e^1) = 0.95 \quad (2.44)$$

$$P(d^1 | e^0)P(c^1 | e^0) = 0.01 \quad (2.45)$$

then we may approximate from the samples in Table 2.5

$$P(b^1 | d^1, c^1) \approx \frac{0.95}{0.95 + 0.95 + 0.01 + 0.01 + 0.95} \quad (2.46)$$

$$= 0.331. \quad (2.47)$$

Although likelihood weighting makes it so that all samples are consistent with the observations, it can still be wasteful. Consider the simple chemical detection Bayesian network shown in Figure 2.15, and assume that we detected a chemical of interest. We want to infer $P(c^1 | d^1)$. Because this network is small, we can easily compute this probability exactly by using Bayes' rule:

$$P(c^1 | d^1) = \frac{P(d^1 | c^1)P(c^1)}{P(d^1 | c^1)P(c^1) + P(d^1 | c^0)P(c^0)} \quad (2.48)$$

$$= \frac{0.999 \times 0.001}{0.999 \times 0.001 + 0.001 \times 0.999} \quad (2.49)$$

$$= 0.5. \quad (2.50)$$

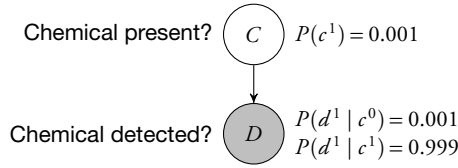


Figure 2.15 Chemical detection Bayesian network.

If we use likelihood weighting, then 99.9% of the samples will have $C = 0$ with a weight of 0.001. Until we get a sample of $C = 1$, which has an associated weight of 0.999, our estimate of $P(c^1 | d^1)$ will be 0.

An alternative approach is to use *Gibbs sampling*, which is a kind of *Markov chain Monte Carlo* technique. Unlike the other sampling methods discussed so far, the samples produced by this method are not independent. The next sample depends probabilistically on the current sample, and so the sequence of samples forms a Markov chain. It can be proven that, in the limit, samples are drawn exactly from the joint distribution over the unobserved variables given the observations.

The initial sample can be generated randomly with the observed variables set to their observed values. Algorithm 2.6 outlines how to generate a new sample $x'_{1:n}$ from an existing sample $x_{1:n}$, given a Bayesian network B and observations $o_{1:n}$. Unlike direct sampling, we can use any ordering for the nodes in the network; the ordering need not be a topological sort. Given this ordering, update the sample one variable at a time given the values of the other variables. To generate the value for x'_i , we sample from $P(X_i | x'_{1:n \setminus i})$, where $x'_{1:n \setminus i}$ represents the values of all the other variables except X_i . To compute the distribution $P(X_i | x'_{1:n \setminus i})$ for a Bayesian network B , we can use Algorithm 2.7. The computation can be done efficiently because we only need to consider the Markov blanket of variable X_i (Section 2.1.5).

Algorithm 2.6 Gibbs sampling from a Bayesian network

```

1: function GIBBSAMPLE( $B, o_{1:n}, x_{1:n}$ )
2:    $X_{1:n} \leftarrow$  an ordering of nodes in  $B$ 
3:    $x'_{1:n} \leftarrow x_{1:n}$ 
4:   for  $i \leftarrow 1$  to  $n$ 
5:     if  $o_i = \text{NIL}$ 
6:        $x'_i \leftarrow$  a random sample from  $P(X_i | x'_{1:n \setminus i})$ 
7:     else
8:        $x'_i \leftarrow o_i$ 
9:   return  $x'_{1:n}$ 

```

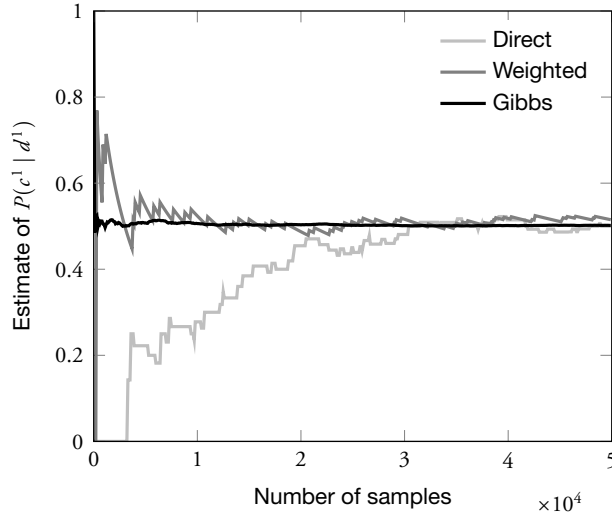


Figure 2.16 Bayesian network sampling methods.

Algorithm 2.7 Distribution at a node given observations at all other nodes

- 1: **function** DISTRIBUTIONATNODE($B, X_i, x_{1:n \setminus i}$)
 - 2: $\mathcal{T} \leftarrow$ all conditional probability tables associated with B involving X_i
 - 3: Remove rows that are inconsistent with $x_{1:n \setminus i}$ from all the tables in $\mathcal{T}$
 - 4: $T \leftarrow$ product of the tables remaining in $\mathcal{T}$
 - 5: $P(X_i | x_{1:n \setminus i}) \leftarrow$ normalize T
 - 6: **return** $P(X_i | x_{1:n \setminus i})$
-

Figure 2.16 compares the convergence of the estimate of $P(c^1 | d^1)$ using direct, likelihood weighted, and Gibbs sampling. Direct sampling takes the longest to converge. The direct sampling curve has long periods during which the estimate does not change because samples are inconsistent with the observations. Likelihood-weighted sampling converges faster in this example. Spikes occur when a sample is generated with $C = 1$ and then gradually decrease. Gibbs sampling, in this example, quickly converges to the true value of 0.5.

As mentioned earlier, Gibbs sampling, like other Markov chain Monte Carlo methods, produces samples from the desired distribution *in the limit*. In practice, we have to run Gibbs for some amount of time, called the *burn-in period*, before converging to a steady state distribution. The samples produced during burn-in are normally discarded. In addition, because of potential correlation between samples, it is common to *thin* the samples by only keeping every k th sample.

Other approximate inference methods do not involve generating samples. For example, a form of belief propagation called *loopy belief propagation* can be used in networks with undirected cycles for approximate inference. Although not guaranteed to be exact, loopy belief propagation tends to work well in practice and is becoming one of the most popular methods for approximate inference in Bayesian networks.

2.3 Parameter Learning

So far in this chapter, we have assumed that the parameters and structure of our probabilistic models were known. This section addresses the problem of learning the parameters of the model from data.

2.3.1 Maximum Likelihood Parameter Learning

Suppose the random variable C represents whether a flight will result in a mid-air collision, and we are interested in estimating the distribution $P(C)$. Because C is either 0 or 1, it is sufficient to estimate the parameter $\theta = P(c^1)$. What we want to do is infer θ from data D . Let us say that we have a historical database spanning a decade, and let us say we know there were n flights and m mid-air collisions. Our intuition, of course, tells us that a good estimate for θ given the data D is m/n . This estimate corresponds to the *maximum likelihood estimate*,

$$\hat{\theta} = \arg \max_{\theta} P(D \mid \theta). \quad (2.51)$$

The probability of m mid-air collisions out of n flights is given by the *binomial distribution*:

$$P(D \mid \theta) = \frac{n!}{m!(n-m)!} \theta^m (1-\theta)^{n-m} \quad (2.52)$$

$$\propto \theta^m (1-\theta)^{n-m}. \quad (2.53)$$

The maximum likelihood estimate $\hat{\theta}$ is the value for θ that maximizes Equation (2.53). Maximizing Equation (2.53) is equivalent to maximizing the logarithm of the likelihood, often referred to as the *log-likelihood* and often denoted $\ell(\theta)$:

$$\ell(\theta) \propto \ln(\theta^m (1-\theta)^{n-m}) \quad (2.54)$$

$$= m \ln \theta + (n-m) \ln(1-\theta). \quad (2.55)$$

We can use the standard technique for finding the maximum of a function by setting the first derivative of ℓ to 0 and then solving for θ . The derivative is given by

$$\frac{\partial \ell(\theta)}{\partial \theta} = \frac{m}{\theta} - \frac{n-m}{1-\theta}. \quad (2.56)$$

We can solve for $\hat{\theta}$ by setting the derivative to 0:

$$\frac{m}{\hat{\theta}} - \frac{n-m}{1-\hat{\theta}} = 0. \quad (2.57)$$

After a few algebraic steps, we see that, indeed, $\hat{\theta} = m/n$.

Computing the maximum likelihood estimate for a variable X that can assume k values is also straightforward. If $m_{1:k}$ are the observed counts for the k different values, then the maximum likelihood estimate for $P(x^i | m_{1:k})$ is given by

$$\hat{\theta}_i = \frac{m_i}{\sum_{j=1}^k m_j}. \quad (2.58)$$

Maximum likelihood estimation can also be applied to continuous distributions. Suppose we have airspeed measurements $v_{1:n}$ of the n aircraft tracks that were used to generate the class conditional distribution in Figure 2.12. Although the density is clearly not exactly Gaussian, let us try to fit a Gaussian model to these data by using maximum likelihood estimation of the parameters. The log-likelihood of the mean μ and variance σ^2 is given by

$$\ell(\mu, \sigma^2) \propto -n \ln \sigma - \frac{\sum_i (v_i - \mu)^2}{2\sigma^2}. \quad (2.59)$$

Again, we can set the derivative to 0 with respect to the parameters and solve for the maximum likelihood estimate:

$$\frac{\partial \ell(\mu, \sigma^2)}{\partial \mu} = \frac{\sum_i (v_i - \hat{\mu})}{\hat{\sigma}^2} = 0 \quad (2.60)$$

$$\frac{\partial \ell(\mu, \sigma^2)}{\partial \sigma} = -\frac{n}{\hat{\sigma}} + \frac{\sum_i (v_i - \hat{\mu})^2}{\hat{\sigma}^3} = 0. \quad (2.61)$$

After some algebraic manipulation, we get

$$\hat{\mu} = \frac{\sum_i v_i}{n} \quad (2.62)$$

$$\hat{\sigma}^2 = \frac{\sum_i (v_i - \hat{\mu})^2}{n}. \quad (2.63)$$

Figure 2.17 shows a Gaussian with the maximum likelihood estimates $\hat{\mu} = 100.2$ kt and $\hat{\sigma} = 31$ kt. The “true” distribution from Figure 2.12 is shown for comparison. In this case, the Gaussian is a fairly reasonable approximation of the true distribution.

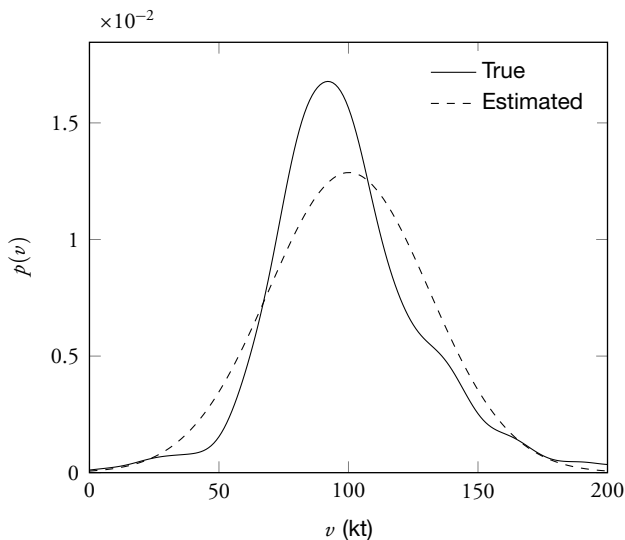


Figure 2.17 Airspeed probability density.

2.3.2 Bayesian Parameter Learning

Although maximum likelihood estimation may be adequate for many applications, it has some serious drawbacks when the amount of data is limited. For example, suppose our aviation safety database was limited to events of the past week, and we found no recorded mid-air collisions. If θ is the probability that a flight results in a mid-air collision, then the maximum likelihood estimate would be $\hat{\theta} = 0$. Believing that there is zero chance of a mid-air collision is not a reasonable conclusion, unless our prior hypothesis was, for example, that either all flights were safe or all flights resulted in collision.

The Bayesian approach to parameter learning involves estimating the posterior over θ and can be viewed as inference in a Bayesian network. For example, Figure 2.18 represents the problem of collision probability estimation, where the observed variable O_i is 1 if the i th flight resulted in collision and 0 otherwise. We assume that the observed variables are conditionally independent of each other. We must specify $p(\theta)$ and $P(O_i | \theta)$. If we want to use a uniform prior, then we can set the density $p(\theta) = 1$. We set $P(O_i = 1 | \theta) = \theta$.

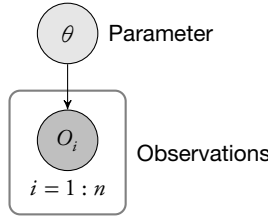


Figure 2.18 Bayesian network representing parameter learning.

We can proceed with the standard method for performing inference in a Bayesian network. Here, we will assume a uniform prior.

$$p(\theta \mid o_{1:n}) \propto p(\theta, o_{1:n}) \quad (2.64)$$

$$= p(\theta) \prod_{i=1}^n P(o_i \mid \theta) \quad (2.65)$$

$$= \prod_{i=1}^n P(o_i \mid \theta) \quad (2.66)$$

$$= \prod_{i=1}^n \theta^{o_i} (1 - \theta)^{1-o_i} \quad (2.67)$$

$$= \theta^m (1 - \theta)^{n-m} \quad (2.68)$$

The posterior is proportional to $\theta^m (1 - \theta)^{n-m}$, where m is the number of mid-air collisions in our data. To find the normalization constant, we integrate

$$\int_0^1 \theta^m (1 - \theta)^{n-m} d\theta = \frac{\Gamma(m+1)\Gamma(n-m+1)}{\Gamma(n+2)}, \quad (2.69)$$

where Γ is the *gamma function*. The gamma function is a real-valued generalization of the factorial. If n is an integer, then $\Gamma(n) = (n-1)!$. Taking normalization into account, we have

$$p(\theta \mid o_{1:n}) = \frac{\Gamma(n+2)}{\Gamma(m+1)\Gamma(n-m+1)} \theta^m (1 - \theta)^{n-m} \quad (2.70)$$

$$= \text{Beta}(\theta \mid m+1, n-m+1). \quad (2.71)$$

The *beta distribution* $\text{Beta}(\alpha, \beta)$ is defined by parameters α and β , and curves for this distribution are shown in Figure 2.19. The distribution $\text{Beta}(1, 1)$ corresponds to the uniform distribution spanning 0 to 1.

Conveniently, if a beta distribution is used as a prior over a parameter of a binomial distribution, then the posterior is also a beta distribution. In particular, if the prior

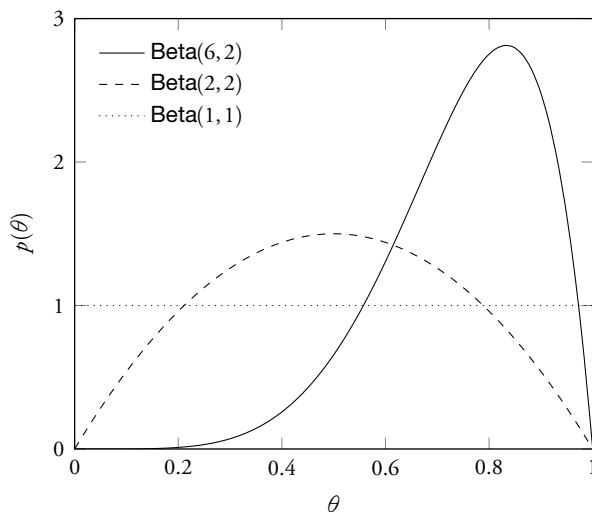


Figure 2.19 Beta distribution.

is given by $\text{Beta}(\alpha, \beta)$ and we make an observation o_i , then we get a posterior of $\text{Beta}(\alpha + 1, \beta)$ if $o_i = 1$ and $\text{Beta}(\alpha, \beta + 1)$ if $o_i = 0$. Hence, if we started with a prior given by $\text{Beta}(\alpha, \beta)$ and our data showed that there were m collisions out of n flights, then the posterior would be given by $\text{Beta}(\alpha + m, \beta + n - m)$. The α and β parameters in the prior are sometimes called *pseudocounts* because they are treated similarly to the observed counts of the two outcomes classes in the posterior, although the pseudocounts need not be integers.

Choosing the prior, in principle, should be done without knowledge of the data used to compute the posterior. Uniform priors often work well in practice, although if expert knowledge is available, then it can be encoded into the prior. For example, suppose we had a slightly bent coin and we wanted to estimate θ , the probability that the coin would land heads. Before we collected any data by flipping the coin, we would start with a belief θ is likely to be around 0.5. Instead of starting with a uniform prior $\text{Beta}(1, 1)$, we might use $\text{Beta}(2, 2)$ (shown in Figure 2.19), which gives more weight to values near 0.5. If we were more confident in an estimate near 0.5, then we could reduce the variance of the prior by increasing the pseudocounts. The prior $\text{Beta}(10, 10)$ is much more peaked than $\text{Beta}(2, 2)$. In general, however, the importance of the prior diminishes with the amount of data used to compute the posterior. If we observe n flips and m were heads, then the difference between $\text{Beta}(1 + m, 1 + n - m)$ and $\text{Beta}(10 + m, 10 + n - m)$ is negligible if we do thousands of coin flips.

The *Dirichlet distribution* is a generalization of the beta distribution and can be used to estimate the parameters of a discrete distribution. Suppose X is a discrete random variable that takes integer values from 1 to n . We define the parameters of the distribution to be $\theta_{1:n}$, where $P(x^i) = \theta_i$. Of course, the parameters must sum to 1, and so only the first $n - 1$ parameters are independent. The Dirichlet distribution can be used to represent both the prior and the posterior distribution and is parameterized by $\alpha_{1:n}$. The density is given by

$$\text{Dir}(\theta_{1:n} \mid \alpha_{1:n}) = \frac{\Gamma(\alpha_0)}{\prod_{i=1}^n \Gamma(\alpha_i)} \prod_{i=1}^n \theta_i^{\alpha_i - 1}, \quad (2.72)$$

where α_0 is used to denote the summation of the parameters $\alpha_{1:n}$. If $n = 2$, then it is easy to see that Equation (2.72) is equivalent to the beta distribution.

It is common to use a uniform prior where all the Dirichlet parameters $\alpha_{1:n}$ are set to 1. A *symmetric Dirichlet distribution* is one where all the parameters are identical. As with the beta distribution, the parameters in the Dirichlet are often referred to as pseudocounts.

If the prior over $\theta_{1:n}$ is given by $\text{Dir}(\alpha_{1:n})$ and there are m_i observations of $X = i$, then the posterior is given by

$$p(\theta_{1:n} \mid \alpha_{1:n}, m_{1:n}) = \text{Dir}(\theta_{1:n} \mid \alpha_1 + m_1, \dots, \alpha_n + m_n). \quad (2.73)$$

As we have seen, Bayesian parameter estimation is straightforward for binary and discrete random variables because it involves simply counting the various outcomes in the data. Bayes' rule can be used to infer the distribution over the parameters for any parametric distribution. Depending on the choice of prior and the form of the parametric distribution, calculating the posterior over the space of parameters also might be done analytically.

2.3.3 Nonparametric Learning

The previous two sections assumed that the probabilistic model was of a fixed form and that a fixed set of parameters were to be learned from the data. An alternative approach is based on *nonparametric* methods in which the number of parameters scale with the amount of data. One of the most common nonparametric methods is called *kernel density estimation*.

Given observations $o_{1:n}$, kernel density estimation represents the density as follows:

$$p(x) = \frac{1}{n} \sum_{i=1}^n K(x - o_i), \quad (2.74)$$

where K is a *kernel function*, which integrates to 1. The kernel function is used to assign greater density to values near the observed data points. A kernel function is generally symmetric, meaning that $K(x) = K(-x)$. A commonly used kernel is the zero-mean Gaussian distribution. When a Gaussian is used as a kernel, the standard deviation is often referred to as the *bandwidth*. Gaussian kernel densities are smooth, unlike the piecewise uniform distributions discussed earlier and shown in Figure 2.1. Larger bandwidths generally lead to smoother densities. Bayesian methods can be applied to the selection of the appropriate bandwidth based on the data.

2.4 Structure Learning

The previous sections assumed that the Bayesian network structure was known a priori. This section discusses methods for learning the structure from data. A maximum likelihood approach to learning the structure of a Bayesian network involves finding the graphical structure G that maximizes $P(G \mid D)$, where D represents the available data. We first explain how to compute a Bayesian network score based on $P(G \mid D)$, and then we explain how to go about searching the space of networks for the highest scoring network. Like inference in Bayesian networks, it can be shown that for general graphs and input data, learning the structure of a Bayesian network is NP-hard.

2.4.1 Bayesian Structure Scoring

Before discussing how to compute $P(G \mid D)$, we need to introduce some notation. We will assume that the n variables in the Bayesian network $X_{1:n}$ are all discrete, although this need not be the case in general. We use r_i to represent the number of instantiations of X_i and q_i to represent the number of instantiations of the parents of X_i . If X_i has no parents, then $q_i = 1$. The j th instantiation of the parents of X_i is denoted π_{ij} .

There are $\sum_{i=1}^n r_i q_i$ parameters in a Bayesian network. Each parameter is written θ_{ijk} and determines

$$P(X_i = k \mid \pi_{ij}) = \theta_{ijk}. \quad (2.75)$$

Although there are $\sum_{i=1}^n r_i q_i$ parameters, only $\sum_{i=1}^n (r_i - 1)q_i$ are independent. We use θ to represent the set of all the parameters.

We use m_{ijk} to represent the number of times $X_i = k$ given π_{ij} in the dataset D . The likelihood is given by

$$P(D \mid \theta, G) = \prod_{i=1}^n \prod_{j=1}^{q_i} \prod_{k=1}^{r_i} \theta_{ijk}^{m_{ijk}}. \quad (2.76)$$

The prior over the Bayesian network parameters θ can be factorized. If we have that $\theta_{ij} = (\theta_{ij1}, \dots, \theta_{ijr_i})$, then

$$p(\theta \mid G) = \prod_{i=1}^n \prod_{j=1}^{q_i} p(\theta_{ij}). \quad (2.77)$$

The prior $p(\theta_{ij})$, under some weak assumptions, can be shown to follow the Dirichlet distribution in Equation (2.72). The distribution over X_i given the j th parental instantiation is given by $\text{Dir}(\alpha_{ij1}, \dots, \alpha_{ijr_i})$.

We compute $P(G \mid D)$ using Bayes' rule and the law of total probability:

$$P(G \mid D) \propto P(G)P(D \mid G) \quad (2.78)$$

$$= P(G) \int P(D \mid \theta, G) p(\theta \mid G) d\theta. \quad (2.79)$$

It turns out that, after integrating the product of Equations (2.76) and (2.77) over θ , we get

$$P(G \mid D) = P(G) \prod_{i=1}^n \prod_{j=1}^{q_i} \frac{\Gamma(\alpha_{ij0})}{\Gamma(\alpha_{ij0} + m_{ij0})} \prod_{k=1}^{r_i} \frac{\Gamma(\alpha_{ijk} + m_{ijk})}{\Gamma(\alpha_{ijk})}. \quad (2.80)$$

In the equation above,

$$\alpha_{ij0} = \sum_{k=1}^{r_i} \alpha_{ijk} \quad (2.81)$$

$$m_{ij0} = \sum_{k=1}^{r_i} m_{ijk}. \quad (2.82)$$

Finding the G that maximizes Equation (2.79) is the same as finding the G that maximizes what is called the *Bayesian score*:

$$\ln P(G \mid D) = \ln P(G) + \sum_{i=1}^n \sum_{j=1}^{q_i} \ln \left(\frac{\Gamma(\alpha_{ij0})}{\Gamma(\alpha_{ij0} + m_{ij0})} \right) + \sum_{k=1}^{r_i} \ln \left(\frac{\Gamma(\alpha_{ijk} + m_{ijk})}{\Gamma(\alpha_{ijk})} \right). \quad (2.83)$$

The Bayesian score is more convenient to compute numerically because it is easier to add the logarithm of small numbers together than to multiply them together. Many software libraries can compute the logarithm of the gamma function directly with reasonable precision.

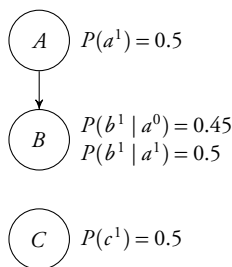


Figure 2.20 Example Bayesian network.

A variety of different graph priors have been explored in the literature, although a uniform prior is often used in practice, in which case $\ln P(G)$ can be dropped from the computation of the Bayesian score in Equation (2.83). One of the useful properties of the Bayesian score, even with a uniform graph prior, is that it optimally balances the complexity of the model with the available data.

To illustrate how Bayesian scoring balances model complexity, consider the simple Bayesian network in Figure 2.20. The value of A weakly influences the value of B , and C is independent of the other variables. We sample from this “true” model to generate data D and then try to learn the model structure. There are 25 possible network structures involving three variables, but we will focus on the scores for the following models:

- The true model with $1 + 2 + 1 = 4$ independent parameters,
- The completely connected model $A \rightarrow B, A \rightarrow C, B \rightarrow C$ with $1 + 2 + 4 = 7$ independent parameters, and
- The completely unconnected model with $1 + 1 + 1 = 3$ independent parameters.

Figure 2.21 shows how the Bayesian scores of the fully connected and unconnected models compare to the true model as the amount of data increases. In the plot, we subtract the score of the true model, so values above 0 indicate that the model provides a better representation than the true model given the available data. The plot shows that the unconnected model does better than the true model when there are fewer than 5000 samples. The fully connected model never does better than the true model, but it starts to do better than the unconnected model at about 10,000 samples because there are sufficient data to adequately estimate its seven independent parameters.

2.4.2 Directed Graph Search

The space of possible Bayesian network structures grows superexponentially. With 10 nodes, there are 4.2×10^{18} possible directed acyclic graphs. With 20 nodes, there are 2.4×10^{72} . Except for Bayesian networks with few nodes, we cannot enumerate the

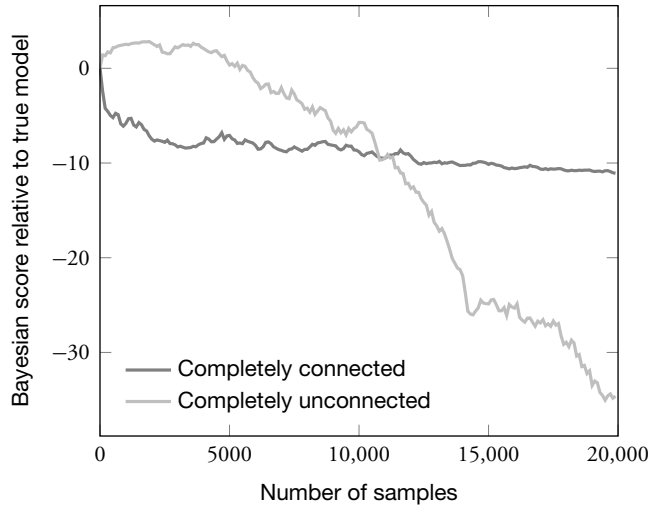


Figure 2.21 Bayesian network score relative to true model.

space of possible structures to find the highest scoring network. Therefore, we have to rely on a search strategy. Fortunately, search is a general problem, and a wide variety of different generic search algorithms have been studied over the years.

One of the most common search strategies is called K2 (named such because it is an evolution of a system called Kutató). The search (Algorithm 2.8) runs in polynomial time but does not guarantee finding a globally optimal network structure. It can use any scoring function f , but it is often used with the Bayesian score because of its ability to balance the complexity of the model with the amount of data available. K2 begins with a graph with no directed edges and then iterates over an assumed variable ordering, greedily adding parents to the nodes in a way that maximally increases the score. It is common for K2 to impose an upper bound on the number of parents for any one node to reduce the required computation. The original K2 algorithm assumed that the Dirichlet prior parameters $\alpha_{ijk} = 1$ for all i , j , and k , but any prior can be used in principle.

A general search strategy is *local search*, which is sometimes called *hill climbing* or *gradient ascent*, and is outlined in Algorithm 2.9. We start with an initial graph G_0 and then move to the highest scoring neighbor. The neighborhood of a graph consists of the graphs that are only one basic graph operation away, where the basic graph operations include:

- If an edge between A and B does not exist, introduce $A \rightarrow B$.
- If $A \rightarrow B$, remove the edge from A to B .
- If $A \rightarrow B$, then reverse the direction of the edge to get $A \leftarrow B$.

Algorithm 2.8 K2 search of space of directed acyclic graphs

```

1: function K2SEARCH( $f$ )
2:    $X_{1:n} \leftarrow$  ordering of nodes
3:    $G' \leftarrow$  graph containing nodes  $X_{1:n}$  and no edges
4:   for  $i \leftarrow 1$  to  $n$ 
5:     repeat
6:        $G \leftarrow G'$ 
7:       Add a parent to node  $X_i$  in graph  $G'$  that maximizes  $f(G')$ 
8:     until  $f(G') \leq f(G)$ 
9:   return  $G$ 

```

Of course, not all operations are possible from a particular graph, and operations that introduce cycles into the graph are invalid. The search continues until the current graph scores no higher than all of its neighbors.

Algorithm 2.9 Local search of space of directed acyclic graphs

```

1: function LOCALDIRECTEDGRAPHSEARCH( $f, G_0$ )
2:    $G' \leftarrow G_0$ 
3:   repeat
4:      $G \leftarrow G'$ 
5:      $\mathcal{G} \leftarrow$  neighbors of  $G$ 
6:      $G' \leftarrow \arg \max_{G' \in \mathcal{G}} f(G')$ 
7:   until  $f(G') \leq f(G)$ 
8:   return  $G$ 

```

Local search can get stuck in *local optima*, preventing it from finding the globally optimal network structure. Various strategies have been proposed for addressing local optima, including the following:

- *Randomized restart*. Once a local optima has been found, simply restart the search at a random point in the search space.
- *Simulated annealing*. Instead of always moving to the neighbor with greatest fitness, the search can visit neighbors with lower fitness according to some randomized exploration strategy. As the search progresses, the randomness in the exploration decreases according to some schedule. This approach is called simulated annealing because of its inspiration from annealing in metallurgy.
- *Tabu search*. This approach involves maintaining a *tabu list* containing recently visited points in the search space. The search algorithm avoids neighbors in the tabu list.

- *Genetic algorithms.* The procedure begins with an initial random population of points in the search space represented as binary strings. When searching the space of directed graphs, a bit in the string indicates the presence or absence of an arrow between two nodes. The individuals in the population reproduce at a rate proportional to their score. Individuals selected for reproduction have their strings recombined randomly through genetic crossover. Genetic crossover involves selecting a crossover point on two randomly selected individuals and then swapping the strings after that point. Mutations are also introduced randomly into the population by randomly flipping bits in the strings. The process of evolution continues until a satisfactory point in the search space is found.
- *Memetic algorithms.* This approach is sometimes called *genetic local search* and is simply a combination of genetic algorithms with local search. After genetic recombination, local search is applied to the individuals.

Some search strategies may work better than others on certain datasets, but in general finding the global optima remains NP-hard. Many applications, however, do not require the globally optimal network structure. A locally optimal structure is often acceptable.

2.4.3 Markov Equivalence Classes

As discussed earlier, the structure of a Bayesian network encodes a set of conditional independence assumptions. An important observation to make when trying to learn the structure of a Bayesian network is that two different graphs can encode the same independence assumptions. As a simple example, consider the two-variable network $A \rightarrow B$. This network contains the same independence assumptions as $A \leftarrow B$. Solely on the basis of the data, one cannot justify the direction of the edge between A and B .

Two graphs are *Markov equivalent* if they encode the same set of conditional independence assumptions. It can be proven that two graphs are Markov equivalent if and only if they have (1) the same edges without regard to direction and (2) the same *immoral* v-structures. An immoral v-structure is a v-structure $X \rightarrow Y \leftarrow Z$ with X and Z not directly connected. A *Markov equivalence class* is a set containing all the directed acyclic graphs that are Markov equivalent to each other.

In general, two structures belonging to the same Markov equivalence class may be given different scores. However, if the Bayesian score is used with Dirichlet priors such that $\kappa = \sum_j \sum_k \alpha_{ijk}$ is constant for all i , then two Markov equivalent structures are assigned the same score. Such priors are called BDe, and a special case is the BDeu prior, which assigns $\alpha_{ijk} = \kappa / (q_i r_i)$. Although the commonly used uniform prior $\alpha_{ijk} = 1$ does not always result in identical scores being assigned to structures in the same equivalence class, they are often fairly close. A scoring function that assigns the same score to all structures in the same class is called *score equivalent*.

2.4.4 Partially Directed Graph Search

A Markov equivalence class can be represented as a *partially directed graph*, sometimes called an *essential graph* or a *directed acyclic graph pattern*. A partially directed graph can contain both directed edges and undirected edges. An example of a partially directed graph that encodes a Markov equivalence class is shown in Figure 2.22a. A directed acyclic graph G is a member of the Markov equivalence class encoded by a partially directed graph G' if and only if G (1) has the same edges as G' without regard to direction and (2) has the same v-structures as G' . Figures 2.22b and 2.22c are examples of members of the equivalence class in Figure 2.22a. Figure 2.22d is not a member.

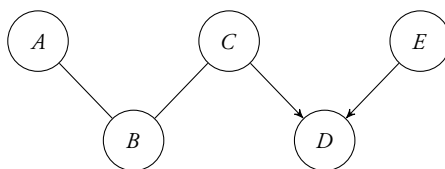
Instead of searching the space of directed acyclic graphs, we can search the space of Markov equivalence classes represented by partially directed graphs. The space of Markov equivalence classes is smaller than the space of directed acyclic graphs, and so search can be done more efficiently. Any of the general search strategies presented in Section 2.4.2 can be used. If a form of local search is used, then we need to define the local graph operations that define the neighborhood of the graph, for example,

- If an edge between A and B does not exist, add either $A - B$ or $A \rightarrow B$.
- If $A - B$ or $A \rightarrow B$, then remove the edge between A and B .
- If $A \rightarrow B$, then reverse the direction of the edge to get $A \leftarrow B$.
- If $A - B - C$, then add $A \rightarrow B \leftarrow C$.

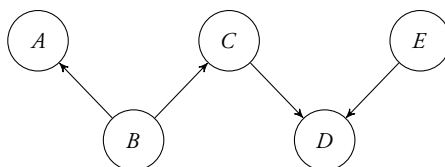
The Bayesian score is defined for directed acyclic graphs. To score a partially directed graph, we need to generate a member of its Markov equivalence class and compute its score. Generating a member from a partially directed graph involves converting the undirected edges to directed edges in a way that does not introduce new v-structures.

2.5 Summary

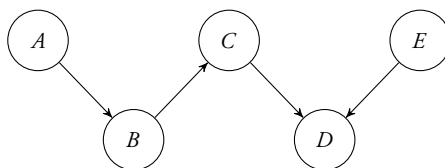
- Uncertainty can arise from incomplete information or the inability to predict future events because of practical or theoretical limitations.
- Properly accounting for uncertainty is important when building robust decision-making systems.
- Bayesian networks compactly represent distributions over variables.
- The structure of a network encodes conditional independence assumptions.
- Bayesian networks are a flexible representation for encoding a wide variety of models.
- Probabilistic inference can be made efficient if network structure is leveraged.
- Bayesian and maximum likelihood methods can be used to infer model parameters and structure.



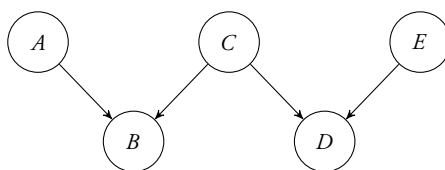
(a) Markov equivalence class.



(b) Member.



(c) Member.



(d) Nonmember.

Figure 2.22 Markov equivalence class example.

2.6 Further Reading

One of the most in-depth treatments of probabilistic models is *Probabilistic Graphical Models: Principles and Techniques* by Koller and Friedman [2]. Barber also provides an overview of probabilistic models and their applications in *Bayesian Reasoning and Machine Learning* [3]. Russell and Norvig use Bayesian networks throughout their popular artificial intelligence textbook, *Artificial Intelligence: A Modern Approach* [4].

The foundations of probability theory are discussed in *Probability Theory: The Logic of Science* by Jaynes [1]. Fishburn surveys the axiomatization of subjective probability [5], and Dupré and Tipler give a more recent axiomatization [6]. The textbook by Bertsekas and Tsitsiklis provides a comprehensive introduction to probability [7].

Several textbooks discuss inference in Bayesian networks and other kinds of probabilistic graphical models, such as Markov random fields and factor graphs [2]–[4], [8]–[10]. These books discuss inference methods, including belief propagation and the junction tree algorithm mentioned in Section 2.2. The message passing algorithm for exact inference in polytrees is given by Kim and Pearl [11]. The proof that inference is NP-hard in Bayesian networks is provided by Cooper [12].

Overviews of Bayesian network structure and parameter learning can be found in the textbooks *Probabilistic Graphical Models: Principles and Techniques* [2] and *Learning Bayesian Networks* [13]. As mentioned in the text, learning the optimal network structure is NP-hard [14], [15]. Cooper and Herskovits developed the K2 search algorithm introduced in Algorithm 2.8 [16]. Searching the space of partially directed graphs is discussed by Chickering [17]. Heckerman, Geiger, and Chickering showed that a BDe prior leads to identical Bayesian scores for any two Markov equivalent structures [18]. The BDeu prior was originally proposed by Buntine [19].

References

1. E.T. Jaynes, *Probability Theory: The Logic of Science*. New York: Cambridge University Press, 2003.
2. D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*. Cambridge, MA: MIT Press, 2009.
3. D. Barber, *Bayesian Reasoning and Machine Learning*. New York: Cambridge University Press, 2012.
4. S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Upper Saddle River, NJ: Pearson, 2010.
5. P.C. Fishburn, “The Axioms of Subjective Probability,” *Statistical Science*, vol. 1, no. 3, pp. 335–345, 1986. doi: 10.1214/ss/1177013611.

6. M.J. Dupré and F.J. Tipler, “New Axioms for Rigorous Bayesian Probability,” *Bayesian Analysis*, vol. 4, no. 3, pp. 599–606, 2009. doi: 10.1214/09-BA422.
7. D.P. Bertsekas and J.N. Tsitsiklis, *Introduction to Probability*. Belmont, MA: Athena Scientific, 2002.
8. F.V. Jensen and T.D. Nielsen, *Bayesian Networks and Decision Graphs*, 2nd ed. New York: Springer, 2007.
9. D.J.C. MacKay, *Information Theory, Inference and Learning Algorithms*. New York: Cambridge University Press, 2003.
10. C.M. Bishop, *Pattern Recognition and Machine Learning*. New York: Springer, 2006.
11. J.H. Kim and J. Pearl, “A Computational Model for Combined Causal and Diagnostic Reasoning in Inference Systems,” in *International Joint Conference on Artificial Intelligence (IJCAI)*, 1983.
12. G.F. Cooper, “The Computational Complexity of Probabilistic Inference Using Bayesian Belief Networks,” *Artificial Intelligence*, vol. 42, no. 2–3, pp. 393–405, 1990. doi: 10.1016/0004-3702(90)90060-D.
13. R.E. Neapolitan, *Learning Bayesian Networks*. Upper Saddle River, NJ: Prentice Hall, 2003.
14. D.M. Chickering, “Learning Bayesian Networks Is NP-Complete,” in *Learning from Data: Artificial Intelligence and Statistics V*, D. Fisher and H.-J. Lenz, eds., New York: Springer, 1996.
15. D.M. Chickering, D. Heckerman, and C. Meek, “Large-Sample Learning of Bayesian Networks Is NP-Hard,” *Journal of Machine Learning Research*, vol. 5, pp. 1287–1330, 2004.
16. G.F. Cooper and E. Herskovits, “A Bayesian Method for the Induction of Probabilistic Networks from Data,” *Machine Learning*, vol. 4, no. 9, pp. 309–347, 1992. doi: 10.1007/BF00994110.
17. D.M. Chickering, “Learning Equivalence Classes of Bayesian-Network Structures,” *Journal of Machine Learning Research*, vol. 2, pp. 445–498, 2002.
18. D. Heckerman, D. Geiger, and D.M. Chickering, “Learning Bayesian Networks: The Combination of Knowledge and Statistical Data,” *Machine Learning*, vol. 20, no. 3, pp. 197–243, 1995. doi: 10.1007/BF00994016.
19. W.L. Buntine, “Theory Refinement on Bayesian Networks,” in *Conference on Uncertainty in Artificial Intelligence (UAI)*, 1991.

3

Decision Problems

Mykel J. Kochenderfer

The previous chapter focused on uncertainty, including how to build probabilistic models of uncertainty and use them to make inferences. This chapter focuses on how to make rational decisions based on a probabilistic model and utility function. We will focus on single-step decisions, reserving discussion of sequential decision problems for the next chapter. This chapter begins by introducing the foundations of utility theory and showing how it forms the basis for rational decision making under uncertainty. We will then show how notions of utility theory can be incorporated into the probabilistic graphical models introduced in the previous chapter to form what are called decision networks. Because many important decision problems involve interacting with other agents, we will briefly discuss game-theoretic models.

3.1 Utility Theory

The previous chapter began by discussing the need to compare the degree of belief with respect to two different statements. This chapter requires the ability to compare the degree of desirability of two different outcomes. We state our preferences using the following operators:

- $A \succ B$ if we prefer A over B .
- $A \sim B$ if we are indifferent between A and B .
- $A \succeq B$ if we prefer A over B or are indifferent.

Just as beliefs can be subjective, so can preferences.

In addition to comparing events, our preference operators can be used to compare preferences over uncertain outcomes. A *lottery* is a set of probabilities associated with a set of outcomes. For example, if $S_{1:n}$ is a set of outcomes and $p_{1:n}$ are their associated probabilities, then the lottery involving these outcomes and probabilities is written

$$[S_1 : p_1; \dots; S_n : p_n]. \quad (3.1)$$

This section discusses how the existence of a real-valued measure of utility emerges from a set of assumptions about preferences. From this utility function, it is possible to define what it means to make rational decisions under uncertainty.

3.1.1 Constraints on Rational Preferences

Just as we imposed a set of constraints on beliefs, we will impose some constraints on preferences. These constraints are sometimes called the *von Neumann-Morgenstern axioms*, named after John von Neumann and Oskar Morgenstern, who formulated a variation of these axioms in the 1940s.

- *Completeness*. Exactly one of the following hold: $A \succ B$, $B \succ A$, or $A \sim B$.
- *Transitivity*. If $A \succeq B$ and $B \succeq C$, then $A \succeq C$.
- *Continuity*. If $A \succeq C \succeq B$, then there exists a probability p such that $[A : p; B : 1 - p] \sim C$.
- *Independence*. If $A \succ B$, then for any C and probability p , $[A : p; C : 1 - p] \succeq [B : p; C : 1 - p]$.

These are constraints on *rational preferences*. They say nothing about the preferences of actual human beings; in fact, there is strong evidence that humans are not very rational (Section 3.1.7). Our objective in this book is to understand rational decision making from a computational perspective so that we can build useful systems. The possible extension of this theory to understanding human decision making is of only secondary interest.

3.1.2 Utility Functions

Just as constraints on the comparison of plausibility of different statements lead to the existence of a real-valued probability measure, constraints on rational preferences lead to the existence of a real-valued *utility* measure. It follows from our constraints on rational preferences that there exists a real-valued utility function U such that

- $U(A) > U(B)$ if and only if $A \succ B$, and
- $U(A) = U(B)$ if and only if $A \sim B$.

The utility function is unique up to an *affine transformation*. In other words, for any constants $m > 0$ and b , $U'(S) = mU(S) + b$ if and only if the preferences induced by U' are the same as U . Utilities are like temperatures: you can compare temperatures using Kelvin, Celsius, or Fahrenheit, all of which are affine transformations of each other.

It follows from the constraints on rational preferences that the utility of a lottery is given by

$$U([S_1 : p_1; \dots; S_n : p_n]) = \sum_{i=1}^n p_i U(S_i). \quad (3.2)$$

Suppose we are building a collision avoidance system. The outcome of an encounter of an aircraft is defined by whether the system alerts (A) and whether a collision results (C). Because A and C are binary, there are four possible outcomes. So long as our preferences are rational, we can write our utility function over the space of possible lotteries in terms of four parameters: $U(a^0, c^0)$, $U(a^1, c^0)$, $U(a^0, c^1)$, and $U(a^1, c^1)$. For example,

$$U([a^0, c^0 : 0.5; a^1, c^0 : 0.3; a^0, c^1 : 0.1; a^1, c^1 : 0.1]) \quad (3.3)$$

is equal to

$$0.5U(a^0, c^0) + 0.3U(a^1, c^0) + 0.1U(a^0, c^1) + 0.1U(a^1, c^1). \quad (3.4)$$

If the utility function is bounded, then we can define a *normalized utility function* where the best possible outcome is assigned utility 1 and the worst possible outcome is assigned utility 0. The utility of each of the other outcomes is scaled and translated as necessary.

3.1.3 Maximum Expected Utility Principle

We are interested in the problem of making rational decisions with imperfect knowledge of the state of the world. Suppose we have a probabilistic model $P(s' | a, o)$, which represents the probability that the state of the world becomes s' given that we observe o and take action a . We have a utility function $U(s')$ that encodes our preferences over the space of outcomes. Our *expected utility* of taking action a given observation o is given by

$$EU(a | o) = \sum_{s'} P(s' | a, o) U(s'). \quad (3.5)$$

The *principle of maximum expected utility* says that a rational agent should choose the action that maximizes expected utility

$$a^* = \arg \max_a EU(a | o). \quad (3.6)$$

Because we are interested in building rational agents, Equation (3.6) plays a central role in this book.

3.1.4 Utility Elicitation

In building a decision making or decision support system, it is often helpful to infer the utility function from a human or a group of humans. This approach is called *utility elicitation* or *preference elicitation*. One way to go about doing this is to fix the utility of the worst outcome $S_{\perp}$ to 0 and the best outcome $S_{\top}$ to 1. So long as the utilities of the outcomes are bounded, we can translate and scale the utilities without altering our preferences. If we want to determine the utility of outcome S , then we determine the probability p such that $S \sim [S_{\top} : p; S_{\perp} : 1 - p]$. It follows that $U(S) = p$.

In our collision avoidance example, the best possible event is to not alert and not have a collision, and so we set $U(a^0, c^0) = 1$. The worst possible event is to alert and have a collision, and so we set $U(a^1, c^1) = 0$. We define the lottery $L(p)$ to be $[a^0, c^0 : p; a^1, c^1 : 1 - p]$. To determine $U(a^1, c^0)$, we must find the p such that $(a^1, c^0) \sim L(p)$. Similarly, to determine $U(a^0, c^1)$, we find the p such that $(a^0, c^1) \sim L(p)$.

3.1.5 Utility of Money

It may be tempting to use monetary values to infer utility functions. For example, when one is building a decision support system for managing wildfires, it would be natural to define a utility function in terms of the monetary cost incurred by property damage and the monetary cost for deploying fire suppression resources. However, it is well known in economics that the utility of money, in general, is not linear. If there were a linear relationship between utility and money, then decisions should be made in terms of maximizing expected monetary value. Someone who tries to maximize expected monetary value would have no use for insurance because the expected monetary values of insurance policies are generally negative.

We can determine the utility of money using the elicitation process in Section 3.1.4. Of course, different people have different utility functions, but the function generally follows the curve shown in Figure 3.1. For small amounts of money, the curve is roughly linear—\$100 is about twice as good as \$50. For larger amounts of money, the relationship is often treated as logarithmic. The flattening of the curve makes sense; after all, \$1000 is worth less to a billionaire than it is to the average person.

When discussing monetary utility functions, the three terms below are often used. To illustrate, assume A represents being given \$50 and B represents a 50% chance of winning \$100.

- *Risk neutral*. The utility function is linear. There is no preference between \$50 and the 50% chance of winning \$100 ($A \sim B$).
- *Risk seeking*. The utility function is concave up. There is a preference for the 50% chance of winning \$100 ($A < B$).

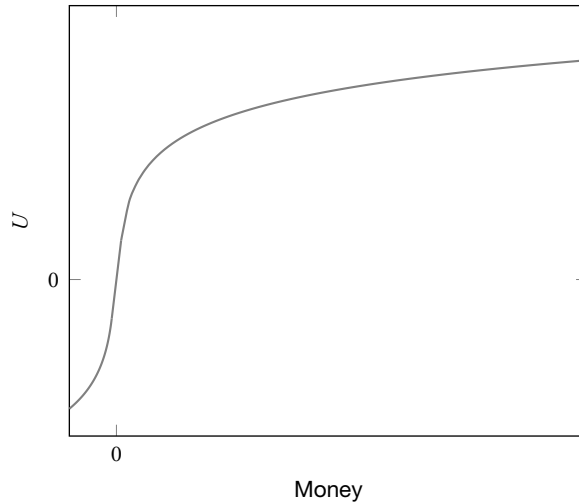


Figure 3.1 Utility of money.

- *Risk averse.* The utility function is concave down. There is a preference for the \$50 ($A \succ B$).

In the process of building decision-making systems, it is useful to use monetary value to inform the construction of the utility function. However, it is important to keep in mind the potentially nonlinear relationship between money and utility.

3.1.6 Multiple Variable Utility Functions

The collision avoidance utility function (Section 3.1.4) depended on two binary variables: whether there was an alert and whether there was a collision. We had to define the utility over all possible combinations of assignments to these two variables. If we had n binary variables, then we would have to specify 2^n parameters in the utility function. If we are able to normalize the utility function, then at least one of the parameters will be 0 and at least one of the parameters will be 1. We can attempt to represent utility functions compactly by leveraging different forms of independence between variables, similar to how Bayesian networks compactly represent joint probability distributions.

Under certain assumptions about the preference structure, we can represent a multiple variable utility function by using a sum of single-variable utility functions. If we have n variables $X_{1:n}$, then we can write

$$U(x_{1:n}) = \sum_{i=1}^n U(x_i). \quad (3.7)$$

To represent the utility function, assuming all variables are binary, we need only $2n$ parameters:

$$U(x_1^0), U(x_1^1), \dots, U(x_n^0), U(x_n^1). \quad (3.8)$$

To illustrate the value of an additive decomposition of the utility function, we will add two additional variables to our collision avoidance example:

- *Strengthening* (S), which indicates whether the collision avoidance system instructed the pilots to strengthen (or increase) their climb or descent.
- *Reversal* (R), which indicates whether the collision avoidance system instructed the pilots to change direction (i.e., up to down or down to up).

If we did not try to leverage the preference structure over A , C , S , and R , then we would need $2^4 = 16$ parameters. With an additive decomposition, we need only eight parameters.

Although it is common to normalize utilities between 0 and 1, it may be more natural to use a different scheme for constructing the utility function. In the collision avoidance problem, it is easy to formulate the utility function in terms of costs associated with an alert, collision, strengthening, and reversal. If there is no alert, collision, strengthening, or reversal, then there is no cost—in other words, $U(a^0)$, $U(c^0)$, $U(s^0)$, and $U(r^0)$ are all equal to 0. The highest cost outcome is collision, and so we set $U(c^1) = -1$ and normalize the other utilities with respect to this outcome. An alert provides the lowest contribution to the overall cost, and a reversal costs more than a strengthening because it is more disruptive to the pilots. Given our assumptions and keeping $U(c^1)$ fixed at -1 , we only have three free parameters to define our utility function.

The utility function for many problems cannot be additively decomposed into utility functions over individual variables as in Equation (3.7). For example, suppose our collision avoidance utility function was defined over three binary variables: whether the intruder comes close horizontally (H), whether the intruder comes close vertically (V), and whether the system alerts (A). A collision threat only really exists if both h^1 and v^1 . Therefore, we cannot additively decompose the utility function over the variables independently. However, we can write $U(h, v, a) = U(h, v) + U(a)$.

We can make the additive decomposition explicit in a diagram, as shown in Figure 3.2. The *utility nodes* are indicated by diamonds. The parents of a utility node are *uncertainty nodes* representing the variables on which the utility node depends. If the parents of a utility node are discrete, then the utility function associated with that node can be represented as a table. If the parents of a utility node are continuous, then any real-valued function can be used to represent the utility. If the diagram has multiple utility nodes, their values are summed together to provide an overall utility value.

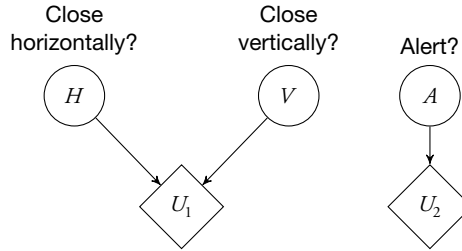


Figure 3.2 Additive decomposition of utility function.

3.1.7 Irrationality

Decision theory is a normative theory that is prescriptive, not a descriptive theory that is predictive of human behavior. Human judgment and preference often do not follow the rules of rationality outlined in Section 3.1.1. Even human experts may have an inconsistent set of preferences, which can be problematic when designing a decision support system that attempts to maximize expected utility.

Tversky and Kahneman studied the preferences of university students who answered questionnaires in a classroom setting. They presented students with questions dealing with the response to an epidemic. The students were to reveal their preference between the following two outcomes:

- *A*: 100% chance of losing 75 lives
- *B*: 80% chance of losing 100 lives

Most preferred *B* over *A*. From Equation (3.2), we know

$$U(\text{lose } 75) < 0.8U(\text{lose } 100). \quad (3.9)$$

They were then asked to choose between the following two outcomes:

- *C*: 10% chance of losing 75 lives
- *D*: 8% chance of losing 100 lives

Most preferred *C* over *D*. Hence, $0.1U(\text{lose } 75) > 0.08U(\text{lose } 100)$. We multiply both sides by 10 and get

$$U(\text{lose } 75) > 0.8U(\text{lose } 100). \quad (3.10)$$

Of course, Equations (3.9) and (3.10) result in a contradiction. We have made no assumption about the actual value of $U(\text{lose } 75)$ and $U(\text{lose } 100)$ —we did not even assume that losing 100 lives was worse than losing 75 lives. Because Equation (3.2) follows directly from the von Neumann-Morgenstern axioms in Section 3.1.1, there must be a violation of at least one of the axioms—even though many people who select *B* and *C* seem to find the axioms agreeable.

The experiments of Tversky and Kahneman show that certainty often exaggerates losses that are certain relative to losses that are merely probable. They found that this *certainty effect* works with gains as well. A smaller gain that is certain is often preferred over a much greater gain that is only probable, in a way that the axioms of rationality are necessarily violated.

Tversky and Kahneman also demonstrated the *framing effect* using a hypothetical scenario in which an epidemic is expected to kill 600 people. They presented students with the following two outcomes:

- *E*: 200 people will be saved
- *F*: 1/3 chance that 600 people will be saved and 2/3 chance that no people will be saved

The majority of students chose *E* over *F*. They then asked them to choose between:

- *G*: 400 people will die
- *H*: 1/3 chance that nobody will die and 2/3 chance that 600 people will die

The majority of students chose *H* over *G*, even though *E* is equivalent to *G* and *F* is equivalent to *H*. This inconsistency is due to how the question is framed.

Many other cognitive biases can lead to deviations from what is prescribed by utility theory. Special care must be given when trying to elicit utility functions from human experts to build decision support systems. Although the recommendations of the decision support system may be rational, they may not exactly reflect human preferences in certain situations.

3.2 Decision Networks

We can extend the notion of Bayesian networks introduced in the last chapter to *decision networks* that incorporate actions and utilities. Decision networks are composed of three types of nodes:

- A *chance node* corresponds to a random variable (indicated by a circle).
- A *decision node* corresponds to each decision to be made (indicated by a square).
- A *utility node* corresponds to an additive utility component (indicated by a diamond).

There are three kinds of directed edges:

- A *conditional edge* ends in a chance node and indicates that the uncertainty in that chance node is conditioned on the values of all of its parents.
- An *informational edge* ends in a decision node and indicates that the decision associated with that node is made with knowledge of the values of its parents. (These edges are often drawn with dashed lines and are sometimes omitted from diagrams for simplicity.)

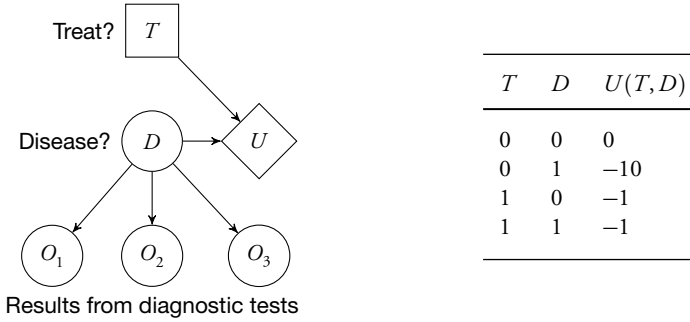


Figure 3.3 Diagnostic test decision network and utility function.

- A *functional edge* ends in a utility node and indicates that the utility node is determined by the outcomes of its parents.

Decision networks are sometimes called *influence diagrams*. Like Bayesian networks, decision networks cannot have cycles. Representing a decision problem as a decision network allows us to take advantage of the structure of the problem when computing the optimal decision with respect to a utility function. This chapter focuses on *single-shot* decision problems, in which decisions are made simultaneously. The next chapter will focus on problems for which decisions can be made sequentially.

Figure 3.3 shows an example decision network. In this network, we have a set of results from diagnostic tests that may indicate the presence of a particular disease. We need to decide whether to apply a treatment based on what is known about the diagnostic tests. The utility is a function of whether a treatment is applied and whether the disease is actually present. This network is used as a running example in this section.

3.2.1 Evaluating Decision Networks

As introduced in Section 3.1.3, the expected utility of a given o is

$$EU(a | o) = \sum_{s'} P(s' | a, o) U(s'). \quad (3.11)$$

The s' in Equation (3.11) represents an instantiation of the nodes in the decision network. We can use the equation to compute the expected utility of treating a disease for the decision network in Figure 3.3. For now, we will assume that we have the result from only the first diagnostic test and that it came back positive. If we wanted to make the knowledge of the first diagnostic test explicit in the diagram, then we would draw

an informational edge from O_1 to T , and we would have

$$EU(t^1 | o_1^1) = \sum_{o_3} \sum_{o_2} \sum_d P(d, o_2, o_3 | t^1, o_1^1) U(t^1, d, o_1^1, o_2, o_3). \quad (3.12)$$

We can use the chain rule for Bayesian networks and the definition of conditional probability to compute $P(d, o_2, o_3 | t^1, o_1^1)$. Because the utility node depends only on whether the disease is present and whether we treat it, we can simplify $U(t^1, d, o_1^1, o_2, o_3)$ to $U(t^1, d)$. Hence,

$$EU(t^1 | o_1^1) = \sum_d P(d | t^1, o_1^1) U(t^1, d). \quad (3.13)$$

Any of the exact or approximate inference methods introduced in the previous section can be used to evaluate $P(d | t^1, o_1^1)$. To decide whether to apply a treatment, we compute $EU(t^1 | o_1^1)$ and $EU(t^0 | o_1^1)$ and make the decision that provides the highest expected utility.

To evaluate general single-shot decision networks, we begin by instantiating the action nodes and observed chance nodes. We then apply any inference algorithm to compute the posterior over the parents of the utility nodes. Instead of summing over the instantiation of all variables in Equation (3.11), we only need to sum over the instantiations of the parents of the utility nodes. The optimal decision is the one that, when instantiated in the decision network, provides the highest expected utility.

A variety of methods have been developed over the years to make evaluating decision networks more efficient. One method involves removing action and chance nodes from decision networks if they have no children, as defined by conditional, informational, or functional edges. For example, in Figure 3.3, we can remove O_2 and O_3 because they have no children. We cannot remove O_1 because we treated it as observed, indicating there is an informational edge from O_1 to T (although it is not drawn explicitly).

3.2.2 Value of Information

So far, we have assumed that for the decision network in Figure 3.3, we have only observed o_1^1 . Given the positive result from that one diagnostic test alone, then we may decide against treatment. However, it may be beneficial to administer additional diagnostic tests to reduce the risk of not treating a disease that is really present. One way to decide which diagnostic test to administer is to compute the *value of information*.

In computing the value of information, we will use $EU^*(o)$ to denote the expected utility of an optimal action given observation o . The value of information about variable O' given o is

$$VOI(O' | o) = \left(\sum_{o'} P(o' | o) EU^*(o, o') \right) - EU^*(o). \quad (3.14)$$

In other words, the value of information about a variable is the increase in expected utility with the observation of that variable. The expected utility can only increase if the observation of the variable can lead to a different optimal decision. If observing a new variable O' makes no difference in the choice of action, then $EU^*(o, o') = EU^*(o)$ for all o' , in which case Equation (3.14) evaluates to 0. For example, if the optimal decision is to treat the disease regardless of the outcome of the diagnostic test, then the value of observing the outcome of the test is 0.

The value of information only captures the increase in expected utility from making an observation. There may be a cost associated with making a particular observation. Some diagnostic tests may be quite inexpensive, such as a temperature reading; other diagnostic tests are more costly and invasive, such as a lumbar puncture. The value of information obtained by a lumbar puncture may be much greater than a temperature reading, but the costs of the tests should be taken into consideration.

The value-of-information metric is an important and often used metric for choosing what to observe. Sometimes the value-of-information metric is used to determine an appropriate sequence of observations. After each observation, the value of information is determined for the remaining unobserved variables. The unobserved variable with the greatest value of information is then selected for observation. If there are costs associated with making different observations, then these costs are subtracted from the value of information when determining which variable to observe. The process continues until it is no longer beneficial to observe any more variables. The optimal action is then chosen. This greedy selection of observations is only a heuristic and may not represent the truly optimal sequence of observations. The optimal selection of observations can be determined by using the techniques for sequential decision making introduced in later chapters.

3.2.3 Creating Decision Networks

Decision networks are a powerful framework for building decision support systems. So far, we have discussed the key elements in building decision networks and how to use them to make optimal single-shot decisions. We will briefly discuss the procedure for creating a decision network.

The first step is to identify the space of possible actions. For an airborne collision avoidance system, the actions may be to climb, descend, or do nothing. In some problems, it may be desirable to factor the space of actions into multiple decision variables. In a collision avoidance system that can recommend both horizontal and vertical maneuvers, one decision variable may govern whether we climb or descend and another variable may govern whether we turn left or right.

The next step is identifying the observed and unobserved variables relevant to the problem. If we have an electro-optical sensor on our collision avoidance system, we may

be able to observe the relative angle to another aircraft, and so this angular measurement would correspond to one of the observed chance nodes. The true position of an intruding aircraft is relevant to the problem, but it cannot be observed directly, and so it is represented by an unobserved variable in the network.

We then identify the relationships between the various chance and decision nodes. Determining these relationships can be done by using expert judgment, learning from data as discussed in Section 2.4, or a combination of both. Often designers try to choose the direction of arrows to reflect causality from one node to another.

Once the relationships of the chance and decision nodes are identified, we choose models to represent the conditional probability distributions. For discrete nodes, an obvious model is a tabular representation. For continuous nodes, we can choose a parametric model such as a linear Gaussian model. The parameters of these models can then be specified by experts or estimated from the data by using the techniques introduced in Section 2.3.

We introduce the utility nodes and add functional edges from the relevant chance and decision nodes. The parameters of the utility nodes can be determined from preference elicitation from human experts (Section 3.1.4). The parameters can also be tuned so that the optimal decision according to the decision network matches the decisions of human experts.

The decision network should then be validated and refined by human experts. Given a decision scenario, the decision network can be used to determine the optimal action. That action can be compared against what a human expert would recommend. Generally, inspection of many decision scenarios is required before confidence can be established in a decision network.

If there is disagreement between the decision network and a human expert, then the decision network can be inspected to determine why that particular action was selected as optimal. In some cases, closer inspection of the model may result in revising the conditional probabilities, modifying the relationship between variables, changing the parameters in the utility nodes, or introducing new variables into the model. Sometimes further study results in the human experts revising their choice of action. Several development iterations may be required before an appropriate decision network is found.

3.3 Games

This chapter has focused on making rational decisions with an assumed model of the environment. The methods introduced in this chapter so far can, of course, be applied to environments containing other agents so long as our probabilistic model captures the effects of the behavior of the other agents. However, there are many cases in which we do not have a probabilistic model of the behavior of the other agents, but we do have a

		Agent 2	
		Testify	Refuse
Agent 1	Testify	-5, -5	0, -10
	Refuse	-10, 0	-1, -1

Figure 3.4 Prisoner's dilemma.

model of their utilities. Making decisions in such situations is the subject of *game theory*, which will be briefly discussed in this section.

3.3.1 Dominant Strategy Equilibrium

One of the most famous problems in game theory is the *prisoner's dilemma*. We have two agents who are prisoners and are being interrogated separately. Each is given the option to testify against the other. If one testifies and the other does not, then the one who testifies gets released, and the other gets ten years in prison. If both testify, then they both get five years. If both refuse to testify, then they both get one year.

The utilities of the two agents in the prisoner's dilemma are shown in Figure 3.4. The first component in the utility matrix is associated with Agent 1, and the second component is associated with Agent 2. It is assumed that the utility matrix is common knowledge between the two agents. The two agents must select their actions simultaneously without knowledge of the other agent's action.

In games like the prisoner's dilemma, the strategy chosen by an agent in a game can be either a

- *pure strategy*, in which an action is chosen deterministically, or
- *mixed strategy*, in which an action is chosen probabilistically.

Of course a pure strategy is just a special case of a mixed strategy in which probability 1 is assigned to a single action. The mixed strategy that assigns probability 0.7 to testifying in the prisoner's dilemma can be written using a notation similar to that used earlier for lotteries:

$$[\text{Testify} : 0.7; \text{Refuse} : 0.3].$$

The utility of a mixed strategy can be written in terms of utilities associated with pure strategies:

$$U([a_1 : p_1; \dots; a_n : p_n]) = \sum_{i=1}^n p_i U(a_i). \quad (3.15)$$

The strategy of agent i is denoted s_i . A *strategy profile*, $s_{1:n}$, is an assignment of strategies to all n agents. The strategy profile of all agents except for that of agent i is written s_{-i} . The utility of agent i given a strategy profile $s_{1:n}$ is written $U_i(s_{1:n})$ or, alternatively, $U_i(s_i, s_{-i})$.

A *best response* of agent i to the strategy profile s_{-i} is a strategy s_i^* such that $U_i(s_i^*, s_{-i}) \geq U_i(s_i, s_{-i})$ for all strategies s_i . There may be, in general, multiple different best responses given s_{-i} . In some games, there may exist an s_i that is a best response to all possible s_{-i} , in which case s_i is called a *dominant strategy*. For example, in the prisoner's dilemma, Agent 1 is better off testifying regardless of whether Agent 2 testifies or refuses. Hence, testifying is the dominant strategy for Agent 1. Because the game is symmetric, testifying is also the dominant strategy for Agent 2. When all agents have a dominant strategy, their combination is called a *dominant strategy equilibrium*.

The prisoner's dilemma has generated significant interest because it shows how playing individual best responses can result in a suboptimal outcome for all agents. The dominant strategy equilibrium results in both agents testifying and receiving five years in prison. However, if they both refused to testify, then they would have both only received one year in prison.

3.3.2 Nash Equilibrium

Suppose we have two aircraft on a collision course, and the pilots of each aircraft must choose between climb or descend to avoid collision. If the pilots both choose the same maneuver, then there is a crash with utility -4 to both pilots. Because climbing requires more fuel than descending, there is an additional penalty of -1 to any pilot who decides to climb. The utility matrix is shown in Figure 3.5.

In the collision avoidance game, there does not exist a dominant strategy equilibrium. The best response of a particular pilot depends on the decision of the other pilot. There is an alternative equilibrium concept known as the *Nash equilibrium*. A strategy profile is a Nash equilibrium if no agent can benefit by switching strategy, given that the other agents adhere to the profile. In other words, $s_{1:n}$ is a Nash equilibrium if s_i is a best response to s_{-i} for all agents i .

There are two pure-strategy Nash equilibria in the collision avoidance game, namely, (Climb, Descend) and (Descend, Climb). It has been proven that every game has at least one Nash equilibrium, which may or may not include pure strategies. There are no known polynomial time algorithms for finding Nash equilibria for general games,

		Agent 2	
		Climb	Descend
Agent 1	Climb	-5, -5	-1, 0
	Descend	0, -1	-4, -4

Figure 3.5 Collision avoidance game.

although the complexity of finding Nash equilibria is not NP-complete (instead it belongs to the complexity class known as PPAD).

3.3.3 Behavioral Game Theory

When one is building a decision-making system that must interact with humans, information about the Nash equilibrium is not always helpful. Humans often do not play a Nash equilibrium strategy. First of all, it may be unclear which equilibrium to adopt if there are many different equilibria in the game. For games with only one equilibrium, it may be difficult for a human to compute the Nash equilibrium because of cognitive limitations. Even if human agents can compute the Nash equilibrium, they may doubt that their opponents can perform that computation.

An area known as *behavioral game theory* aims to model human agents. Many different behavioral models exist, but the *logit level- k model*, sometimes called the *quantal level- k model*, has become popular recently and tends to work well in practice. The logit level- k model captures the assumption that humans are

- more likely to make errors when those errors are less costly, and
- limited in the number of steps of strategic look-ahead (as in “I think that you think that I think...”).

The model is defined by

- a *precision parameter* $\lambda \geq 0$, which controls sensitivity to utility differences (0 is insensitive); and
- a *depth parameter* $k > 0$, which controls the depth of rationality.

In the logit level- k model, a level-0 agent selects actions uniformly. A level-1 agent assumes the opponents adopt level-0 strategies and selects actions according to the logit distribution

$$P(a_i) \propto e^{\lambda U_i(a_i, s_{-i})}, \quad (3.16)$$

where s_{-i} represents the assumed strategy profile of the other agents. A level- k agent assumes that the other agents adopt level $k - 1$ strategies and select their own actions according to Equation (3.16). The parameters k and λ can be learned from data by using techniques discussed in the previous chapter.

To illustrate the logit level- k model, we will use the *traveler's dilemma*. In this game, an airline loses two identical suitcases from two travelers. The airline asks the travelers to write down the value of their suitcases, which can be between \$2 and \$100, inclusive. If both put down the same value, then they both get that value. The traveler with the lower value gets their value plus \$2. The traveler with the higher value gets the lower value minus \$2. In other words, the utility function is as follows:

$$U_i(a_i, a_{-i}) = \begin{cases} a_i & \text{if } a_i = a_{-i} \\ a_i + 2 & \text{if } a_i < a_{-i} \\ a_{-i} - 2 & \text{otherwise} \end{cases} \quad (3.17)$$

Most people tend to put down between \$97 and \$100. However, somewhat counterintuitively, there is a unique Nash equilibrium of only \$2.

Figure 3.6 shows the strategies of the logit level- k model with different values for λ and k . The level-0 agent selects actions uniformly. The level-1 agent is peaked toward the upper end of the spectrum, with the precision parameter controlling the spread. As k increases, the difference between the strategies for $\lambda = 0.3$ and $\lambda = 0.5$ becomes less apparent. Human behavior can often be modeled well by logit level 2; as we can see, this provides a better model of human behavior than the Nash equilibrium.

3.4 Summary

- Rational decision making combines probability and utility theory.
- The existence of utility function follows from constraints on rational preferences.
- A rational decision is one that maximizes expected utility.
- We can build rational decision-making systems based on utility functions inferred from humans.
- Humans are not always rational.
- Decision networks compactly represent decision problems.
- Behavioral game theory is useful when making decisions involving multiple agents.

3.5 Further Reading

The theory of expected utility was initiated by Bernoulli in 1793 [2]. The axioms for rational decision making presented in Section 3.1.1 are based on those of Neumann and Morgenstern in their classic text, *Theory of Games and Economic Behavior* [3]. Neumann

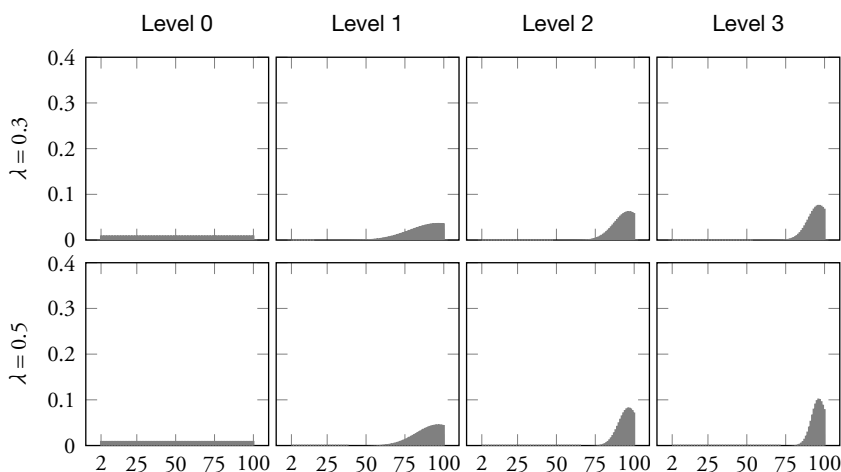


Figure 3.6 Logit level- k for traveler's dilemma.

and Morgenstern prove that those axioms lead to the existence of a utility function and establish the basis for the maximum expected utility principle [3]. Schoemaker provides an overview of the development of utility theory [4], and Fishburn surveys the field [5]. Russell and Norvig discuss the importance of the maximum expected utility principle to the field of artificial intelligence [6].

Farquhar surveys a variety of methods for utility elicitation [7], and Markowitz discusses the utility of money [8]. *Decisions with Multiple Objectives: Preferences and Value Tradeoffs* by Keeney and Raiffa provides an overview of multiple attribute utility theory [9]. The book discusses the assumptions about preference structure that allow certain kinds of utility function decompositions, including the additive decomposition discussed in Section 3.1.6.

The example of irrational preferences in Section 3.1.7 is taken from Tversky and Kahneman [1]. Kahneman and Tversky provide a critique of expected utility theory and introduce an alternative model called prospect theory that appears to be more consistent with human behavior [10]. Several recent books discuss human irrationality, including *Predictably Irrational: The Hidden Forces That Shape Our Decisions* [11] and *How We Decide* [12].

The textbook *Bayesian Networks and Decision Graphs* by Jensen and Nielsen discusses decision networks. Early papers by Shachter provide algorithms for evaluating decision networks [14], [15]. Howard introduces the quantitative concept of the value of information [16], which has been applied to decision networks [17], [18].

Game theory is a broad field, and there are several standard introductory books [19]–[21]. Koller and Milch extend decision networks to a game-theoretic context [22].

Daskalakis, Goldberg, and Papadimitriou discuss the complexity of computing Nash equilibria [23]. Camerer provides an overview of behavioral game theory [24]. Wright and Leyton-Brown discuss behavioral game theory and show how to extract parameters from experimental data of human behavior [25], [26].

References

1. A. Tversky and D. Kahneman, “The Framing of Decisions and the Psychology of Choice,” *Science*, vol. 211, no. 4481, pp. 453–458, 1981. doi: 10.1126/science.7455683.
2. D. Bernoulli, “Exposition of a New Theory on the Measurement of Risk,” *Econometrica*, vol. 22, no. 1, pp. 23–36, 1954. doi: 10.2307/1909829.
3. J.V. Neumann and O. Morgenstern, *Theory of Games and Economic Behavior*, 3rd ed. Princeton, NJ: Princeton University Press, 1953.
4. P.J.H. Schoemaker, “The Expected Utility Model: Its Variants, Purposes, Evidence and Limitations,” *Journal of Economic Literature*, vol. 20, no. 2, pp. 529–563, 1982.
5. P.C. Fishburn, “Utility Theory,” *Management Science*, vol. 14, no. 5, pp. 335–378, 1968.
6. S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Upper Saddle River, NJ: Pearson, 2010.
7. P.H. Farquhar, “Utility Assessment Methods,” *Management Science*, vol. 30, no. 11, pp. 1283–1300, 1984.
8. H. Markowitz, “The Utility of Wealth,” *Journal of Political Economy*, vol. 60, no. 2, pp. 151–158, 1952.
9. R.L. Keeney and H. Raiffa, *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*. New York: Cambridge University Press, 1993.
10. D. Kahneman and A. Tversky, “Prospect Theory: An Analysis of Decision Under Risk,” *Econometrica*, vol. 47, no. 2, pp. 263–292, 1979. doi: 10.2307/1914185.
11. D. Ariely, *Predictably Irrational: The Hidden Forces That Shape Our Decisions*. New York: Harper, 2008.
12. J. Lehrer, *How We Decide*. New York: Houghton Mifflin, 2009.
13. F.V. Jensen and T.D. Nielsen, *Bayesian Networks and Decision Graphs*, 2nd ed. New York: Springer, 2007.
14. R.D. Shachter, “Evaluating Influence Diagrams,” *Operations Research*, vol. 34, no. 6, pp. 871–882, 1986.

15. ———, “Probabilistic Inference and Influence Diagrams,” *Operations Research*, vol. 36, no. 4, pp. 589–604, 1988.
16. R.A. Howard, “Information Value Theory,” *IEEE Transactions on Systems Science and Cybernetics*, vol. 2, no. 1, pp. 22–26, 1966. doi: 10.1109/TSSC.1966.300074.
17. S.L. Dittmer and F.V. Jensen, “Myopic Value of Information in Influence Diagrams,” in *Conference on Uncertainty in Artificial Intelligence (UAI)*, 1997.
18. R.D. Shachter, “Efficient Value of Information Computation,” in *Conference on Uncertainty in Artificial Intelligence (UAI)*, 1999.
19. R.B. Myerson, *Game Theory: Analysis of Conflict*. Cambridge, MA: Harvard University Press, 1997.
20. Y. Shoham and K. Leyton-Brown, *Multiagent Systems: Algorithmic, Game Theoretic, and Logical Foundations*. New York: Cambridge University Press, 2009.
21. D. Fudenberg and J. Tirole, *Game Theory*. Cambridge, MA: MIT Press, 1991.
22. D. Koller and B. Milch, “Multi-Agent Influence Diagrams for Representing and Solving Games,” *Games and Economic Behavior*, vol. 45, no. 1, pp. 181–221, 2003. doi: 10.1016/S0899-8256(02)00544-4.
23. C. Daskalakis, P.W. Goldberg, and C.H. Papadimitriou, “The Complexity of Computing a Nash Equilibrium,” *Communications of the ACM*, vol. 52, no. 2, pp. 89–97, 2009. doi: 10.1145/1461928.1461951.
24. C.F. Camerer, *Behavioral Game Theory: Experiments in Strategic Interaction*. Princeton, NJ: Princeton University Press, 2003.
25. J.R. Wright and K. Leyton-Brown, “Behavioral Game Theoretic Models: A Bayesian Framework for Parameter Analysis,” in *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2012.
26. J.R. Wright and K. Leyton-Brown, “Beyond Equilibrium: Predicting Human Behavior in Normal Form Games,” in *AAAI Conference on Artificial Intelligence (AAAI)*, 2010.

4

Sequential Problems

Mykel J. Kochenderfer

The previous chapter discussed problems in which a single decision is to be made, but many important problems require the decision maker to make a series of decisions. The same principle of maximum expected utility still applies, but optimal decision making requires reasoning about future sequences of actions and observations. This chapter will discuss sequential decision problems in stochastic environments.

4.1 Formulation

In this chapter, we will focus on a general formulation of sequential decision problems under the assumption that the model is known and that the environment is fully observable. Both of these assumptions will be relaxed in the next two chapters.

4.1.1 Markov Decision Processes

In a *Markov decision process* (MDP), an agent chooses action a_t at time t based on observing state s_t . The agent then receives a reward r_t . The state evolves probabilistically based on the current state and action taken by the agent. The assumption that the next state depends only on the current state and action and not on any prior state or action is known as the *Markov assumption*.

An MDP can be represented using a decision network as shown in Figure 4.1a. There are information edges (not shown in the figure) from $A_{0:t-1}$ and $S_{0:t}$ to A_t . The utility function is decomposed into rewards $R_{0:t}$.

We will focus on *stationary* MDPs in which $P(S_{t+1} \mid S_t, A_t)$ and $P(R_t \mid A_t, S_t)$ do not vary with time. Stationary MDPs can be compactly represented by a dynamic decision diagram as shown in Figure 4.1b. The *state transition function* $T(s' \mid s, a)$ represents the probability of transitioning from state s to s' after executing action a . The *reward function* $R(s, a)$ represents the expected reward received when executing

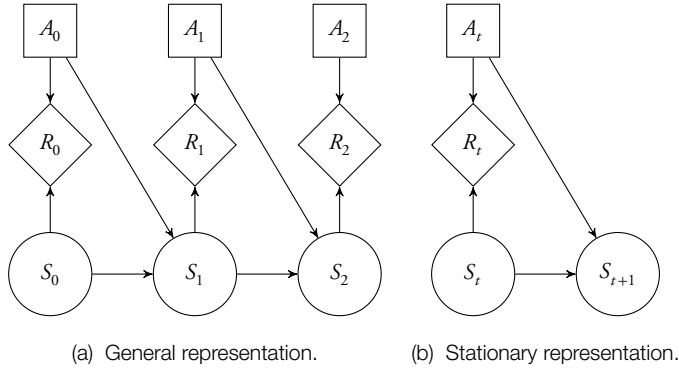


Figure 4.1 Markov decision process decision diagram.

action a from state s . We assume that the reward function is a deterministic function of s and a , but this need not be the case.

The problem of aircraft collision avoidance can be formulated as an MDP. The states represent the positions and velocities of our aircraft and the intruder aircraft, and the actions represent whether we climb, descend, or stay level. We receive a large negative reward for colliding with the other aircraft and a small negative reward for climbing or descending.

4.1.2 Utility and Reward

The rewards in an MDP are treated as components in an additively decomposed utility function (Section 3.1.6). In a *finite horizon* problem with n decisions, the utility associated with a sequence of rewards $r_{0:n-1}$ is simply

$$\sum_{t=0}^{n-1} r_t. \quad (4.1)$$

In an *infinite horizon* problem in which the number of decisions is unbounded, the sum of rewards can become infinite. Suppose strategy A results in a reward of 1 per time step and strategy B results in a reward of 100 per time step. Intuitively, a rational agent should prefer strategy B over strategy A , but both provide the same infinite expected utility.

There are several ways to define utility in terms of individual rewards in infinite horizon problems. One way is to impose a *discount factor* γ between 0 and 1. The utility

is given by

$$\sum_{t=0}^{\infty} \gamma^t r_t. \quad (4.2)$$

So long as $0 \leq \gamma < 1$ and the rewards are finite, the utility will be finite. The discount factor makes it so that rewards in the present are worth more than rewards in the future, a concept that also appears in economics.

Another way to define utility in infinite horizon problems is to use the *average reward* given by

$$\lim_{n \rightarrow \infty} \frac{1}{n} \sum_{t=0}^{n-1} r_t. \quad (4.3)$$

This book focuses primarily on optimizing with respect to discounted rewards over an infinite horizon.

4.2 Dynamic Programming

The optimal strategy can be found by using a computational technique called *dynamic programming*. Although we will focus on dynamic programming algorithms for MDPs, dynamic programming is a general technique that can be applied to a wide variety of other problems. For example, dynamic programming can be used in computing a Fibonacci sequence, finding the longest common subsequence between two strings, and finding the most likely sequence of states in a hidden Markov model. In general, algorithms that use dynamic programming for solving MDPs are much more efficient than brute force methods.

4.2.1 Policies and Utilities

A *policy* in an MDP determines what action to select given the past history of states and actions. The action to select at time t , given the history $h_t = (s_{0:t}, a_{0:t-1})$, is written $\pi_t(h_t)$. Because the future state sequence and rewards depend only on the current state and action (as made apparent in the conditional independence assumptions in Figure 4.1a), we can restrict our attention to policies that depend only on the current state.

In infinite horizon MDPs in which the transitions and rewards are stationary, we can further restrict our attention to stationary policies. We will write the action associated with stationary policy π in state s as $\pi(s)$, without the temporal subscript. In finite horizon problems, however, it may be beneficial to select a different action depending on how many time steps are remaining. For example, when playing basketball, it is generally not a good strategy to attempt a half-court shot unless there are just a couple seconds remaining in the game.

The expected utility of executing π from state s is denoted $U^\pi(s)$. In the context of MDPs, U^π is often referred to as the *value function*. An *optimal policy* π^* is a policy that maximizes expected utility:

$$\pi^*(s) = \arg \max_{\pi} U^\pi(s) \quad (4.4)$$

for all states s . Depending on the model, there may be multiple policies that are optimal.

4.2.2 Policy Evaluation

Computing the expected utility obtained from executing a policy is known as *policy evaluation*. We may use dynamic programming to evaluate the utility of a policy π for t steps. If we do not execute the policy at all, then $U_0^\pi(s) = 0$. If we execute the policy for one step, then $U_1^\pi(s) = R(s, \pi(s))$, which is simply the expected reward associated with the first step.

Suppose we know the utility associated with executing π for $t - 1$ steps. Computing the utility associated with executing π for t steps can be computed as follows:

$$U_t^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s'} T(s' | s, \pi(s)) U_{t-1}^\pi(s'), \quad (4.5)$$

where γ is the discount factor, which can be set to 1 if discounting is not desired. Algorithm 4.1 shows how to iteratively compute the expected utility of a policy up to an arbitrary horizon n .

Algorithm 4.1 Iterative policy evaluation

```

1: function ITERATIVEPOLICYEVALUATION( $\pi, n$ )
2:    $U_0^\pi(s) \leftarrow 0$  for all  $s$ 
3:   for  $t \leftarrow 1$  to  $n$ 
4:      $U_t^\pi(s) \leftarrow R(s, \pi(s)) + \gamma \sum_{s'} T(s' | s, \pi(s)) U_{t-1}^\pi(s')$  for all  $s$ 
5:   return  $U_n^\pi$ 

```

For an infinite horizon with discounted rewards,

$$U^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s'} T(s' | s, \pi(s)) U^\pi(s'). \quad (4.6)$$

We can compute U^π arbitrarily well with enough iterations of iterative policy evaluation. An alternative is to solve a system of n linear equations where n is the number of states. We can represent the system of equations in matrix form:

$$\mathbf{U}^\pi = \mathbf{R}^\pi + \gamma \mathbf{T}^\pi \mathbf{U}^\pi, \quad (4.7)$$

where $\mathbf{U}^\pi$ and $\mathbf{R}^\pi$ are the utility and reward functions represented as an n -dimensional vector. The $n \times n$ matrix $\mathbf{T}^\pi$ contains state transition probabilities. The probability of transitioning from the i th state to the j th state is given by $\mathbf{T}_{ij}^\pi$.

We can easily solve for $\mathbf{U}^\pi$ as follows:

$$\mathbf{U}^\pi - \gamma \mathbf{T}^\pi \mathbf{U}^\pi = \mathbf{R}^\pi \quad (4.8)$$

$$(\mathbf{I} - \gamma \mathbf{T}^\pi) \mathbf{U}^\pi = \mathbf{R}^\pi \quad (4.9)$$

$$\mathbf{U}^\pi = (\mathbf{I} - \gamma \mathbf{T}^\pi)^{-1} \mathbf{R}^\pi. \quad (4.10)$$

Solving for $\mathbf{U}^\pi$ in this way requires $O(n^3)$ time.

4.2.3 Policy Iteration

Policy evaluation can be used in a general process called *policy iteration* for computing an optimal policy π^* as outlined in Algorithm 4.2. Policy iteration starts with any policy π_0 and iterates the following two steps:

- *Policy evaluation.* Given the current policy π_k , compute $\mathbf{U}^{\pi_k}$.
- *Policy improvement.* Using $\mathbf{U}^{\pi_k}$, compute a new policy using the equation in Line 5.

The algorithm terminates when there is no more improvement. Because every step leads to improvement and there are finitely many policies, the algorithm terminates with an optimal solution.

Algorithm 4.2 Policy iteration

```

1: function POLICYITERATION( $\pi_0$ )
2:    $k \leftarrow 0$ 
3:   repeat
4:     Compute  $\mathbf{U}^{\pi_k}$ 
5:      $\pi_{k+1}(s) = \arg \max_a (R(s, a) + \gamma \sum_{s'} T(s' | s, a) \mathbf{U}^{\pi_k}(s'))$  for all states  $s$ 
6:      $k \leftarrow k + 1$ 
7:   until  $\pi_k = \pi_{k-1}$ 
8:   return  $\pi_k$ 

```

There are many variants of policy iteration. One method known as *modified policy iteration* involves approximating $\mathbf{U}^{\pi_k}$ using only a few iterations of iterative policy evaluation instead of computing the utility function exactly.

4.2.4 Value Iteration

An alternative to policy iteration is *value iteration* (Algorithm 4.3), which is often used because it is simple and easy to implement. First, let us compute the optimal value

function U_n associated with a horizon of n and no discounting. If $n = 0$, then $U_0(s) = 0$ for all s . We can compute U_n recursively from this base case

$$U_n(s) = \max_a \left(R(s, a) + \sum_{s'} T(s' | s, a) U_{n-1}(s') \right). \quad (4.11)$$

For an infinite horizon problem with discount γ , it can be proven that the value of an optimal policy satisfies the *Bellman equation*:

$$U^*(s) = \max_a \left(R(s, a) + \gamma \sum_{s'} T(s' | s, a) U^*(s') \right). \quad (4.12)$$

The optimal value function U^* appears on both sides of the equation. Value iteration approximates U^* by iteratively updating the estimate of U^* using Equation (4.12). Once we know U^* , we can extract an optimal policy by using

$$\pi(s) \leftarrow \arg \max_a \left(R(s, a) + \gamma \sum_{s'} T(s' | s, a) U^*(s') \right). \quad (4.13)$$

Algorithm 4.3 Value iteration

```

1: function VALUEITERATION
2:    $k \leftarrow 0$ 
3:    $U_0(s) \leftarrow 0$  for all states  $s$ 
4:   repeat
5:      $U_{k+1}(s) \leftarrow \max_a [R(s, a) + \gamma \sum_{s'} T(s' | s, a) U_k(s')] \text{ for all states } s$ 
6:      $k \leftarrow k + 1$ 
7:   until convergence
8:   return  $U_k$ 

```

Algorithm 4.3 shows U_0 being initialized to 0, but value iteration can be proven to converge with any bounded initialization (i.e., $|U_0(s)| < \infty$ for all s). It is common to initialize the utility function to a guess of the optimal value function in an attempt to speed convergence.

A common termination condition for the loop in Algorithm 4.3 is when $\|U_k - U_{k-1}\| < \delta$. In this context, $\|\cdot\|$ denotes the *max norm*, where $\|U\| = \max_s |U(s)|$. The quantity $\|U_k - U_{k-1}\|$ is known as the *Bellman residual*.

If we want to guarantee that our estimate of the value function is within ϵ of U^* at all states, then we should choose δ to be $\epsilon(1-\gamma)/\gamma$. As γ approaches 1, the termination threshold becomes smaller, implying slower convergence. In general, the less future rewards are discounted, the more we have to iterate to look out to an acceptable horizon.

If we know that $\|U_k - U^*\| < \epsilon$, then we can bound the *policy loss* of the policy extracted from U_k . If the extracted policy is π , then the policy loss is defined as $\|U^\pi - U^*\|$. It can be proven that $\|U_k - U^*\| < \epsilon$ implies that the policy loss is less than 2ϵ .

4.2.5 Grid World Example

To illustrate value iteration, we will use a 10×10 grid world problem. Each cell in the grid represents a state in an MDP. The available actions are up, down, left, and right. The effects of these actions are stochastic. We move one step in the specified direction with probability 0.7, and we move one step in one of the three other directions, each with probability 0.1. If we bump against the outer border of the grid, then we do not move at all.

We receive a cost of 1 for bumping against the outer border of the grid. There are four cells in which we receive rewards upon entering:

- (8, 9) has a reward of +10
- (3, 8) has a reward of +3
- (5, 4) has a reward of −5
- (8, 4) has a reward of −10

The coordinates are specified using the matrix convention in which the first coordinate is the row starting from the top and the second coordinate is the column starting from the left. The cells with rewards of +10 and +3 are absorbing states where no additional reward is ever received from that point onward.

Figure 4.2a shows the result of the first sweep of value iteration with a discount factor of 0.9. After this first sweep, the value function is simply the maximum expected immediate reward—i.e., $\max_a R(s, a)$. The gray pointers indicate the optimal actions from the cells as determined by Equation (4.13). As indicated in the figure, all actions are optimal for the interior cells. For the cells adjacent to the wall, the optimal actions are in the directions away from the wall.

Figure 4.2b shows the result of the second sweep. The values at the states with non-zero rewards remain the same, but the values are dispersed to adjacent cells. The value of the cells is based on expected discounted rewards after two time steps. Consequently, cells more than one step away from an absorbing cell or a cell bordering a wall have zero value. Cells within one step have had their optimal action set updated to direct us to positive rewards and away from negative rewards.

Figures 4.3a and 4.3b show the value function and policy after three and four sweeps, respectively. The value associated with the +3 and +10 cells spread outward over the grid. As the value is propagated further throughout the grid, there are fewer ties for optimal actions for the various cells.

Figures 4.4a and 4.4b show the value function and policy at convergence for $\gamma = 0.9$ and $\gamma = 0.5$, respectively. When $\gamma = 0.9$, even the cells on the left side of the grid have

positive value. When the rewards are discounted more steeply with $\gamma = 0.5$, the +3 and +10 rewards do not propagate as far. The effect of steeper discounting can also be seen in the difference in policy at cell (4, 8). With discounting at 0.5, the best strategy is to head straight to the +3 cell, whereas with discounting at 0.9, the best strategy is to head toward the +10 cell.

4.2.6 Asynchronous Value Iteration

The value iteration algorithm in Section 4.2.4 computes U_{k+1} based on U_k for *all* the states at each iteration. In *asynchronous value iteration*, only a subset of the states may be updated per iteration. It can be proven that, so long as the value function is updated at each state infinitely often, the value function is guaranteed to converge to the optimal value function.

Gauss-Seidel value iteration is a type of asynchronous value iteration. It sweeps through an ordering of the states and applies the following update:

$$U(s) \leftarrow \max_a \left(R(s, a) + \gamma \sum_{s'} T(s' | s, a) U(s') \right). \quad (4.14)$$

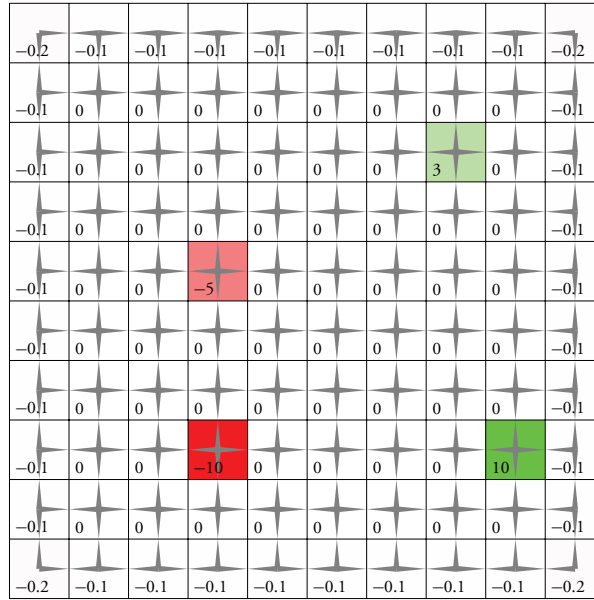
With Gauss-Seidel value iteration, we only have to keep one copy of state values in memory instead of two because the values are updated *in place*. In addition, Gauss-Seidel can converge more quickly than standard value iteration can depending on the ordering chosen.

4.2.7 Closed- and Open-Loop Planning

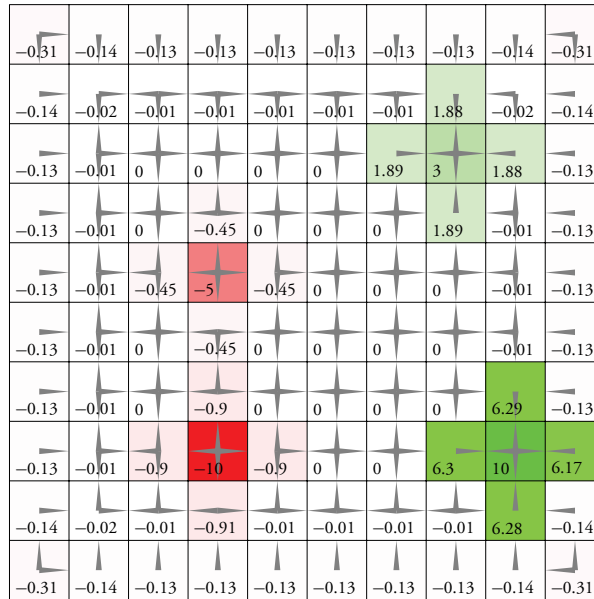
The process of using a model to choose an action in a sequential problem is called *planning*. There are two general approaches to planning:

- *Closed-loop planning* accounts for future state information. The dynamic programming algorithms discussed in this chapter fall within this category. They involve developing a *reactive* plan (or policy) that can react to the different outcomes of the actions over time.
- *Open-loop planning* does not account for future state information. Many *path planning* algorithms fall within this category. They involve developing a static sequence of actions.

The advantage of closed-loop planning can be illustrated with the example in Figure 4.5. There are nine states, and we start in state s_0 . There are two decision steps, where we must decide between going up (black arrows) or going down (gray arrows). The effects of the actions are deterministic, except that if we go up from s_0 , then we end up in state s_1 half the time and in state s_2 half the time. We receive a reward of 30 in states s_4 and s_6 and a reward of 20 in states s_7 and s_8 , as indicated in the figure.



(a) Sweep 1.

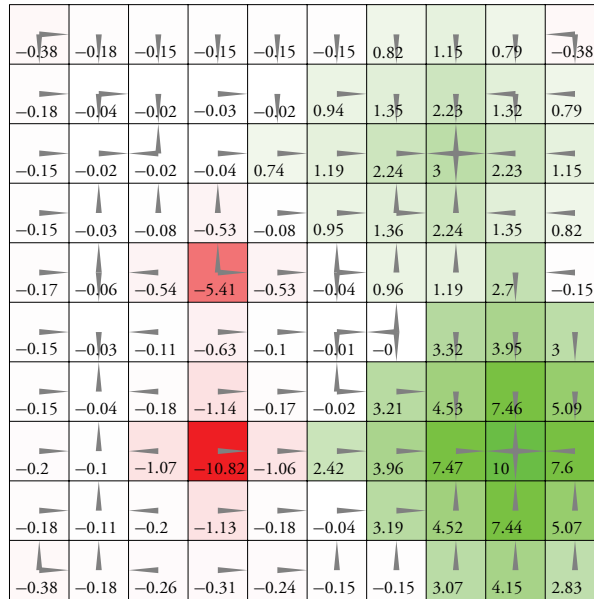


(b) Sweep 2.

Figure 4.2 Value iteration with $\gamma = 0.9$.



(a) Sweep 3.



(b) Sweep 4.

Figure 4.3 Value iteration with $\gamma = 0.9$.

0.41	0.74	0.96	1.18	1.43	1.71	1.98	2.11	2.39	2.09
0.74	1.04	1.27	1.52	1.81	2.15	2.47	2.58	3.02	2.69
0.86	1.18	1.45	1.76	2.15	2.55	2.97	3	3.69	3.32
0.84	1.11	1.31	1.55	2.45	3.01	3.56	4.1	4.53	4.04
0.91	1.2	1.09	-3	2.48	3.53	4.21	4.93	5.5	4.88
1.1	1.46	1.79	2.24	3.42	4.2	4.97	5.85	6.68	5.84
1.06	1.41	1.7	2.14	3.89	4.9	5.85	6.92	8.15	6.94
0.92	1.18	0.7	-7.39	3.43	5.39	6.67	8.15	10	8.19
1.09	1.45	1.75	2.18	3.89	4.88	5.84	6.92	8.15	6.94
1.07	1.56	2.05	2.65	3.38	4.11	4.92	5.83	6.68	5.82

(a) $\gamma = 0.9$.

-0.28	-0.13	-0.12	-0.11	-0.09	-0.04	0.08	0.31	0.07	-0.19
-0.13	-0.01	0	0.02	0.07	0.18	0.46	1.11	0.45	0.07
-0.12	-0	0.01	0.04	0.15	0.42	1.12	3	1.11	0.31
-0.12	-0.01	-0.02	-0.24	0.05	0.19	0.47	1.12	0.48	0.09
-0.13	-0.02	-0.27	-5.12	-0.23	0.08	0.2	0.46	0.54	0.13
-0.12	-0.01	-0.04	-0.28	0.02	0.11	0.28	0.65	1.39	0.53
-0.12	-0.02	-0.06	-0.51	0.05	0.26	0.64	1.55	3.72	1.49
-0.13	-0.04	-0.53	-10.19	-0.33	0.5	1.39	3.72	10	3.74
-0.14	-0.03	-0.07	-0.51	0.04	0.25	0.63	1.55	3.72	1.49
-0.28	-0.14	-0.15	-0.18	-0.1	-0.01	0.16	0.54	1.32	0.43

(b) $\gamma = 0.5$.**Figure 4.4** Value iteration at convergence.

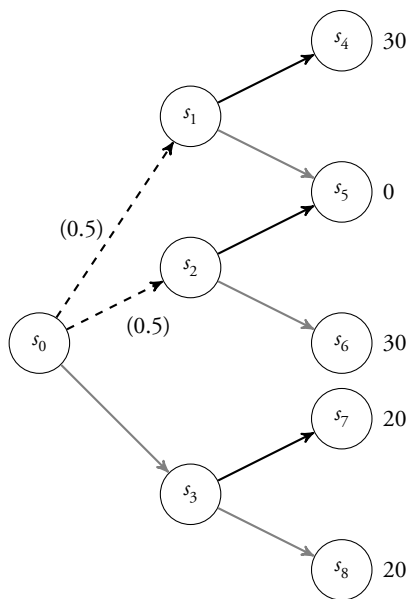


Figure 4.5 Example illustrating suboptimality of open-loop planning.

There are exactly four open-loop plans: (up, up), (up, down), (down, up), and (down, down). In this simple example, it is easy to compute their expected utilities:

- $U(\text{up, up}) = 0.5 \times 30 + 0.5 \times 0 = 15$
- $U(\text{up, down}) = 0.5 \times 0 + 0.5 \times 30 = 15$
- $U(\text{down, up}) = 20$
- $U(\text{down, down}) = 20$

According to the set of open-loop plans, it is best to choose down from s_0 because our expected reward is 20 instead of 15.

Closed-loop planning, in contrast, takes into account the fact that we can base our next decision on the observed outcome of our first action. If we choose to go up from s_0 , then we can choose to go down or up depending on whether we end up in s_1 or s_2 , thereby guaranteeing a reward of 30.

In sequential problems where the effects of actions are uncertain, closed-loop planning can provide a significant benefit over open-loop planning. However, in some domains, the size of the state space can make the application of closed-loop planning methods, such as value iteration, infeasible. Open-loop planning algorithms, although suboptimal in principle, can provide satisfactory performance. There are many open-loop planning algorithms, but this book will focus on closed-loop methods and ways for addressing

large problems without sacrificing the ability to account for the availability of future information.

4.3 Structured Representations

The dynamic programming algorithms described earlier in this chapter have assumed that the state space is discrete. If the state space is determined by n binary variables, then the number of discrete states is 2^n . This exponential growth of discrete states restricts the direct application of algorithms such as value iteration and policy iteration to problems with only a limited number of state variables. This section discusses methods that can help solve higher dimensional problems by leveraging their structure.

4.3.1 Factored Markov Decision Processes

A *factored Markov decision process* compactly represents the transition and reward functions using a dynamic decision network. The actions, rewards, and states may be factored into multiple nodes. Figure 4.6 shows an example of a factored MDP with two decision variables (A and F), three state variables (B , D , and G), and two reward variables (C and E).

We can use decision trees to compactly represent the conditional probability distributions and reward functions. For example, the conditional probability distribution $P(G_{t+1} \mid D_t, F_t, G_t)$ shown in tabular form in Figure 4.7a can be represented using the decision tree in Figure 4.7b.

Additional efficiency can be gained using *decision diagrams* instead of decision trees. In trees, all nodes (except for the root) have exactly one parent, but nodes in decision diagrams can have multiple parents. Figure 4.8 shows an example of a decision tree and an equivalent decision diagram. Instead of requiring five leaf nodes as in the decision tree, the decision diagram only requires two leaf nodes.

4.3.2 Structured Dynamic Programming

Several dynamic programming algorithms exist for finding policies for factored MDPs. Algorithms such as *structured value iteration* and *structured policy iteration* perform updates on the leaves of the decision trees instead of all the states. These algorithms improve efficiency by *aggregating states* and leveraging the additive decomposition of the reward and value functions. The resulting policy is represented as a decision tree in which the interior nodes correspond to tests of the state variables and the leaf nodes correspond to actions.

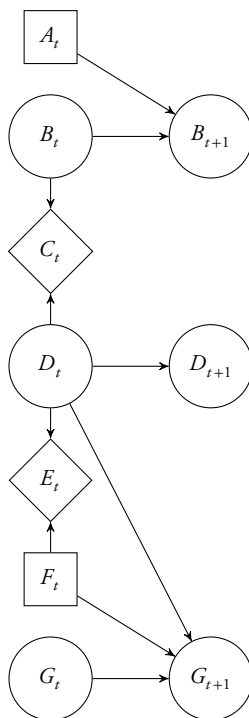


Figure 4.6 Factored Markov decision process.

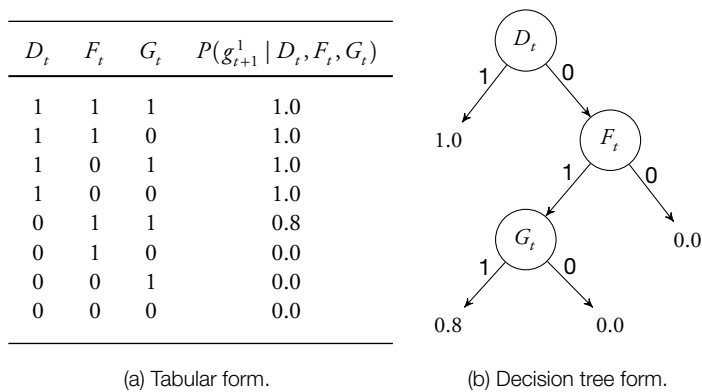


Figure 4.7 Conditional distribution as a decision tree.

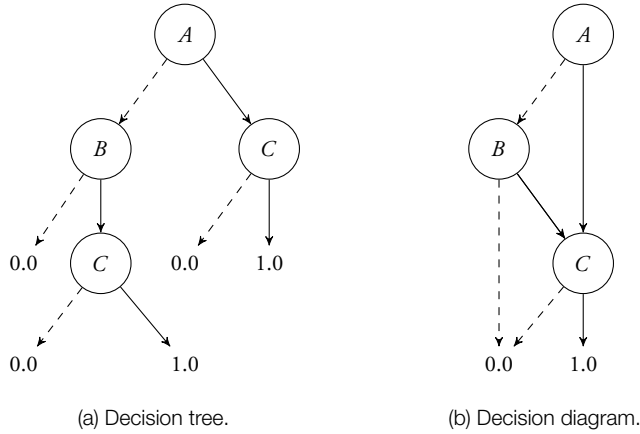


Figure 4.8 Tree- and graph-based representations of a conditional probability table. Dashed lines are followed from nodes when the outcome of the variable test is false.

4.4 Linear Representations

The methods presented in this chapter so far have required that the problems be discrete. Of course, we may discretize a problem that is naturally continuous, but doing so may not be feasible if the state or action spaces are large. This section presents a method for finding exact optimal policies for problems with continuous state and action spaces that meet certain criteria:

- *Dynamics are linear Gaussian.* The state transition function has the form

$$T(\mathbf{z} \mid \mathbf{s}, \mathbf{a}) = \mathcal{N}(\mathbf{z} \mid \mathbf{T}_s \mathbf{s} + \mathbf{T}_a \mathbf{a}, \Sigma), \quad (4.15)$$

where $\mathbf{T}_s$ and $\mathbf{T}_a$ are matrices that determine the mean of the next state $\mathbf{z}$ based on $\mathbf{s}$ and $\mathbf{a}$, and Σ is a covariance matrix that controls the amount of noise in the dynamics.

- *Reward is quadratic.* The reward function has the following form

$$R(\mathbf{s}, \mathbf{a}) = \mathbf{s}^\top \mathbf{R}_s \mathbf{s} + \mathbf{a}^\top \mathbf{R}_a \mathbf{a}, \quad (4.16)$$

where $\mathbf{R}_s = \mathbf{R}_s^\top \leq 0$ and $\mathbf{R}_a = \mathbf{R}_a^\top < 0$.

For simplicity, we will assume a finite horizon undiscounted reward problem, but the approach can be generalized to average reward and discounted infinite horizon problems as well. We may generalize Equation (4.11) for continuous state spaces by replacing the summation with an integral and making $T(s' \mid s, a)$ represent a probability density

rather than a probability mass:

$$U_n(\mathbf{s}) = \max_{\mathbf{a}} \left(R(\mathbf{s}, \mathbf{a}) + \int T(\mathbf{z} | \mathbf{s}, \mathbf{a}) U_{n-1}(\mathbf{z}) d\mathbf{z} \right). \quad (4.17)$$

From our assumptions about T and R , we can rewrite the equation above:

$$U_n(\mathbf{s}) = \max_{\mathbf{a}} \left(\mathbf{s}^\top \mathbf{R}_s \mathbf{s} + \mathbf{a}^\top \mathbf{R}_a \mathbf{a} + \int \mathcal{N}(\mathbf{z} | \mathbf{T}_s \mathbf{s} + \mathbf{T}_a \mathbf{a}, \Sigma) U_{n-1}(\mathbf{z}) d\mathbf{z} \right). \quad (4.18)$$

It can be proven inductively that $U_n(\mathbf{s})$ can be written in the form $\mathbf{s}^\top \mathbf{V}_n \mathbf{s} + q_n$. We can rewrite the equation above as

$$U_n(\mathbf{s}) = \max_{\mathbf{a}} \left(\mathbf{s}^\top \mathbf{R}_s \mathbf{s} + \mathbf{a}^\top \mathbf{R}_a \mathbf{a} + \int \mathcal{N}(\mathbf{z} | \mathbf{T}_s \mathbf{s} + \mathbf{T}_a \mathbf{a}, \Sigma) (\mathbf{z}^\top \mathbf{V}_{n-1} \mathbf{z} + q_{n-1}) d\mathbf{z} \right). \quad (4.19)$$

Simplifying, we get

$$U_n(\mathbf{s}) = q_{n-1} + \mathbf{s}^\top \mathbf{R}_s \mathbf{s} + \max_{\mathbf{a}} \left(\mathbf{a}^\top \mathbf{R}_a \mathbf{a} + \int \mathcal{N}(\mathbf{z} | \mathbf{T}_s \mathbf{s} + \mathbf{T}_a \mathbf{a}, \Sigma) \mathbf{z}^\top \mathbf{V}_{n-1} \mathbf{z} d\mathbf{z} \right). \quad (4.20)$$

The integral in the equation above evaluates to

$$\text{Tr}(\Sigma \mathbf{V}_{n-1}) + (\mathbf{T}_s \mathbf{s} + \mathbf{T}_a \mathbf{a})^\top \mathbf{V}_{n-1} (\mathbf{T}_s \mathbf{s} + \mathbf{T}_a \mathbf{a}), \quad (4.21)$$

where Tr represents the *trace* of a matrix, which is simply the sum of the main diagonal elements. We now have

$$U_n(\mathbf{s}) = q_{n-1} + \mathbf{s}^\top \mathbf{R}_s \mathbf{s} + \text{Tr}(\Sigma \mathbf{V}_{n-1}) + \max_{\mathbf{a}} \left(\mathbf{a}^\top \mathbf{R}_a \mathbf{a} + (\mathbf{T}_s \mathbf{s} + \mathbf{T}_a \mathbf{a})^\top \mathbf{V}_{n-1} (\mathbf{T}_s \mathbf{s} + \mathbf{T}_a \mathbf{a}) \right). \quad (4.22)$$

We can determine the $\mathbf{a}$ that maximizes the last term in the equation above by computing the derivative with respect to $\mathbf{a}$, setting it to 0, and solving for $\mathbf{a}$:

$$2\mathbf{a}^\top \mathbf{R}_a + 2(\mathbf{T}_s \mathbf{s} + \mathbf{T}_a \mathbf{a})^\top \mathbf{V}_{n-1} \mathbf{T}_a = 0 \quad (4.23)$$

$$\mathbf{a} = -(\mathbf{T}_a^\top \mathbf{V}_{n-1} \mathbf{T}_a + \mathbf{R}_a)^{-1} \mathbf{T}_a^\top \mathbf{V}_{n-1} \mathbf{T}_s \mathbf{s}. \quad (4.24)$$

Substituting Equation (4.24) into Equation (4.22) and simplifying, we get $U_n(s) = s^\top V_n s + q_n$ with

$$V_n = T_s^\top V_{n-1} T_s - T_s^\top V_{n-1} T_a (T_a^\top T_a + R_a)^{-1} T_a^\top V_{n-1} T_s + R_s \quad (4.25)$$

$$q_n = q_{n-1} + \text{Tr}(\Sigma V_{n-1}). \quad (4.26)$$

To compute V_n and q_n for arbitrary n , we first set $V_0 = 0$ and $q_0 = 0$ and iterate using the equations above. Once we know V_{n-1} and q_{n-1} , we can extract the optimal n -step policy

$$\pi_n(s) = -(T_a^\top V_{n-1} T_a + R_a)^{-1} T_a^\top V_{n-1} T_s s. \quad (4.27)$$

Interestingly, $\pi_n(s)$ does not depend on the covariance of the noise Σ , although the optimal cost does depend on the noise. A linear system with quadratic cost that has no noise in the dynamics is known in control theory as a *linear quadratic regulator* and has been well studied.

4.5 Approximate Dynamic Programming

The field of *approximate dynamic programming* is concerned with finding approximately optimal policies for problems with large or continuous spaces. Approximate dynamic programming is an active area of research that shares ideas with reinforcement learning. In reinforcement learning, we try to quickly accrue as much reward as possible without a known model. Many of the algorithms for reinforcement learning (discussed in the next chapter) can be applied directly to approximate dynamic programming. This section focuses on several local and global approximation strategies for efficiently finding value functions and policies for known models.

4.5.1 Local Approximation

Local approximation relies on the assumption that states close to each other have similar values. If we know the value associated with a finite set of states $s_{1:n}$, then we can approximate the value of arbitrary states by using the equation

$$U(s) = \sum_{i=1}^n \lambda_i \beta_i(s) = \lambda^\top \beta(s), \quad (4.28)$$

where $\beta_{1:n}$ are weighting functions, such that $\sum_{i=1}^n \beta_i(s) = 1$. The value λ_i is the value of state s_i . In general, $\beta_i(s)$ should assign greater weight to states that are closer (in some sense) to s_i . A weighting function is often referred to as a *kernel*.

Algorithm 4.4 shows how to compute an approximation of the optimal value function by iteratively updating λ . The loop is continued until convergence. Once an approximate value function is known, an approximately optimal policy can be extracted as follows:

$$\pi(s) \leftarrow \arg \max_a \left(R(s, a) + \gamma \sum_{s'} T(s' | s, a) \lambda^\top \beta(s') \right). \quad (4.29)$$

Algorithm 4.4 Local approximation value iteration

```

1: function LOCALAPPROXIMATIONVALUEITERATION
2:    $\lambda \leftarrow \mathbf{0}$ 
3:   loop
4:     for  $i \leftarrow 1$  to  $n$ 
5:        $u_i \leftarrow \max_a [R(s_i, a) + \gamma \sum_{s'} T(s' | s_i, a) \lambda^\top \beta(s')]$ 
6:      $\lambda \leftarrow \mathbf{u}$ 
7:   return  $\lambda$ 

```

A simple approach to local approximation, called *nearest neighbor*, is to assign all weight to the closest discrete state, resulting in a piecewise constant value function. A smoother approximation can be achieved using *k-nearest neighbor*, where a weight of $1/k$ is assigned to each of the k nearest discrete states of s .

If we define a neighborhood function $N(s)$ that returns a set of states from $s_{1:n}$, then we can use *linear interpolation*. If the state space is one dimensional and $N(s) = \{s_1, s_2\}$, then the interpolated value is given by

$$U(s) = \lambda_1 \underbrace{\left(1 - \frac{s - s_1}{s_2 - s_1} \right)}_{\beta_1(s)} + \lambda_2 \underbrace{\left(1 - \frac{s_2 - s}{s_2 - s_1} \right)}_{\beta_2(s)}. \quad (4.30)$$

The equation above can be generalized to d -dimensional state spaces and is often called *bilinear interpolation* in two dimensions and *multilinear interpolation* in arbitrary dimensions.

If the state space has been discretized using a multidimensional grid and the vertices of the grid correspond to the discrete states, then $N(s)$ could be defined to be the set of vertices of the rectangular cell that encloses s . In d dimensions, there may be as many as 2^d neighbors.

Figure 4.9 shows an example grid-based discretization of a two-dimensional state space. To determine the interpolated value of state s (shown as a black circle in the figure), we look at the discrete states in $N(s) = \{s_{12}, s_{13}, s_{17}, s_{18}\}$ (shown as white circles). Using the neighboring values and weights shown in the figure, we compute the value at

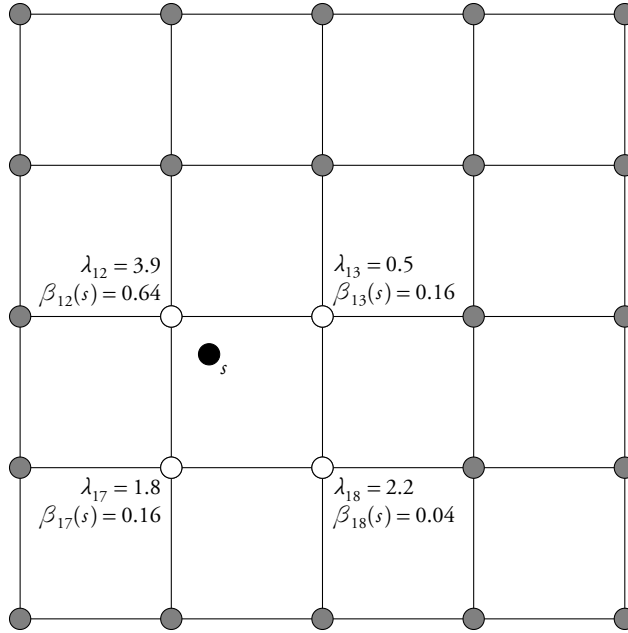


Figure 4.9 Multilinear interpolation in two dimensions.

s as follows:

$$U(s) = \lambda_{12}\beta_{12}(s) + \lambda_{13}\beta_{13}(s) + \lambda_{17}\beta_{17}(s) + \lambda_{18}\beta_{18}(s) \quad (4.31)$$

$$= 3.9 \times 0.64 + 0.5 \times 0.16 + 1.8 \times 0.16 + 2.2 \times 0.04 \quad (4.32)$$

$$= 2.952. \quad (4.33)$$

When the dimensionality of the problem is high, it may be prohibitive to interpolate over the 2^d vertices in the enclosing rectangular cell. An alternative is to use *simplex-based interpolation*. In the simplex method, the rectangular cells are broken into $d!$ multidimensional triangles, called *simplexes*. Instead of interpolating over rectangular cells, we interpolate over a simplex defined by up to $d + 1$ vertices. Hence, interpolating over the simplex scales linearly instead of exponentially with the dimensionality of the state space. However, rectangular interpolation can provide higher quality estimates that can lead to better policies for the same grid resolution. Figure 4.10 shows an example of two-dimensional rectangular and simplex interpolation.

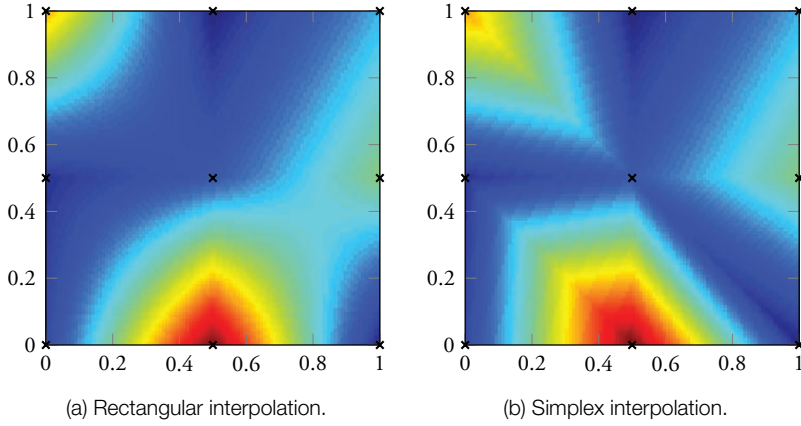


Figure 4.10 Interpolation methods. Data points are indicated by crosses.

4.5.2 Global Approximation

Global approximation uses a fixed set of parameters $\lambda_{1:m}$ to approximate the value function over the entire state space $\mathcal{S}$. One of the most commonly used global approximation methods is based on *linear regression*. We define a set of *basis functions* $\beta_{1:m}$, where $\beta_i : \mathcal{S} \rightarrow \mathbb{R}$. Sometimes these basis functions are called *features*. The approximation of $U(s)$ is a linear combination of the parameters and output of the basis functions:

$$U(s) = \sum_{i=1}^m \lambda_i \beta_i(s) = \lambda^\top \beta(s). \quad (4.34)$$

The approximation above has the same form as Equation (4.28), but the interpretation is different. The parameters $\lambda_{1:m}$ do not correspond to values at discrete states. The basis functions $\beta_{1:m}$ are not necessarily related to a distance measure, and they need not sum to 1.

Algorithm 4.5 shows how to incorporate linear regression into value iteration. The algorithm is nearly identical to Algorithm 4.4, except for Line 6. Instead of simply assigning $\lambda \leftarrow \mathbf{u}$ as done in local approximation, we call $\lambda_{1:m} \leftarrow \text{REGRESS}(\beta, s_{1:n}, u_{1:n})$. The REGRESS function finds the λ that leads to the best approximation of the target values $u_{1:n}$ at points $s_{1:n}$ using the basis function β . A common regression objective is to minimize the *sum-squared error*:

$$\sum_{i=1}^n (\lambda^\top \beta(s_i) - u_i)^2. \quad (4.35)$$

Linear least-squares regression can compute the λ that minimizes the sum-squared error through simple matrix operations. There are a wide variety of other well-studied regression approaches, both linear and non-linear.

Algorithm 4.5 Linear regression value iteration

```

1: function LINEARREGRESSIONVALUEITERATION
2:    $\lambda \leftarrow 0$ 
3:   loop
4:     for  $i \leftarrow 1$  to  $n$ 
5:        $u_i \leftarrow \max_a [R(s_i, a) + \gamma \sum_{s'} T(s' | s_i, a) \lambda^\top \beta(s')]$ 
6:        $\lambda_{1:n} \leftarrow \text{REGRESS}(\beta, s_{1:n}, u_{1:n})$ 
7:   return  $\lambda$ 

```

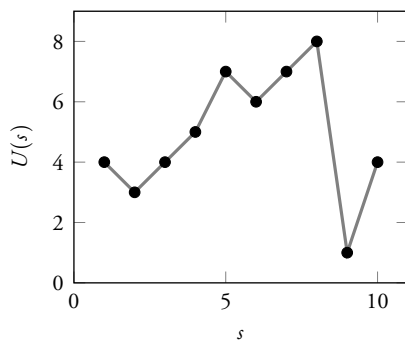
Figure 4.11 compares linear interpolation with linear regression by using different basis functions. For simplicity, the figure assumes a one-dimensional state space, and the states $s_{1:10}$ are evenly spaced. The target values $u_{1:10}$ obtained through dynamic programming are plotted as dots.

Figure 4.11a shows linear interpolation, which produces an approximate value function that matches $u_{1:10}$ exactly at the states $s_{1:10}$. Linear interpolation, of course, requires 10 parameters.

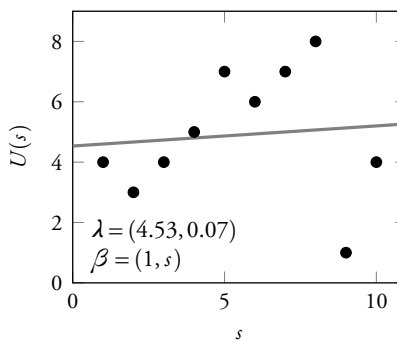
Figure 4.11b shows the result of linear least-squares regression with basis functions $\beta_1(s) = 1$ and $\beta_2(s) = s$. In this case, $\lambda_1 = 4.53$ and $\lambda_2 = 0.07$, meaning that $U(s)$ is approximated as $4.53 + 0.07s$. Although this λ minimizes the sum-squared error given those two basis functions, the plot shows that the resulting approximate value function is not especially accurate.

Figure 4.11c shows the result of adding an additional basis function $\beta_3(s) = s^2$. The approximate value function is now quadratic over the state space. Adding this additional basis function results in new values for λ_1 and λ_2 . The sum-squared error of the quadratic value function at the states $s_{1:n}$ is much smaller than with the linear function.

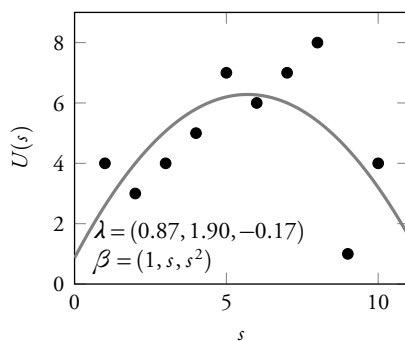
Figure 4.11d adds an additional cubic basis function $\beta_4(s) = s^3$, further improving the approximation. All of the basis functions in this example are polynomials, but we could have easily added other basis functions such as $\sin(s)$ and e^s . Adding additional basis functions can generally improve the ability to match the target values at the known states, but too many basis functions can lead to poor approximations at other states. Principled methods exist for choosing an appropriate set of basis functions for regression.



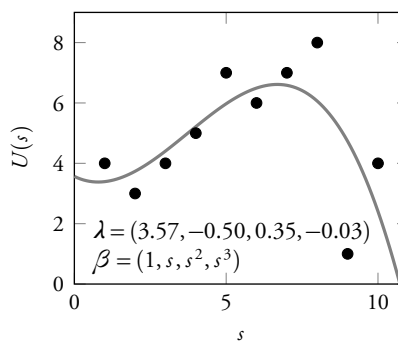
(a) Linear interpolation.



(b) Linear regression (linear basis).



(c) Linear regression (quadratic basis).



(d) Linear regression (cubic basis).

Figure 4.11 Approximations of the value function.

4.6 Online Methods

All the methods presented in this chapter so far involve computing the policy for the entire state space *offline*—that is, prior to execution in the environment. Although factored representations and value function approximation can help scale dynamic programming to higher dimensional state spaces, computing and representing a policy over the full state space can still be intractable. This section discusses *online* methods that restrict computation to states that are reachable from the current state. Because the reachable state space can be orders of magnitude smaller than the full state space, online methods can significantly reduce the amount of storage and computation required to choose optimal (or approximately optimal) actions.

4.6.1 Forward Search

Forward search (Algorithm 4.6) is a simple online action-selection method that looks ahead from some initial state s_0 to some horizon (or depth) d . The forward search function $\text{SELECTACTION}(s, d)$ returns the optimal action a^* and its value v^* . The pseudocode uses $A(s)$ to represent the set of actions available from state s , which may be a subset of the full action space A . The set of possible states that can follow immediately from s after executing action a is denoted $S(s, a)$, which may be a small subset of the full state spaces S .

Algorithm 4.6 Forward search

```

1: function SELECTACTION( $s, d$ )
2:   if  $d = 0$ 
3:     return (NIL, 0)
4:    $(a^*, v^*) \leftarrow (\text{NIL}, -\infty)$ 
5:   for  $a \in A(s)$ 
6:      $v \leftarrow R(s, a)$ 
7:     for  $s' \in S(s, a)$ 
8:        $(a', v') \leftarrow \text{SELECTACTION}(s', d - 1)$ 
9:        $v \leftarrow v + \gamma T(s' \mid s, a)v'$ 
10:    if  $v > v^*$ 
11:       $(a^*, v^*) \leftarrow (a, v)$ 
12:   return  $(a^*, v^*)$ 

```

Algorithm 4.6 iterates over all possible action and next state pairings and calls itself recursively until the desired depth is reached. The call tree has depth d with a worst-case

branching factor of $|S| \times |A|$ and proceeds depth-first. The computational complexity is $O((|S| \times |A|)^d)$.

4.6.2 Branch and Bound Search

Branch and bound search (Algorithm 4.7) is an extension to forward search that uses knowledge of the upper and lower bounds of the value function to prune portions of the search tree. This algorithm assumes that prior knowledge is available that allows us to easily compute a lower bound on the value function $\underline{U}(s)$ and an upper bound on the state-action value function $\overline{U}(s, a)$. The pseudocode is identical to Algorithm 4.6, except for the use of the lower bound in Line 3 and the pruning check in Line 6. The call to `SELECTACTION`(s, d) returns the action to execute and a lower bound on the value function.

The order in which we iterate over the actions in Line 5 is important. In order to prune, the actions must be in descending order of upper bound. In other words, if action a_i is evaluated before a_j , then $\overline{U}(s, a_i) \geq \overline{U}(s, a_j)$. The tighter we are able to make the upper and lower bounds, the more we can prune the search space and decrease computation time. The worst-case computational complexity, however, remains the same as for forward search.

Algorithm 4.7 Branch-and-bound search

```

1: function SELECTACTION( $s, d$ )
2:   if  $d = 0$ 
3:     return (NIL,  $\underline{U}(s)$ )
4:   ( $a^*, v^*$ )  $\leftarrow$  (NIL,  $-\infty$ )
5:   for  $a \in A(s)$ 
6:     if  $\overline{U}(s, a) < v^*$ 
7:       return ( $a^*, v^*$ )
8:      $v \leftarrow R(s, a)$ 
9:     for  $s' \in S(s, a)$ 
10:      ( $a', v'$ )  $\leftarrow$  SELECTACTION( $s', d - 1$ )
11:       $v \leftarrow v + \gamma T(s' | s, a) v'$ 
12:     if  $v > v^*$ 
13:       ( $a^*, v^*$ )  $\leftarrow$  ( $a, v$ )
14:   return ( $a^*, v^*$ )

```

4.6.3 Sparse Sampling

Sampling methods can be used to avoid the worst-case exponential complexity of forward and branch-and-bound search. Although these methods are not guaranteed to produce the optimal action, they can be shown to produce approximately optimal actions most of the time and can work well in practice. One of the simplest approaches is referred to as *sparse sampling* (Algorithm 4.8).

Sparse sampling uses a generative model G to produce samples of the next state s' and reward r . An advantage of using a generative model is that it is often easier to implement code for drawing random samples from a complex, multidimensional distribution rather than explicitly representing probabilities. Line 8 of the algorithm draws $(s', r) \sim G(s, a)$. All of the information about the state transitions and rewards is represented by G ; the state transition probabilities $T(s' | s, a)$ and expected reward function $R(s, a)$ are not used directly.

Algorithm 4.8 Sparse sampling

```

1: function SELECTACTION( $s, d$ )
2:   if  $d = 0$ 
3:     return (NIL, 0)
4:    $(a^*, v^*) \leftarrow (\text{NIL}, -\infty)$ 
5:   for  $a \in A(s)$ 
6:      $v \leftarrow 0$ 
7:     for  $i \leftarrow 1$  to  $n$ 
8:        $(s', r) \sim G(s, a)$ 
9:        $(a', v') \leftarrow \text{SELECTACTION}(s', d - 1)$ 
10:       $v \leftarrow v + (r + \gamma v') / n$ 
11:   if  $v > v^*$ 
12:      $(a^*, v^*) \leftarrow (a, v)$ 
13:   return  $(a^*, v^*)$ 

```

Sparse sampling is similar to forward search, except that it iterates over n samples instead of all the states in $S(s, a)$. Each iteration results in a sample of $r + \gamma v'$, where r comes from the generative model and v' comes from a recursive call to $\text{SELECTACTION}(s', d - 1)$. These samples of $r + \gamma v'$ are averaged together to estimate $Q(s, a)$. The run time complexity $O((n \times |A|)^d)$ is still exponential in the horizon but does not depend on the size of the state space.

4.6.4 Monte Carlo Tree Search

One of the most successful sampling-based online approaches in recent years is *Monte Carlo tree search*. Algorithm 4.9 is the Upper Confidence Bound for Trees (UCT) implementation of Monte Carlo tree search. In contrast with sparse sampling, the complexity of Monte Carlo tree search does not grow exponentially with the horizon. As in sparse sampling, we use a generative model.

Algorithm 4.9 Monte Carlo tree search

```

1: function SELECTACTION( $s, d$ )
2:   loop
3:     SIMULATE( $s, d, \pi_0$ )
4:   return  $\arg \max_a Q(s, a)$ 
5: function SIMULATE( $s, d, \pi_0$ )
6:   if  $d = 0$ 
7:     return 0
8:   if  $s \notin T$ 
9:     for  $a \in A(s)$ 
10:       $(N(s, a), Q(s, a)) \leftarrow (N_0(s, a), Q_0(s, a))$ 
11:     $T = T \cup \{s\}$ 
12:    return ROLLOUT( $s, d, \pi_0$ )
13:   $a \leftarrow \arg \max_{a \in A(s)} \left[ Q(s, a) + c \sqrt{\frac{\log N(s)}{N(s, a)}} \right]$ 
14:   $(s', r) \sim G(s, a)$ 
15:   $q \leftarrow r + \gamma \text{SIMULATE}(s', d - 1, \pi_0)$ 
16:   $N(s, a) \leftarrow N(s, a) + 1$ 
17:   $Q(s, a) \leftarrow Q(s, a) + \frac{q - Q(s, a)}{N(s, a)}$ 
18:  return  $q$ 

```

Algorithm 4.10 Rollout evaluation

```

1: function ROLLOUT( $s, d, \pi_0$ )
2:   if  $d = 0$ 
3:     return 0
4:    $a \sim \pi_0(s)$ 
5:    $(s', r) \sim G(s, a)$ 
6:   return  $r + \gamma \text{ROLLOUT}(s', d - 1, \pi_0)$ 

```

The algorithm involves running many simulations from the current state while updating an estimate of the state-action value function $Q(s, a)$. There are three stages in each simulation:

- *Search.* If the current state in the simulation is in the set T (initially empty), then we enter the search stage. Otherwise we proceed to the expansion stage. During the search stage, we update $Q(s, a)$ for the states and actions visited and tried in our search. We also keep track of the number of times we have taken an action from a state $N(s, a)$. During the search, we execute the action that maximizes

$$Q(s, a) + c \sqrt{\frac{\log N(s)}{N(s, a)}}, \quad (4.36)$$

where $N(s) = \sum_a N(s, a)$ and c is a parameter that controls the amount of exploration in the search (exploration will be covered in depth in the next chapter). The second term is an *exploration bonus* that encourages selecting actions that have not been tried as frequently. This bonus is infinite if $N(s, a) = 0$.

- *Expansion.* Once we have reached a state that is not in the set T , we iterate over all of the actions available from that state and initialize $N(s, a)$ and $Q(s, a)$ with $N_0(s, a)$ and $Q_0(s, a)$, respectively. The functions N_0 and Q_0 can be based on prior expert knowledge of the problem; if none is available, then they can both be initialized to 0. We then add the current state to the set T .
- *Rollout.* After the expansion stage, we simply select actions according to some *rollout* (or default) policy π_0 until the desired depth is reached (Algorithm 4.10). Typically, rollout policies are stochastic, and so the action to execute is sampled $a \sim \pi_0(s)$. The rollout policy does not have to be close to optimal, but it is a way for an expert to bias the search into areas that are promising. The expected value is returned and used in the search to update the value for $Q(s, a)$.

Simulations are run until some stopping criterion is met, often simply a fixed number of iterations. We then execute the action that maximizes $Q(s, a)$. Once that action has been executed, we can rerun the Monte Carlo tree search to select the next action. It is common to carry over the values of $N(s, a)$ and $Q(s, a)$ computed in the previous step.

4.7 Direct Policy Search

The previous sections have presented methods that involve computing or approximating the value function. An alternative is to search the space of policies directly. Although the state space may be high dimensional, making approximation of the value function difficult, the space of possible policies may be relatively low dimensional and can be easier to search directly.

4.7.1 Objective Function

Suppose we have a policy that is parametrized by λ . The probability that the policy selects action a given state s is written $\pi_\lambda(a | s)$. Given an initial state s , we can estimate

$$U^{\pi_\lambda}(s) \approx \frac{1}{n} \sum_{i=1}^n u_i, \quad (4.37)$$

where u_i is the i th rollout of the policy π_λ to some depth.

The objective in direct policy search is to find the parameter λ that maximizes the function

$$V(\lambda) = \sum_s b(s) U^{\pi_\lambda}(s), \quad (4.38)$$

where $b(s)$ is a distribution over the initial state. We can estimate $V(\lambda)$ using Monte Carlo simulation and a generative model G up to depth d as outlined in Algorithm 4.11.

Algorithm 4.11 Monte Carlo policy evaluation

```

1: function MONTECARLOPOLICYEVALUATION( $\lambda, d$ )
2:   for  $i \leftarrow 1$  to  $n$ 
3:      $s \sim b$ 
4:      $u_i \leftarrow \text{ROLLOUT}(s, d, \pi_\lambda)$ 
5:   return  $\frac{1}{n} \sum_{i=1}^n u_i$ 

```

The function $V(\lambda)$, as estimated by Algorithm 4.11, is a *stochastic function*; given the same input λ , it may give different outputs. As the number of samples (determined by n) increases, the variability of the outputs of the function decreases. Many different methods exist for searching the space of policy parameters that maximizes $V(\lambda)$, and we will discuss a few of them.

4.7.2 Local Search Methods

A common stochastic optimization approach is *local search*, also known as *hill climbing* or *gradient ascent*. Local search begins at a single point in the search space and then incrementally moves from neighbor to neighbor in the search space until convergence. The search operates with the assumption that the value of the stochastic function at a point in the search space is an indication of how close that point is to the global optimum. Therefore, local search generally selects the neighbor with the largest value.

Some local search techniques directly estimate the gradient $\nabla_{\lambda} V$ for a particular policy and then step some amount in the direction of steepest ascent. For some policy representations, it is possible to analytically derive the gradient. Other local search techniques evaluate a finite sampling of the neighborhood of the current search point and then move to the neighbor with the greatest value. Local search is susceptible to local optima and plateaus in $V(\lambda)$. Simulated annealing or some of the other methods suggested at the end of Section 2.4.2 can be applied to help find a global optimum.

4.7.3 Cross Entropy Methods

There is another class of policy search methods that maintain a distribution over policies and updates the distribution based on policies that perform well. One approach to updating this distribution is to use the *cross entropy* method. Cross entropy is a concept from information theory and is a measure of the difference between two distributions. If distributions p and q are discrete, then the cross entropy is given by

$$H(p, q) = - \sum_x p(x) \log q(x). \quad (4.39)$$

For continuous distributions, the summation is replaced with an integral. In the context of direct policy search, we are interested in distributions over λ . These distributions are parameterized by θ , which may be multivariate.

The cross entropy method takes as input an initial θ and parameters n and m that determine the number of samples to use. The process consists of two stages that are repeated until convergence or some other stopping criterion is met:

- *Sample.* We draw n samples from $P(\lambda \mid \theta)$ and evaluate their performance using Algorithm 4.11. Sort the samples in decreasing order of performance so that $i < j$ implies that $V(\lambda_i) \geq V(\lambda_j)$.
- *Update.* Use the top m performing samples (often called the *elite* samples) to update θ using cross entropy minimization, which boils down to:

$$\theta \leftarrow \arg \max_{\theta} \sum_{j=1}^m \log P(\lambda_j \mid \theta). \quad (4.40)$$

The new θ happens to correspond to the maximum likelihood estimate based on the m top-performing samples. Further explanation can be found in the references at the end of the chapter.

Algorithm 4.12 Cross entropy policy search

```

1: function CROSSENTROPYPOLICYSEARCH( $\theta$ )
2:   repeat
3:     for  $i \leftarrow 1$  to  $n$ 
4:        $\lambda_i \sim P(\cdot \mid \theta)$ 
5:        $v_i \leftarrow \text{MONTECARLOPOLICYEVALUATION}(\lambda_i)$ 
6:       Sort  $(\lambda_1, \dots, \lambda_n)$  in decreasing order of  $v_i$ 
7:        $\theta \leftarrow \arg \max_{\theta} \sum_{j=1}^m \log P(\lambda_j \mid \theta)$ 
8:   until convergence
9:   return  $\lambda \leftarrow \arg \max P(\lambda \mid \theta)$ 

```

The full process is outlined in Algorithm 4.12. The initial distribution parameter θ , the number of samples n , and the number of elite samples m are the input parameters to this process. To prevent the search from focusing too much on local maxima, the initial θ should provide a diffuse distribution over λ . The choice of the number of samples and elite samples depends on the problem.

To illustrate the cross entropy method, we will assume that the space of policies is one dimensional and that $V(\lambda)$ is as shown in Figure 4.12. We will assume for this example that the parameter $\theta = (\mu, \sigma)$ and $P(\lambda \mid \theta) = \mathcal{N}(\lambda \mid \mu, \sigma^2)$. Initially, we set $\theta = (0, 10)$. To not overwhelm the plots, we use only $n = 20$ samples and $m = 5$ elite samples. Typically, especially for higher dimensional problems, we use one to two orders of magnitude more samples.

Figure 4.12a shows the initial distribution over λ . We draw 20 samples from this distribution. The 5 elite samples are shown with circles, and the 15 other samples are shown with crosses. Those 5 elite samples are used to update θ . Because we are using a Gaussian distribution, the update simply sets the mean to the mean of the elite samples and the standard deviation to the sample standard deviation of the elite samples. The updated distribution is shown in Figure 4.12b. The process is repeated. In the third iteration (Figure 4.12c), the distribution is moved toward the more promising area of the search space. By the fourth iteration (Figure 4.12d), we have found the global optimum.

4.7.4 Evolutionary Methods

Evolutionary search methods derive inspiration from biological evolution. A common approach is to use a *genetic algorithm* that evolves populations of (typically binary) strings representing policies, starting with an initial random population. The strings recombine through genetic crossover and mutation at a rate proportional to their measured fitness

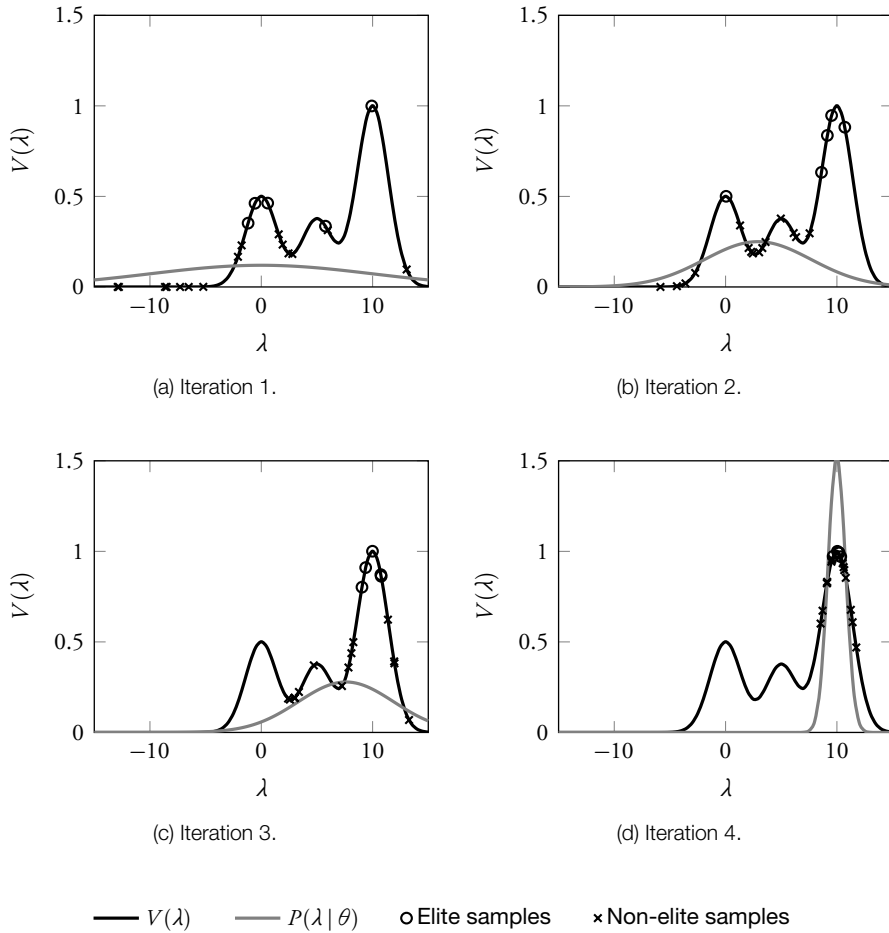


Figure 4.12 Cross entropy method iterations.

to produce a new generation. The process continues until arriving at a satisfactory solution.

A related approach is *genetic programming*, which involves evolving tree structures representing policies. Trees consist of symbols selected from predefined sets of terminals and non-terminals, allowing more flexible policy representations than fixed-length bit strings. Crossover works by swapping subtrees, and mutation works by randomly modifying subtrees.

Genetic algorithms and genetic programming may be combined with other methods, including local search. For example, a genetic algorithm might evolve a satisfactory policy and then use local search to further improve the policy. Such an approach is called *genetic local search* or *memetic algorithms*.

4.8 Summary

- Markov decision processes represent sequential decision-making problems using a transition and reward function.
- Optimal policies can be found using dynamic programming algorithms.
- Continuous problems with linear Gaussian dynamics and quadratic costs can be solved analytically.
- Structured dynamic programming can efficiently solve factored Markov decision processes.
- Problems with large or continuous state spaces can be solved approximately using function approximation.
- Instead of solving for the optimal strategy for the full state space offline, online methods search for the optimal action from the current state.
- In some problems, it can be easier to search the space of policies directly using stochastic optimization methods.

4.9 Further Reading

Much of the pioneering work on sequential decision problems was begun in 1949 by Richard Bellman [1]. Markov decision processes have since become the standard framework for modeling such problems, and there are several books on the subject [2]–[5]. The example grid world problem in Section 4.2.5 comes from *Artificial Intelligence: Foundations of Computational Agents* by Poole and Mackworth [6]. The website associated with the book contains an open-source software demonstration of the grid world example.

Boutilier, Dearden, and Goldszmidt present structured value iteration and policy iteration algorithms for factored MDPs using decision trees [7]. As mentioned in Section 4.3.2, it can be more efficient to use decision diagrams instead of trees [7]–[9].

Approximate linear programming approaches to factored MDPs have been explored by Guestrin et al. [10].

Optimal control in linear systems with quadratic costs has been well studied in the control theory community, and there are many books on the subject [11]–[13]. Section 4.4 presented a special case of the linear-quadratic-Gaussian (LQG) control problem in which the state of the system is known perfectly. Chapter 6 will present the more traditional version of LQG with imperfect state information.

An overview of the field of approximate dynamic programming is provided in *Approximate Dynamic Programming: Solving the Curses of Dimensionality* by Powell [14]. The book *Reinforcement Learning and Dynamic Programming Using Function Approximators* by Busoniu et al. outlines approximation methods and provides source code for cases where the model is known and unknown [15]. Solving problems in which the model is unknown is called reinforcement learning and is discussed in the next chapter. Reinforcement learning is often used in problems with a known model that is too complex or high dimensional to apply exact dynamic programming.

As discussed in Section 4.6, online methods are often appropriate when the state space is high dimensional and adequate computational resources are available to perform planning during execution. Land and Doig originally proposed the branch and bound method for discrete programming problems [16]. This method has been applied to a wide variety of optimization problems. Sparse sampling was developed by Kearns, Mansour, and Ng [17]. Other methods that can be used online include real-time dynamic programming [18] and LAO* [19].

Kocsis and Szepesvári originally introduced the idea of Monte Carlo tree search with the use of the exploration bonus in Algorithm 4.9 [20]. Since that paper’s publication and with the successful application to the game Go, a tremendous amount of work has focused on Monte Carlo tree search methods [21]. One important extension to the algorithm presented in this chapter is the idea of progressive widening of the actions and states considered at each step in the search [22]. Progressive widening allows the algorithm to better handle large or continuous state or action spaces.

Many methods have been proposed for searching the space of policies directly. Examples of local search algorithms using gradient methods include those of Williams [23] and Baxter and Bartlett [24]. The cross entropy method [25], [26] has been applied to a variety of formulations of MDP policy search [27]–[29]. Any stochastic optimization technique can be applied to policy search. The evolutionary methods discussed in Section 4.7.4 date back to the 1950s [30]. Genetic algorithms, in particular, were made popular by the work of Holland [31], with more recent theoretical work done by Schmitt [32], [33]. Genetic programming was introduced by Koza [34].

References

1. S. Dreyfus, “Richard Bellman on the Birth of Dynamic Programming,” *Operations Research*, vol. 50, no. 1, pp. 48–51, 2002.
2. R.E. Bellman, *Dynamic Programming*. Princeton, NJ: Princeton University Press, 1957.
3. D. Bertsekas, *Dynamic Programming and Optimal Control*. Belmont, MA: Athena Scientific, 2007.
4. M.L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Hoboken, NJ: Wiley, 2005.
5. O. Sigaud and O. Buffet, eds., *Markov Decision Processes in Artificial Intelligence*. New York: Wiley, 2010.
6. D.L. Poole and A.K. Mackworth, *Artificial Intelligence: Foundations of Computational Agents*. New York: Cambridge University Press, 2010.
7. C. Boutilier, R. Dearden, and M. Goldszmidt, “Stochastic Dynamic Programming with Factored Representations,” *Artificial Intelligence*, vol. 121, no. 1-2, pp. 49–107, 2000. DOI: 10.1016/S0004-3702(00)00033-3.
8. J. Hoey, R. St-Aubin, A.J. Hu, and C. Boutilier, “SPUDD: Stochastic Planning Using Decision Diagrams,” in *Conference on Uncertainty in Artificial Intelligence (UAI)*, 1999.
9. R. St-Aubin, J. Hoey, and C. Boutilier, “APRICODD: Approximate Policy Construction Using Decision Diagrams,” in *Advances in Neural Information Processing Systems (NIPS)*, 2000.
10. C. Guestrin, D. Koller, R. Parr, and S. Venkataraman, “Efficient Solution Algorithms for Factored MDPs,” *Journal of Artificial Intelligence Research*, vol. 19, pp. 399–468, 2003. DOI: 10.1613/jair.1000.
11. F.L. Lewis, D.L. Vrabie, and V.L. Syrmos, *Optimal Control*, 3rd ed. Hoboken, NJ: Wiley, 2012.
12. R.F. Stengel, *Optimal Control and Estimation*. New York: Dover Publications, 1994.
13. D.E. Kirk, *Optimal Control Theory: An Introduction*. Englewood Cliffs, NJ: Prentice-Hall, 1970.
14. W.B. Powell, *Approximate Dynamic Programming: Solving the Curses of Dimensionality*, 2nd ed. Hoboken, NJ: Wiley, 2011.
15. L. Busoniu, R. Babuska, B. De Schutter, and D. Ernst, *Reinforcement Learning and Dynamic Programming Using Function Approximators*. Boca Raton, FL: CRC Press, 2010.

16. A.H. Land and A.G. Doig, “An Automatic Method of Solving Discrete Programming Problems,” *Econometrica*, vol. 28, no. 3, pp. 497–520, 1960. doi: 10.2307/1910129.
17. M.J. Kearns, Y. Mansour, and A.Y. Ng, “A Sparse Sampling Algorithm for Near-Optimal Planning in Large Markov Decision Processes,” *Machine Learning*, vol. 49, no. 2-3, pp. 193–208, 2002. doi: 10.1023/A:1017932429737.
18. A.G. Barto, S.J. Bradtke, and S.P. Singh, “Learning to Act Using Real-Time Dynamic Programming,” *Artificial Intelligence*, vol. 72, no. 1-2, pp. 81–138, 1995. doi: 10.1016/0004-3702(94)00011-O.
19. E.A. Hansen and S. Zilberstein, “LAO*: A Heuristic Search Algorithm That Finds Solutions with Loops,” *Artificial Intelligence*, vol. 129, no. 1-2, pp. 35–62, 2001. doi: 10.1016/S0004-3702(01)00106-0.
20. L. Kocsis and C. Szepesvári, “Bandit Based Monte-Carlo Planning,” in *European Conference on Machine Learning (ECML)*, 2006.
21. C.B. Browne, E. Powley, D. Whitehouse, S.M. Lucas, P.I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, “A Survey of Monte Carlo Tree Search Methods,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012. doi: 10.1109/TCIAIG.2012.2186810.
22. A. Couëtoux, J.-B. Hoock, N. Sokolovska, O. Teytaud, and N. Bonnard, “Continuous Upper Confidence Trees,” in *Learning and Intelligent Optimization (LION)*, 2011.
23. R.J. Williams, “Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning,” *Machine Learning*, vol. 8, pp. 229–256, 1992. doi: 10.1007/BF00992696.
24. J. Baxter and P.L. Bartlett, “Infinite-Horizon Policy-Gradient Estimation,” *Journal of Artificial Intelligence Research*, vol. 15, pp. 319–350, 2001. doi: 10.1613/jair.806.
25. P.-T. de Boer, D.P. Kroese, S. Mannor, and R.Y. Rubinstein, “A Tutorial on the Cross-Entropy Method,” *Annals of Operations Research*, vol. 134, no. 1, pp. 19–67, 2005. doi: 10.1007/s10479-005-5724-z.
26. R.Y. Rubinstein and D.P. Kroese, *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning*. New York: Springer, 2004.
27. S. Mannor, R.Y. Rubinstein, and Y. Gat, “The Cross Entropy Method for Fast Policy Search,” in *International Conference on Machine Learning (ICML)*, 2003.
28. I. Szita and A. Lörincz, “Learning Tetris Using the Noisy Cross-Entropy Method,” *Neural Computation*, vol. 18, no. 12, pp. 2936–2941, 2006. doi: 10.1162/neco.2006.18.12.2936.

29. —, “Learning to Play Using Low-Complexity Rule-Based Policies: Illustrations Through Ms. Pac-Man,” *Journal of Artificial Intelligence Research*, vol. 30, pp. 659–684, 2007. DOI: 10.1613/jair.2368.
30. D.B. Fogel, ed., *Evolutionary Computation: The Fossil Record*. New York: Wiley-IEEE Press, 1998.
31. J.H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press, 1975.
32. L.M. Schmitt, “Theory of Genetic Algorithms,” *Theoretical Computer Science*, vol. 259, no. 1-2, pp. 1–61, 2001. DOI: 10.1016/S0304-3975(00)00406-0.
33. —, “Theory of Genetic Algorithms II: Models for Genetic Operators over the String-Tensor Representation of Populations and Convergence to Global Optima for Arbitrary Fitness Function Under Scaling,” *Theoretical Computer Science*, vol. 310, no. 1-3, pp. 181–231, 2004. DOI: 10.1016/S0304-3975(03)00393-1.
34. J.R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press, 1992.

5

Model Uncertainty

Mykel J. Kochenderfer

The previous chapter discussed sequential decision problems with a known transition and reward model. In many problems, the dynamics and rewards are not known exactly, and the agent must learn to act through experience. By observing the outcomes of its actions in the form of state transitions and rewards, the agent is to choose actions that maximize its long-term accumulation of rewards. Solving such problems in which there is model uncertainty is the subject of the field of *reinforcement learning* and the focus of this chapter. Several challenges in addressing model uncertainty will be discussed. First, the agent must carefully balance exploration of the environment with the exploitation of that knowledge gained through experience. Second, rewards may be received long after the important decisions have been made, so credit for later rewards must be assigned to earlier decisions. Third, the agent must generalize from limited experience. This chapter will review the theory and some of the key algorithms for addressing these challenges.

5.1 Exploration and Exploitation

Reinforcement learning problems require us to carefully balance *exploration* of the environment with *exploitation* of knowledge obtained through evaluative feedback. If we continuously explore our environment, then we may be able to build a comprehensive model, but we will not be able to accumulate much reward. If we continuously make the decision we believe is best without ever trying a new strategy, then we may miss out on improving our strategy and accumulating more reward. This section introduces the challenges of balancing exploration with exploitation on problems with a single state.

5.1.1 Multi-Armed Bandit Problems

Some of the earliest studies of balancing exploration with exploitation were focused on slot machines—sometimes referred to as *one-armed bandits* because they are often controlled by a single lever and, on average, they take away gamblers' money. Bandit

problems appear in a wide variety of applications, such as the allocation of clinical trials and adaptive network routing. They were originally formulated during World War II and proved exceptionally challenging to solve. According to Peter Whittle, “efforts to solve [bandit problems] so sapped the energies and minds of Allied analysts that the suggestion was made that the problem be dropped over Germany as the ultimate instrument of intellectual sabotage” (see comment following article [1]).

Many different formulations of bandit problems are presented in the literature, but we will focus on a simple one involving a slot machine with n arms. Arm i pays off 1 with probability θ_i and 0 with probability $1 - \theta_i$. There is no deposit to play, but we are limited to h pulls. We can view this problem as an h -step finite horizon Markov decision process (Chapter 4) with a single state, n actions, and an unknown reward function $R(s, a)$.

5.1.2 Bayesian Model Estimation

We can use the beta distribution introduced in Section 2.3.2 to represent our posterior over the win probability θ_i for arm i , and we will use the uniform prior distribution, which corresponds to $\text{Beta}(1, 1)$. We just have to keep track of the number of wins w_i and the number of losses ℓ_i for each arm i . The posterior for θ_i is given by $\text{Beta}(w_i + 1, \ell_i + 1)$. We can then compute the posterior probability of winning:

$$\rho_i = P(\text{win}_i \mid w_i, \ell_i) = \int_0^1 \theta \times \text{Beta}(\theta \mid w_i + 1, \ell_i + 1) d\theta = \frac{w_i + 1}{w_i + \ell_i + 2}. \quad (5.1)$$

For example, suppose we have a two-armed bandit that we have pulled six times. The first arm had 1 win and 0 losses, and the second arm has 4 wins and 1 loss. Assuming a uniform prior, the posterior distribution for θ_1 is given by $\text{Beta}(2, 1)$, and the posterior distribution for θ_2 is given by $\text{Beta}(5, 2)$. The posteriors are plotted in Figure 5.1.

The maximum likelihood estimate for θ_1 is 1 and the maximum likelihood estimate for θ_2 is $4/5$. If we were to choose the next pull solely on the basis of the maximum-likelihood estimate, then we would want to go with the first arm—with a guaranteed win! Of course, just because we have not yet observed a loss with the first arm does not mean that a loss is impossible.

In contrast with the maximum likelihood estimate of the payoff probabilities, the Bayesian posterior shown in Figure 5.1 assigns non-zero probability to probabilities between 0 and 1. The density at 0 for both arms is 0 because at least one win was observed from both arms. There is also zero density at $\theta_2 = 1$ because a loss was observed. Using Equation (5.1), we can compute the payoff probabilities:

$$\rho_1 = 2/3 = 0.67 \quad (5.2)$$

$$\rho_2 = 5/7 = 0.71. \quad (5.3)$$

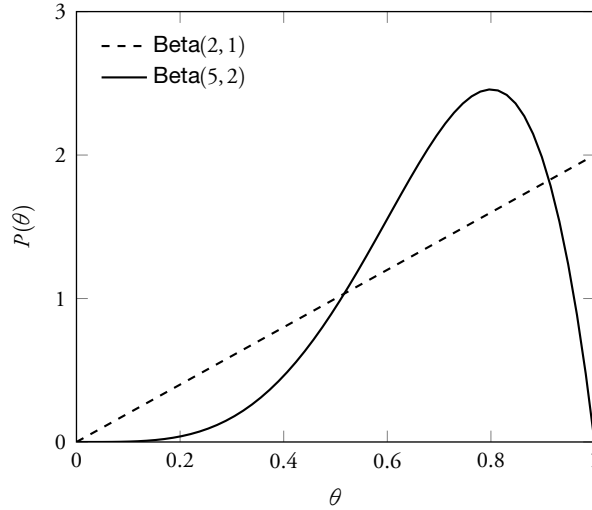


Figure 5.1 Posterior distribution of payoff probabilities.

Hence, if we assume that we only have one pull remaining, it is best to pull the second arm.

5.1.3 Ad Hoc Exploration Strategies

Several different ad hoc exploration strategies have been suggested in the literature. In one of the most common strategies, ϵ -greedy, we choose a random arm with probability ϵ ; otherwise, we choose $\arg \max_i \rho_i$. Larger values of ϵ allow us to more quickly identify the best arm, but more pulls are wasted on suboptimal arms.

Directed exploration strategies involve using information gathered from previous pulls. For example, the *softmax* strategy is to pull arms according to the logit model (introduced in Section 3.3.3), where arm i is selected with probability proportional to $\exp(\lambda \rho_i)$. The precision parameter $\lambda \geq 0$ controls the amount of exploration, with uniform random selection as $\lambda \rightarrow 0$ and greedy selection as $\lambda \rightarrow \infty$. Another approach is to use *interval exploration*, by which we compute the $\alpha\%$ confidence interval for θ_i and choose the arm with the highest upper bound. Larger values for α result in more exploration.

5.1.4 Optimal Exploration Strategies

The counts $w_1, \ell_1, \dots, w_n, \ell_n$ represent a *belief state*, which summarizes our belief about payoffs. As discussed in Section 5.1.2, these $2n$ numbers can be used to represent n

continuous probability distributions over possible values for $\rho_{1:n}$. These belief states can be used as states in an MDP that represents the n -armed bandit problem. We can use dynamic programming to determine an optimal policy π^* , which specifies which arm to pull given the counts.

We use $Q^*(w_{1:n}, \ell_{1:n}, i)$ to represent the expected payoff after pulling arm i and then acting optimally. The optimal utility function and policy are given in terms of Q^* :

$$U^*(w_1, \ell_1, \dots, w_n, \ell_n) = \max_i Q^*(w_1, \ell_1, \dots, w_n, \ell_n, i) \quad (5.4)$$

$$\pi^*(w_1, \ell_1, \dots, w_n, \ell_n) = \arg \max_i Q^*(w_1, \ell_1, \dots, w_n, \ell_n, i). \quad (5.5)$$

We can decompose Q^* into two terms:

$$\begin{aligned} Q^*(w_1, \ell_1, \dots, w_n, \ell_n, i) = & \frac{w_i + 1}{w_i + \ell_i + 2} \left(1 + U^*(\dots, w_i + 1, \ell_i, \dots) \right) \\ & + \left(1 - \frac{w_i + 1}{w_i + \ell_i + 2} \right) U^*(\dots, w_i, \ell_i + 1, \dots). \end{aligned} \quad (5.6)$$

The first term is associated with a win for arm i , and the second term is associated with a loss. The value $(w_i + 1)/(w_i + \ell_i + 2)$ is the posterior probability of a win, which comes from Equation (5.1). The first U^* in the equation above assumes that pulling arm i brought a win, and the second U^* assumes a loss.

Assuming a horizon h , we can compute Q^* for the entire belief space. We start with the belief states with $\sum_i (w_i + \ell_i) = h$. With no pulls left, $U^*(w_1, \ell_1, \dots, w_n, \ell_n) = 0$. We can then work backward to the states where $\sum_i (w_i + \ell_i) = h - 1$ and apply Equation (5.6).

Although this dynamic programming solution is optimal, the number of belief states—and consequently the amount of computation and memory required—is exponential in h . We can formulate an infinite horizon, discounted version of the problem that can be solved efficiently using the *Gittins allocation index*. The allocation index can be stored as a lookup table that specifies a scalar allocation index value given the number of pulls and the number of wins associated with an arm. The arm that has the highest allocation index is the one that should be pulled next.

5.2 Maximum Likelihood Model-Based Methods

A variety of reinforcement learning methods have been proposed for addressing problems with multiple states. Solving problems with multiple states is more challenging than bandit problems because we need to plan to visit states to determine their value. One approach to reinforcement learning involves estimating the transition and reward models directly from experience. We keep track of the counts of transitions $N(s, a, s')$ and the

sum of rewards $\rho(s, a)$. The maximum likelihood estimates of the transition and reward models are as follows:

$$N(s, a) = \sum_{s'} N(s, a, s') \quad (5.7)$$

$$T(s' | s, a) = N(s, a, s') / N(s, a) \quad (5.8)$$

$$R(s, a) = \rho(s, a) / N(s, a). \quad (5.9)$$

If we have prior knowledge about the transition probabilities or the rewards, then we can initialize $N(s, a, s')$ and $\rho(s, a)$ to values other than 0.

We can solve the MDP assuming the estimated models are correct. Of course, we have to incorporate some exploration strategy, such as those mentioned in Section 5.1.3, in order to ensure that we converge to an optimal strategy. The basic structure of maximum likelihood model-based reinforcement learning is outlined in Algorithm 5.1.

Algorithm 5.1 Maximum likelihood model-based reinforcement learning

```

1: function MAXIMUMLIKELIHOODMODELBASEDREINFORCEMENTLEARNING
2:    $t \leftarrow 0$ 
3:    $s_0 \leftarrow$  initial state
4:   Initialize  $N$ ,  $\rho$ , and  $Q$ 
5:   loop
6:     Choose action  $a_t$  based on some exploration strategy
7:     Observe new state  $s_{t+1}$  and reward  $r_t$ 
8:      $N(s_t, a_t, s_{t+1}) \leftarrow N(s_t, a_t, s_{t+1}) + 1$ 
9:      $\rho(s_t, a_t) \leftarrow \rho(s_t, a_t) + r_t$ 
10:    Update  $Q$  based on revised estimate of  $T$  and  $R$ 
11:     $t \leftarrow t + 1$ 
```

5.2.1 Randomized Updates

Although we can use any dynamic programming algorithm to update Q in Line 10 of Algorithm 5.1, the computational expense is often not necessary. One algorithm that avoids solving the entire MDP at each time step is *Dyna*. Dyna performs the following update at the current state:

$$Q(s, a) \leftarrow R(s, a) + \gamma \sum_{s'} T(s' | s, a) \max_{a'} Q(s', a'). \quad (5.10)$$

Here, R and T are the estimated reward and transition functions. We then perform some number of additional updates of Q for random states and actions depending on how much time is available between decisions. Following the updates, we use Q to choose which action to execute—perhaps using softmax or one of the other exploration strategies.

5.2.2 Prioritized Updates

An approach known as *prioritized sweeping* uses a priority queue to help identify which states require updating U the most (Algorithm 5.2). If we transition from s to s' , then we update $U(s)$ based on our updated transition and reward models. We then iterate over the predecessor set, $\text{pred}(s) = \{(s', a') \mid T(s \mid s', a') > 0\}$, the set of all state-action pairs leading immediately to state s . The priority of s' is increased to $T(s \mid s', a') \times |U(s) - u|$, where u was the value of $U(s)$ prior to the update. Hence, the larger the change in $U(s)$, the higher the priority of the states leading to s . The process of updating the highest priority state in the queue continues for some fixed number of iterations or until the queue becomes empty.

Algorithm 5.2 Prioritized sweeping

```

1: function PRIORITIZEDSWEEPING( $s$ )
2:   Increase the priority of  $s$  to  $\infty$ 
3:   while priority queue is not empty
4:      $s \leftarrow$  highest priority state
5:     UPDATE( $s$ )
6: function UPDATE( $s$ )
7:    $u \leftarrow U(s)$ 
8:    $U(s) \leftarrow \max_a [R(s, a) + \gamma \sum_{s'} T(s' \mid s, a) U(s')]$ 
9:   for  $(s', a') \in \text{pred}(s)$ 
10:     $p \leftarrow T(s \mid s', a') \times |U(s) - u|$ 
11:    Increase priority of  $s'$  to  $p$ 

```

5.3 Bayesian Model-Based Methods

The previous section used maximum likelihood estimates of the transition probabilities and rewards and then relied on a heuristic exploration strategy to converge on an optimal strategy in the limit. Bayesian methods, in contrast, allow us to optimally balance exploration with exploitation without having to rely on heuristics. This section describes a generalization of the formulation for multi-armed bandit problems (discussed in Section 5.1.4) applied to general MDPs.

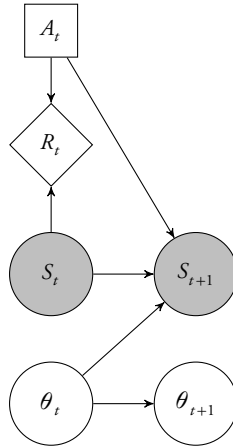


Figure 5.2 Markov decision process with model uncertainty.

5.3.1 Problem Structure

In Bayesian reinforcement learning, we specify a prior distribution over all the model parameters θ . These model parameters may include the parameters governing the distribution over immediate rewards, but we will focus on the parameters governing the state transition probabilities. If $\mathcal{S}$ represents the state space and $\mathcal{A}$ represents the action space, then the parameter vector θ consists of $|\mathcal{S}|^2|\mathcal{A}|$ components representing every possible transition probability. The component of θ that governs the transition probability $T(s' | s, a)$ is denoted $\theta_{(s,a,s')}$.

The structure of the problem can be represented using the dynamic decision network shown in Figure 5.2, which is an extension of the network shown in Figure 4.1b with the model parameters made explicit. As indicated by the shaded nodes, the states are observed, but the model parameters are not. We generally assume that the model parameters are time invariant, and so $\theta_{t+1} = \theta_t$. However, our belief about θ evolves with time as we transition to new states.

5.3.2 Beliefs over Model Parameters

We want to represent a prior belief over θ , and a natural way to do this for discrete state spaces is with a product of Dirichlet distributions. Each Dirichlet would represent a distribution over the next state given the current state s and action a . If $\theta_{(s,a)}$ is an $|\mathcal{S}|$ -element vector representing the distribution over the next state, then the prior distribution is given by

$$\text{Dir}(\theta_{(s,a)} | \alpha_{(s,a)}). \quad (5.11)$$

The Dirichlet distribution above is governed by the $|\mathcal{S}|$ parameters in $\alpha_{(s,a)}$. It is common to use a uniform prior with all the components of $\alpha_{(s,a)}$ set to 1, but if we have prior knowledge about the dynamics, then we can set these parameters differently, as discussed in Section 2.3.2.

The prior distribution over θ is given by the product

$$b_0(\theta) = \prod_s \prod_a \text{Dir}(\theta_{(s,a)} \mid \alpha_{(s,a)}). \quad (5.12)$$

The factorization shown above is often used for small discrete state spaces, but other lower dimensional parametric representations may be desirable.

The posterior distribution over θ after t steps is denoted b_t . Suppose that during the first t steps, we observe $m_{(s,a,s')}$ transitions from s to s' by action a . We compute the posterior using Bayes' rule. If $\mathbf{m}_{(s,a)}$ represents a vector of transition counts, then the posterior is given by

$$b_t(\theta) = \prod_s \prod_a \text{Dir}(\theta_{(s,a)} \mid \alpha_{(s,a)} + \mathbf{m}_{(s,a)}). \quad (5.13)$$

5.3.3 Bayes-Adaptive Markov Decision Processes

We can formulate the problem of acting optimally in an MDP with an *unknown* model as a higher dimensional MDP with a *known* model. This higher dimensional MDP is known as a *Bayes-adaptive Markov decision process*, which is related to the partially observable Markov decision process discussed in the next chapter.

The state space in a Bayes-adaptive MDP is the Cartesian product $\mathcal{S} \times \mathcal{B}$, where $\mathcal{B}$ is the space of all possible beliefs over the model parameters θ . Although $\mathcal{S}$ is discrete, $\mathcal{B}$ is often a high-dimensional continuous space. A state in a Bayes-adaptive MDP is written as a pair (s, b) consisting of the state s of the base MDP and the belief state b . The action space and reward function are exactly the same as for the base MDP.

The transition function in a Bayes-adaptive MDP is $T(s', b' \mid s, b, a)$, which is the probability of transitioning to some new state s' with a new belief state b' , given that you start in state s with belief b and execute action a . The new belief state b' is a deterministic function of s, b, a , and s' as computed by Bayes' rule in Section 5.3.2. Let us denote this deterministic function τ so that $b' = \tau(s, b, a, s')$. The Bayes-adaptive MDP transition function can be decomposed as follows:

$$T(s', b' \mid s, b, a) = \delta_{\tau(s,b,a,s')}(b') P(s' \mid s, b, a), \quad (5.14)$$

where $\delta_x(y)$ is the *Kronecker delta* function such that

$$\delta_x(y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases}. \quad (5.15)$$

Computing $P(s' | s, b, a)$ requires integration:

$$P(s' | s, b, a) = \int_{\theta} b(\theta) P(s' | s, \theta, a) d\theta = \int_{\theta} b(\theta) \theta_{(s,a,s')} d\theta. \quad (5.16)$$

Similar to Equation (5.1), the integral above can be evaluated analytically.

5.3.4 Solution Methods

We can generalize the Bellman equation from Section 4.2.4 for MDPs with a known model to the case in which the model is unknown:

$$U^*(s, b) = \max_a \left(R(s, a) + \gamma \sum_{s'} P(s' | s, b, a) U^*(s', \tau(s, b, a, s')) \right). \quad (5.17)$$

Unfortunately, we cannot simply use the policy iteration and value iteration algorithms directly as presented in Chapter 4 because b is continuous. However, we can use the approximations in Section 4.5 as well as the online methods in Section 4.6. Methods that better leverage the structure of the Bayes-adaptive MDP will be presented in the next chapter.

An alternative to solving for the optimal value function over the belief space is to use a technique known as *Thompson sampling*. The idea here is to draw a sample θ from the current belief b_t and then assume θ is the true model. We use dynamic programming to solve for the best action. At the next time step, we update our belief, draw a new sample, and resolve the MDP. The advantage of this approach is that we do not have to decide on heuristic exploration parameters. However, Thompson sampling has been shown to over-explore, and resolving the MDP at every step can be expensive.

5.4 Model-Free Methods

In contrast to model-based methods, model-free reinforcement learning does not require building explicit representations of the transition and reward models. Avoiding explicit representations is attractive, especially when the problem is high dimensional.

5.4.1 Incremental Estimation

Many model-free methods involve incremental estimation of the expected discounted return from the various states in the problem. Suppose we have a random variable X and want to estimate the mean from a set of samples $x_{1:n}$. After n samples, we have the estimate:

$$\hat{x}_n = \frac{1}{n} \sum_{i=1}^n x_i. \quad (5.18)$$

We can show that

$$\hat{x}_n = \hat{x}_{n-1} + \frac{1}{n}(x_n - \hat{x}_{n-1}) \quad (5.19)$$

$$= \hat{x}_{n-1} + \alpha(n)(x_n - \hat{x}_{n-1}). \quad (5.20)$$

The function $\alpha(n)$ is referred to as the *learning rate*. The learning rate can be a function other than $1/n$; there are rather loose conditions on the learning rate to ensure convergence to the mean. If the learning rate is constant, which is common in reinforcement learning applications, then the weights of older samples decay exponentially at the rate $(1 - \alpha)$. With a constant learning rate, we can update our estimate after observing x using the following rule:

$$\hat{x} \leftarrow \hat{x} + \alpha(x - \hat{x}). \quad (5.21)$$

The update rule above will appear again in later sections and is related to stochastic gradient descent. The magnitude of the update is proportional to the difference in the sample and the previous estimate. The difference between the sample and previous estimate is called the *temporal difference error*.

5.4.2 Q-Learning

One of the most popular model-free reinforcement learning algorithms is *Q-learning*. The idea is to apply incremental estimation to the Bellman equation

$$Q(s, a) = R(s, a) + \gamma \sum_{s'} T(s' | s, a) U(s') \quad (5.22)$$

$$= R(s, a) + \gamma \sum_{s'} T(s' | s, a) \max_{a'} Q(s', a'). \quad (5.23)$$

Instead of using T and R , we use the observed next state s' and reward r to obtain the following incremental update rule:

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a)). \quad (5.24)$$

Q-learning is outlined in Algorithm 5.3. As with the model-based methods, some exploration strategy is required to ensure that Q converges to the optimal state-action value function. We can initialize Q to values other than 0 to encode any prior knowledge we may have about the environment.

Algorithm 5.3 Q-learning

```

1: function QLEARNING
2:    $t \leftarrow 0$ 
3:    $s_0 \leftarrow$  initial state
4:   Initialize  $Q$ 
5:   loop
6:     Choose action  $a_t$  based on  $Q$  and some exploration strategy
7:     Observe new state  $s_{t+1}$  and reward  $r_t$ 
8:      $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$ 
9:      $t \leftarrow t + 1$ 

```

5.4.3 Sarsa

An alternative to Q-learning is *Sarsa*, which derives its name from the fact that it uses $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$ to update the Q function at each step. It uses the actual action taken to update Q instead of maximizing over all possible actions as done in Q-learning. The Sarsa algorithm is identical to Algorithm 5.3 except that Line 8 is replaced with

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)). \quad (5.25)$$

With a suitable exploration strategy, a_{t+1} will converge to the $\arg \max_a Q(s_{t+1}, a)$ that is used to update the Q function in the Q-learning algorithm. Although Q-learning and Sarsa both converge to the optimal strategy, the speed of the convergence depends on the application.

5.4.4 Eligibility Traces

One of the disadvantages of Q-learning and Sarsa is that learning can be very slow. For example, suppose the environment has a single goal state that provides a large reward. The reward is zero at all other states. After some amount of random exploration in the environment, we reach the goal state. Regardless of whether we use Q-learning or Sarsa, we only update the state-action value of the state immediately preceding the goal state. The values at all other states leading up to the goal remain at zero. Much more exploration is required to slowly propagate non-zero values to the remainder of the state space.

Q-learning and Sarsa can be modified to assign credit to achieving the goal to past states and actions using *eligibility traces*. The reward associated with reaching the goal is propagated backward to the states and actions leading up to the goal. The credit is decayed exponentially, so states closer to the goal are assigned larger state-action values. It is common to use λ as the exponential decay parameter, and so the versions of Q-learning and Sarsa with eligibility traces are often called $Q(\lambda)$ and $Sarsa(\lambda)$.

Algorithm 5.4 shows a version of Sarsa(λ). We keep track of an exponentially decaying visit count $N(s, a)$ for all the state-action pairs. When we take action a_t in state s_t , $N(s_t, a_t)$ is incremented by 1. We then update $Q(s, a)$ by adding $\alpha \delta N(s, a)$ at every state s and for every action a , where

$$\delta = r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t). \quad (5.26)$$

After performing the updates, we decay $N(s, a) \leftarrow \gamma \lambda N(s, a)$. Although the impact of eligibility traces is especially pronounced in environments with sparse reward, the algorithm can speed learning in general environments where reward is more evenly distributed.

Algorithm 5.4 Sarsa(λ)-learning

```

1: function SARSA(LAMBDA)LEARNING( $\lambda$ )
2:   Initialize  $Q$  and  $N$ 
3:    $t \leftarrow 0$ 
4:    $s_0, a_0 \leftarrow$  initial state and action
5:   loop
6:     Observe reward  $r_t$  and new state  $s_{t+1}$ 
7:     Choose action  $a_{t+1}$  based on some exploration strategy
8:      $N(s_t, a_t) \leftarrow N(s_t, a_t) + 1$ 
9:      $\delta \leftarrow r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)$ 
10:    for  $s \in S$ 
11:      for  $a \in A$ 
12:         $Q(s, a) \leftarrow Q(s, a) + \alpha \delta N(s, a)$ 
13:         $N(s, a) \leftarrow \gamma \lambda N(s, a)$ 
14:     $t \leftarrow t + 1$ 

```

5.5 Generalization

Up to this point in this chapter, we have assumed that the state-action value function can be represented as a table, which is only useful for small discrete problems. The problem with larger state spaces is not just the size of the state-action table but also the amount of experience required to accurately estimate the values. The agent must *generalize* from limited experience to states that have not yet been visited. Many different approaches have been explored, many related to techniques for approximate dynamic programming Section 4.5.

5.5.1 Local Approximation

The assumption in local approximation methods is that states that are close together are likely to have similar state-action values. A common technique is to store estimates of $Q(s, a)$ at a limited number of states in set S and actions in set A . We denote the vector containing these estimates as θ , which has $|S| \times |A|$ elements. To denote the component associated with state s and action a , we use $\theta_{s,a}$. If we use a weighting function such that $\sum_{s'} \beta(s, s') = 1$ for all s , then we can approximate the state action value at arbitrary states as

$$Q(s, a) = \sum_{s'} \theta_{s',a} \beta(s, s'). \quad (5.27)$$

We can define a vectorized version of the weighting function as follows:

$$\beta(s) = (\beta(s, s_1), \dots, \beta(s, s_{|S|})), \quad (5.28)$$

where $s_1, \dots, s_{|S|}$ are the states in S . We can also define a two-argument version of the function β that takes as input both a state and an action and returns a vector with $|S| \times |A|$ elements. The vector $\beta(s, a)$ is identical to $\beta(s)$, except that the elements associated with actions other than a are set to 0. This notation allows us to rewrite Equation (5.27) as follows:

$$Q(s, a) = \theta^\top \beta(s, a). \quad (5.29)$$

The linear approximation of Equation (5.29) can be easily integrated into Q-learning. The state-action value estimates represented by θ are updated as follows based on observing a state transition from s_t to s_{t+1} by action a_t with reward r_t :

$$\theta \leftarrow \theta + \alpha(r_t + \gamma \max_a \theta^\top \beta(s_{t+1}, a) - \theta^\top \beta(s_t, a_t)) \beta(s_t, a_t). \quad (5.30)$$

The update rule above comes from substituting Equation (5.29) directly into the standard Q-learning update rule and multiplying the last term by $\beta(s_t, a_t)$ to provide greater updates at states that are closer to s_t .

Algorithm 5.5 shows this linear approximation Q-learning method. If we have prior knowledge about the state-action values, then we can initialize θ appropriately. This linear approximation method can easily be extended to other reinforcement learning methods, such as Sarsa.

The algorithm presented above assumes that the points in S remain fixed. However, for some problems, it may be beneficial to adjust the locations of the points in S to produce a better approximation. The locations can be adjusted based on the temporal difference error using a representation such as a *self-organizing map* (see references in Section 5.7). Various criteria have been explored for identifying when it is appropriate to add new points to S , such as when a new state has been observed that is outside some

Algorithm 5.5 Linear approximation Q-learning

```

1: function LINEARAPPROXIMATIONQLearning
2:    $t \leftarrow 0$ 
3:    $s_0 \leftarrow$  initial state
4:   Initialize  $\theta$ 
5:   loop
6:     Choose action  $a_t$  based on  $\theta_a^\top \beta(s_t)$  and some exploration strategy
7:     Observe new state  $s_{t+1}$  and reward  $r_t$ 
8:      $\theta \leftarrow \theta + \alpha(r_t + \gamma \max_a \theta^\top \beta(s_{t+1}, a) - \theta^\top \beta(s_t, a_t)) \beta(s_t, a_t)$ 
9:      $t \leftarrow t + 1$ 

```

threshold distance of the states in S . Although memory can become an issue, some methods simply store all observed states.

5.5.2 Global Approximation

Global approximation methods do not rely on a notion of distance. One such approximation method is a *perceptron*. Perceptrons have been widely used since at least the 1950s to mimic individual neurons for various learning tasks. A perceptron has a set of input nodes $x_{1:m}$, a set of weights $\theta_{1:m}$, and an output node q . The value of the output node is determined as follows:

$$q = \sum_{i=1}^m \theta_i x_i = \theta^\top \mathbf{x}. \quad (5.31)$$

The structure of a perceptron is shown in Figure 5.3a.

In *perceptron Q-learning*, we have a set of n perceptrons, one for each available action. The input is based on the state, and the output is the state-action value. We define a set of basis functions $\beta_1, \dots, \beta_m$ over the state space, similar to the weighting functions in Section 5.5.1. The inputs to the perceptrons are $\beta_1(s), \dots, \beta_m(s)$. If θ_a are the m weights associated with the perceptron for action a , then we have

$$Q(s, a) = \theta_a^\top \beta(s). \quad (5.32)$$

We can define a two-argument version of β as done in Section 5.5.1 and define θ to contain all the weights of all the perceptrons so that we can write

$$Q(s, a) = \theta^\top \beta(s, a). \quad (5.33)$$

Q-learning with a perceptron-based approximation follows Algorithm 5.5 exactly, but θ represents perceptron weights instead of value estimates and β represents basis functions instead of distance measures.

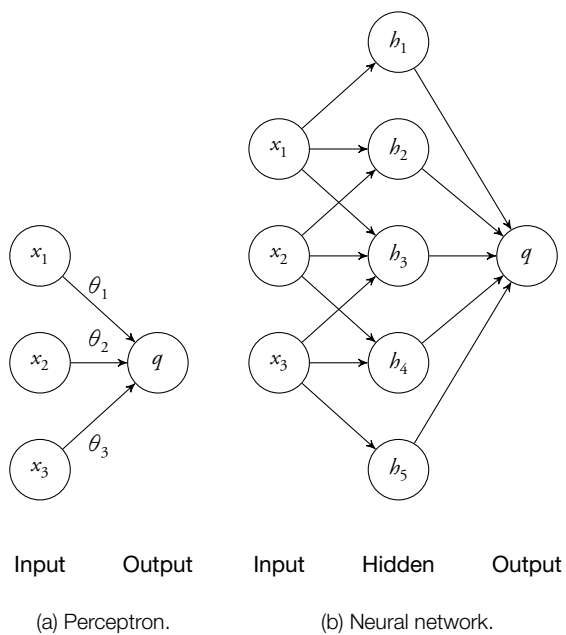


Figure 5.3 Approximation structures.

Perceptrons only represent linear functions, but *neural networks* can represent non-linear functions. A neural network is a network of perceptrons. They are organized into an input layer, a hidden layer, and an output layer as shown in Figure 5.3b; the weights are omitted from the diagram. Adding hidden nodes generally increases the complexity of the state-action function the network can represent.

We can use an algorithm known as *backpropagation* to adjust the weights in the neural network to reduce the temporal difference error. The idea is to first adjust the weights associated with the edges leading from the hidden nodes to the output node in much the same way it is done for perceptron learning. We then compute the error associated with the hidden nodes and adjust the weights from the input nodes to the hidden nodes appropriately. Although convergence when using this form of function approximation is not guaranteed, it can result in satisfactory performance in a variety of domains.

5.5.3 Abstraction Methods

Abstraction methods involve partitioning the state space into discrete regions and estimating the state-action values for each of these regions. Abstraction methods tend to use model-based learning, which often results in faster convergence than model-free learning. Abstraction methods often use decision trees to partition the state space. Associated with the internal nodes of the tree are various tests along the various dimensions of the state space. The leaf nodes correspond to regions.

There are a variety of different abstraction methods, but one method is to start with a single region represented by a decision tree with a single node and then apply a series of acting, modeling, and planning phases. In the acting phase, we select an action based on the state-action value of the region associated with the current state s . After observing the transition from state s to s' by action a with reward r , the experience tuple (s, a, s', r) is stored in the leaf node associated with s .

In the modeling phase, we decide whether to split nodes. For each of the experience tuples at all of the leaf nodes, we compute

$$q(s, a) = r + \gamma U(s'), \quad (5.34)$$

where r is the observed reward and $U(s')$ is the value of the leaf node associated with the next state s' . We want to split a leaf node when the values of the experience tuples come from different distributions. One approach is to choose the split that minimizes the variance of the experience tuples at the resulting leaves. We stop splitting when some stopping criterion is met, such as when the variance at the leaf nodes is below some threshold.

In the planning phase, we use the experience tuples at the leaf nodes to estimate the state transition model and reward model. We then solve the resulting MDP using dynamic programming. The modeling and planning procedures require much more computation than what is required in the other generalization methods, but this approach can find better policies without as much interaction with the environment.

5.6 Summary

- Reinforcement learning is a computational approach to learning intelligent behavior from experience.
- Exploration must be carefully balanced with exploitation.
- In general, solving the exploration problem optimally is not feasible, but there are several Bayesian and heuristic approximation methods that can work well.
- It is important to determine how much actions in the past are responsible for later rewards.
- Model-based reinforcement learning involves building a model from experience and using this model to generate a plan.
- Model-free reinforcement learning involves directly estimating the values of states and actions without the use of transition and reward models.
- We must generalize from observations of rewards and state transitions because our interaction with the world is limited.
- Generalization can be done in a variety of ways, such as local and global approximations of the value function and state abstraction.

5.7 Further Reading

The classic book by Sutton and Barto, titled *Reinforcement Learning: An Introduction*, is the standard introductory text on classical reinforcement learning and provides a historical overview of the emergence of the field [2]. The volume edited by Wiering and Otterlo provides an up-to-date survey of much of the research that occurred since the publication of the Sutton and Barto book [3]. Kovacs and Egginton survey software for reinforcement learning [4].

Multi-armed bandit problems and their many variations have received considerable attention over the years [5]. Gittins developed the concept of an allocation index for solving multi-armed bandit problems [1]. Recent work has focused on improving the efficiency of computing allocation indices [6], [7].

Model-based reinforcement learning can be grouped into non-Bayesian and Bayesian methods [8]. The non-Bayesian methods generally rely on maximum likelihood estimation as discussed in Section 5.2. The Dyna approach was introduced by Sutton [9]. Prioritized sweeping was introduced by Moore and Atkeson [10].

Bayesian model-based methods have started to receive attention more recently [11]. Duff discusses the formulation of model-based reinforcement learning as a Bayes-adaptive Markov decision process [12]. In general, solving such a belief-state formulation exactly is intractable. Strens applies the concept of Thompson sampling [14] to model-based reinforcement learning [13]. Variants of the online planning algorithms presented in the previous chapter have been extended to Bayesian model-based reinforcement learning, including sparse sampling [15] and Monte Carlo tree search [16], [17].

Model-free reinforcement learning algorithms are often used for situations in which it is not feasible to build an explicit representation of the transition and reward models. Q-learning and Sarsa are two commonly used model-free techniques. Eligibility traces were proposed in the context of temporal difference learning by Sutton [18], and they were extended to Sarsa(λ) [19] and Q(λ) [20], [21].

Much of the ongoing work in the field of reinforcement learning is concerned with generalizing from limited experience. The recent book *Reinforcement Learning and Dynamic Programming Using Function Approximators* by Busoniu et al. surveys a variety of different local and global function approximation methods [22]. Several different abstraction methods have been proposed over the years [23]–[26].

Although not discussed in this chapter, there has been some work on Bayesian approaches to model-free reinforcement learning. One approach is to maintain a distribution over state-action values [27], [28]. There are also Bayesian policy gradient methods that have been used with some success [29]. Multiagent reinforcement learning was also not discussed in this chapter, but Busoniu, Babuska, and De Schutter survey recent research in the area [30].

References

1. J.C. Gittins, “Bandit Processes and Dynamic Allocation Indices,” *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 41, no. 2, pp. 148–177, 1979.
2. R.S. Sutton and A.G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.
3. M. Wiering and M. van Otterlo, eds., *Reinforcement Learning: State of the Art*. New York: Springer, 2012.
4. T. Kovacs and R. Egginton, “On the Analysis and Design of Software for Reinforcement Learning, with a Survey of Existing Systems,” *Machine Learning*, vol. 84, no. 1-2, pp. 7–49, 2011. doi: 10.1007/s10994-011-5237-8.
5. J. Gittins, K. Glazebrook, and R. Weber, *Multi-Armed Bandit Allocation Indices*, 2nd ed. Hoboken, NJ: Wiley, 2011.

6. J. Niño-Mora, “A $(2/3)$ Fast-Pivoting Algorithm for the Gittins Index and Optimal Stopping of a Markov Chain,” *INFORMS Journal on Computing*, vol. 19, no. 4, pp. 596–606, 2007. doi: 10.1287/ijoc.1060.0206.
7. I.M. Sonin, “A Generalized Gittins Index for a Markov Chain and Its Recursive Calculation,” *Statistics and Probability Letters*, vol. 78, no. 12, pp. 1526–1533, 2008. doi: 10.1016/j.spl.2008.01.049.
8. P.R. Kumar, “A Survey of Some Results in Stochastic Adaptive Control,” *SIAM Journal on Control and Optimization*, vol. 23, no. 3, pp. 329–380, 1985. doi: 10.1137/0323023.
9. R.S. Sutton, “Dyna, an Integrated Architecture for Learning, Planning, and Reacting,” *SIGART Bulletin*, vol. 2, no. 4, pp. 160–163, 1991. doi: 10.1145/122344.122377.
10. A.W. Moore and C.G. Atkeson, “Prioritized Sweeping: Reinforcement Learning With Less Data and Less Time,” *Machine Learning*, vol. 13, no. 1, pp. 103–130, 1993. doi: 10.1007/BF00993104.
11. P. Poupart, N.A. Vlassis, J. Hoey, and K. Regan, “An Analytic Solution to Discrete Bayesian Reinforcement Learning,” in *International Conference on Machine Learning (ICML)*, 2006.
12. M.O. Duff, “Optimal Learning: Computational Procedures for Bayes-Adaptive Markov Decision Processes,” Ph.D. dissertation, University of Massachusetts at Amherst, 2002.
13. M.J.A. Strens, “A Bayesian Framework for Reinforcement Learning,” in *International Conference on Machine Learning (ICML)*, 2000.
14. W.R. Thompson, “On the Likelihood That One Unknown Probability Exceeds Another in View of the Evidence of Two Samples,” *Biometrika*, vol. 25, no. 3/4, pp. 285–294, 1933. doi: 10.2307/2332286.
15. T. Wang, D.J. Lizotte, M.H. Bowling, and D. Schuurmans, “Bayesian Sparse Sampling for On-Line Reward Optimization,” in *International Conference on Machine Learning (ICML)*, 2005.
16. J. Asmuth and M.L. Littman, “Learning Is Planning: Near Bayes-Optimal Reinforcement Learning via Monte-Carlo Tree Search,” in *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2011.
17. A. Guez, D. Silver, and P. Dayan, “Scalable and Efficient Bayes-Adaptive Reinforcement Learning Based on Monte-Carlo Tree Search,” *Journal of Artificial Intelligence Research*, vol. 48, pp. 841–883, 2013. doi: 10.1613/jair.4117.
18. R. Sutton, “Learning to Predict by the Methods of Temporal Differences,” *Machine Learning*, vol. 3, no. 1, pp. 9–44, 1988. doi: 10.1007/BF00115009.

19. G.A. Rummery, “Problem Solving with Reinforcement Learning,” Ph.D. dissertation, University of Cambridge, 1995.
20. C.J.C.H. Watkins, “Learning from Delayed Rewards,” Ph.D. dissertation, University of Cambridge, 1989.
21. J. Peng and R.J. Williams, “Incremental Multi-Step Q-Learning,” *Machine Learning*, vol. 22, no. 1-3, pp. 283–290, 1996. doi: 10.1023/A:1018076709321.
22. L. Busoniu, R. Babuska, B. De Schutter, and D. Ernst, *Reinforcement Learning and Dynamic Programming Using Function Approximators*. Boca Raton, FL: CRC Press, 2010.
23. D. Chapman and L.P. Kaelbling, “Input Generalization in Delayed Reinforcement Learning: An Algorithm and Performance Comparisons,” in *International Joint Conference on Artificial Intelligence (IJCAI)*, 1991.
24. A.K. McCallum, “Reinforcement Learning with Selective Perception and Hidden State,” Ph.D. dissertation, University of Rochester, 1995.
25. W.T.B. Uther and M.M. Veloso, “Tree Based Discretization for Continuous State Space Reinforcement Learning,” in *AAAI Conference on Artificial Intelligence (AAAI)*, 1998.
26. M.J. Kochenderfer, “Adaptive Modelling and Planning for Learning Intelligent Behaviour,” Ph.D. dissertation, University of Edinburgh, 2006.
27. R. Dearden, N. Friedman, and S.J. Russell, “Bayesian Q-Learning,” in *AAAI Conference on Artificial Intelligence (AAAI)*, 1998.
28. Y. Engel, S. Mannor, and R. Meir, “Reinforcement Learning with Gaussian Processes,” in *International Conference on Machine Learning (ICML)*, 2005.
29. M. Ghavamzadeh and Y. Engel, “Bayesian Policy Gradient Algorithms,” in *Advances in Neural Information Processing Systems (NIPS)*, 2006.
30. L. Busoniu, R. Babuska, and B. De Schutter, “A Comprehensive Survey of Multi-agent Reinforcement Learning,” *IEEE Transactions on Systems Science and Cybernetics Part*, vol. 38, no. 2, pp. 156–172, 2008. doi: 10.1109/TSMCC.2007.913919.

6

State Uncertainty

Mykel J. Kochenderfer

The previous two chapters discussed sequential decision-making problems in which the current state is known by the agent. Because of sensor limitations or noise, the state might not be perfectly observable. This chapter discusses sequential decision problems with state uncertainty and methods for computing optimal and approximately optimal solutions.

6.1 Formulation

A sequential decision problem with state uncertainty can be modeled as a *partially observable Markov decision process* (POMDP). A POMDP is an extension to the MDP formulation introduced in Chapter 4. In a POMDP, a model specifies the probability of making a particular observation given the current state.

6.1.1 Example Problem

Suppose we are assigned the task of taking care of a baby. We decide when to feed the baby on the basis of whether the baby is crying. Crying is a noisy indication that the baby is hungry. There is a 10% chance the baby cries when not hungry, and there is a 80% chance the baby cries when hungry.

The dynamics are as follows. If we feed the baby, then the baby stops being hungry at the next time step. If the baby is not hungry and we do not feed the baby, then 10% of the time the baby may become hungry at the next time step. Once hungry, the baby continues being hungry until fed.

The cost of feeding the baby is 5 and the cost of the baby being hungry is 10. These costs are additive, and so if we feed the baby when the baby is hungry, then there is a cost of 15. We want to find the optimal strategy assuming an infinite horizon with a discount factor of 0.9. Figure 6.1 shows the structure of the crying-baby problem as a dynamic decision network.

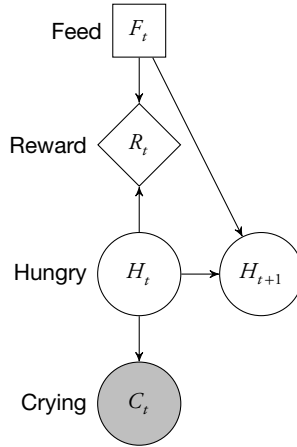


Figure 6.1 Crying-baby problem structure.

6.1.2 Partially Observable Markov Decision Processes

A POMDP is an MDP with an observation model. The probability of observing o given state s is written $O(o | s)$. In some formulations, the observation can also depend on the action a , and so we can write $O(o | s, a)$. The decisions in a POMDP at time t can only be based on the history of observations $o_{1:t}$. Instead of keeping track of arbitrarily long histories, it is common to keep track of the *belief state*. A belief state is a distribution over states. In belief state b , probability $b(s)$ is assigned to being in state s . A policy in a POMDP is a mapping from belief states to actions. The structure of a POMDP can be represented using the dynamic decision network in Figure 6.2.

6.1.3 Policy Execution

Algorithm 6.1 outlines how a POMDP policy is executed. We choose actions on the basis of the policy evaluated at the current belief state. Different ways to represent policies will be discussed in this chapter. When we receive a new observation and reward, we update our belief state. Belief-state updating is discussed in Section 6.2.

6.1.4 Belief-State Markov Decision Processes

A POMDP is really an MDP in which the states are belief states. We sometimes call the MDP over belief states a *belief-state MDP*. The state space of a belief-state MDP is simply the set of all possible beliefs, $\mathcal{B}$, in the POMDP. If there are n discrete states, then $\mathcal{B}$ is a subset of $\mathbb{R}^n$. The set of actions in the belief-state MDP is exactly the same

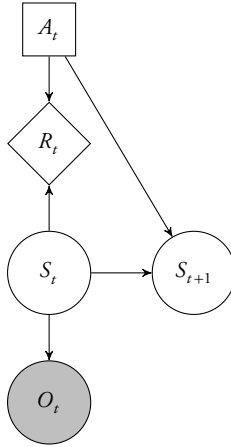


Figure 6.2 POMDP problem structure.

Algorithm 6.1 POMDP policy execution

```

1: function POMDPPOLICYEXECUTION( $\pi$ )
2:    $b \leftarrow$  initial belief state
3:   loop
4:     Execute action  $a = \pi(b)$ 
5:     Observe  $o$  and reward  $r$ 
6:      $b \leftarrow$  UPDATEBELIEF( $b, a, o$ )

```

as for the POMDP. The state transition function $\tau(b' | b, a)$ is given by

$$\tau(b' | b, a) = P(b' | b, a) \quad (6.1)$$

$$= \sum_o P(b' | b, a, o) P(o | b, a) \quad (6.2)$$

$$= \sum_o P(b' | b, a, o) \sum_{s'} P(o | b, a, s') P(s' | b, a) \quad (6.3)$$

$$= \sum_o P(b' | b, a, o) \sum_{s'} O(o | s') \sum_s P(s' | b, a, s) P(s | b, a) \quad (6.4)$$

$$= \sum_o P(b' | b, a, o) \sum_{s'} O(o | s') \sum_s T(s' | s, a) b(s). \quad (6.5)$$

Here, $P(b' | b, a, o) = \delta_{b'}(\text{UPDATEBELIEF}(b, a, o))$, with δ being the Kronecker delta function. The immediate reward in the belief-state MDP is

$$R(b, a) = \sum_s R(s, a) b(s). \quad (6.6)$$

Solving belief-state MDPs is challenging because the state space is continuous. We can use the approximate dynamic programming techniques presented in Section 4.5, but we can often do better by taking advantage of the structure of the belief-state MDP. This chapter will discuss such techniques after elaborating on how to update beliefs.

6.2 Belief Updating

Given an initial belief state, we can update our belief state using recursive Bayesian estimation based on the last observation and action executed. The update can be done exactly for problems with discrete states and for problems with linear-Gaussian dynamics and observations. For general problems with continuous state spaces, we often have to rely on approximation methods. This section presents methods for belief updating.

6.2.1 Discrete State Filter

In problems with a discrete state space, applying recursive Bayesian estimation is straightforward. Suppose our initial belief state is b , and we observe o after executing a . Our

new belief state b' is given by

$$b'(s') = P(s' \mid o, a, b) \quad (6.7)$$

$$\propto P(o \mid s', a, b)P(s' \mid a, b) \quad (6.8)$$

$$\propto O(o \mid s', a)P(s' \mid a, b) \quad (6.9)$$

$$\propto O(o \mid s', a) \sum_s P(s' \mid a, b, s)P(s \mid a, b) \quad (6.10)$$

$$\propto O(o \mid s', a) \sum_s T(s' \mid s, a)b(s). \quad (6.11)$$

The observation space can be continuous without posing any difficulty in computing the equation above exactly. The value $O(o \mid s', a)$ would represent a probability density rather than a probability mass.

To illustrate belief state updating, we will use the crying-baby problem. We simply apply Equation (6.11) to the model outlined in Section 6.1.1. Here are the first six steps of one potential scenario.

1. We begin with an initial belief state that assigns $b(h^0) = 0.5$ and $b(h^1) = 0.5$; in other words, uniform probability is assigned to whether the baby is hungry. If we had some prior belief that babies tend to be hungry more often than not, then we could have chosen a different initial belief. For compactness, we will represent our beliefs as tuples, $(b(h^0), b(h^1))$. In this case, our initial belief state is $(0.5, 0.5)$.
2. We do not feed the baby and the baby cries. According to Equation (6.11), the new belief state is $(0.0928, 0.9072)$. Although the baby is crying, it is only a noisy indication that the baby is actually hungry.
3. We feed the baby and the baby stops crying. Because we know that feeding the baby deterministically makes the baby not hungry, the result of our belief update is $(1, 0)$.
4. We do not feed the baby and the baby does not cry. In the prior step, we were certain that the baby was not hungry, and the dynamics specify that the baby becomes hungry at the next time step only 10% of the time. The fact that the baby is not crying further reduces our belief that the baby is hungry. Our new belief state is $(0.9759, 0.0241)$.
5. Again, we do not feed the baby and the baby does not cry. Our belief that the baby is hungry increases slightly. The new belief state is $(0.9701, 0.0299)$.
6. We do not feed the baby and the baby begins to cry. Our new belief is then $(0.4624, 0.5376)$. Because we were fairly certain that the baby was not hungry to begin with, our confidence that the baby is now hungry is significantly lower than when the baby started crying in the second step.

6.2.2 Linear-Gaussian Filter

If we generalize the linear-Gaussian dynamics in Section 4.4 to partial observability, we find that we can perform exact belief updates by using what is known as a *Kalman filter*. The dynamics and observations have the following form:

$$T(\mathbf{z} \mid \mathbf{s}, \mathbf{a}) = \mathcal{N}(\mathbf{z} \mid \mathbf{T}_s \mathbf{s} + \mathbf{T}_a \mathbf{a}, \Sigma_s) \quad (6.12)$$

$$O(\mathbf{o} \mid \mathbf{s}) = \mathcal{N}(\mathbf{o} \mid \mathbf{O}_s \mathbf{s}, \Sigma_o). \quad (6.13)$$

Hence, the continuous dynamics and observation models are specified using matrices $\mathbf{T}_s$, $\mathbf{T}_a$, Σ_s , $\mathbf{O}_s$, and Σ_o .

We assume that the initial belief state is represented by a Gaussian:

$$b(\mathbf{s}) = \mathcal{N}(\mathbf{s} \mid \mu_b, \Sigma_b). \quad (6.14)$$

Under the linear-Gaussian assumptions for the dynamics and observations, it can be shown that the belief state can be updated as follows:

$$\mathbf{K} \leftarrow \Sigma_p \mathbf{O}_s^\top (\mathbf{O}_s \Sigma_p \mathbf{O}_s^\top + \Sigma_o)^{-1} \quad (6.15)$$

$$\mu_b \leftarrow \mu_p + \mathbf{K}(\mathbf{o} - \mathbf{O}_s \mu_p) \quad (6.16)$$

$$\Sigma_b \leftarrow (\mathbf{I} - \mathbf{K} \mathbf{O}_s) \Sigma_p, \quad (6.17)$$

where $\mu_p = \mathbf{T}_s \mu_b + \mathbf{T}_a \mathbf{a}$ and $\Sigma_p = \mathbf{T}_s \Sigma_b \mathbf{T}_s^\top + \Sigma_s$ are the predicted mean and covariance prior to an observation, respectively. The matrix $\mathbf{K}$ is called the *Kalman gain*. Kalman filters are often applied to systems that do not actually have linear-Gaussian dynamics. A variety of different modifications to the basic Kalman filter have been proposed to better accommodate nonlinear dynamics, as discussed in Section 6.7.

6.2.3 Particle Filter

If the state space is large or continuous and the dynamics are not well approximated by a linear-Gaussian model, then a sampling-based approach can be used to perform belief updates. The belief state is represented as a collection of *particles*, which are simply samples from the state space. The algorithm for adjusting these particles based on observations is known as a *particle filter*. There are many different variations of the particle filter, including versions in which the particles are assigned weights.

Our belief b is simply a set of samples from the state space. Updating b is based on a generative model G . We can draw a sample $(s', o') \sim G(s, a)$, which gives the next state s' and observation o' given the current state s and action a . The generative model can be implemented as a black-box simulator, without any explicit knowledge of the actual transition or observation probabilities.

Algorithm 6.2 returns the updated belief state b' based on the current belief state b , action a , and observation o . The process for generating a set of $|b|$ new particles is simple. Each sample is generated by randomly selecting a sample in b and then drawing samples $(s', o') \sim G(s, a)$ until the sampled o' matches the observed o . The sampled s' is then added to the new belief state b' .

Algorithm 6.2 Particle filter with rejection

```

1: function UPDATEBELIEF( $b, a, o$ )
2:    $b' \leftarrow \emptyset$ 
3:   for  $i \leftarrow 1$  to  $|b|$ 
4:     repeat
5:        $s \leftarrow$  random state in  $b$ 
6:        $(s', o') \sim G(s, a)$ 
7:     until  $o' = o$ 
8:     Add  $s'$  to  $b'$ 
9:   return  $b'$ 
  
```

The problem with the version of the particle filter in Algorithm 6.2 is that many draws from the generative model may be required until the sampled observation is consistent with actual observation. The problem of rejecting many observation draws becomes especially apparent when the observation space is large or continuous. This issue was observed in Section 2.2.5 where we were trying to perform inference using direct sampling from a Bayesian network. The remedy presented in that section was to not sample the observed values but to weight the results using the likelihood of the observations.

Algorithm 6.3 is a version of a particle filter that does not involve rejecting samples. In this version, the generative model only returns states, instead of both states and observations. An observation model is required that specifies $O(o \mid s, a)$, which can be either a probability mass function or a probability density function depending on whether the observation space is continuous.

The algorithm is broken into two stages. The first stage involves generating $|b|$ new samples by randomly selecting samples in b and then propagating them forward using the generative model. For each of the new samples s'_i , we compute a corresponding weight w_i based on $O(o \mid s'_i, a)$, where o is the actual observation and a is the action taken. The second stage involves building the updated belief state b' by drawing $|b|$ samples from the set of new state samples with probability proportional to their weights.

In both versions of the particle filter presented here, it can be shown that as the number of particles increases, the distribution represented by the particles approaches the true posterior distribution. However, in practice, particle filters can fail. Because of the random nature of the particle filter, it is possible that a series of samples can lead

Algorithm 6.3 Particle filter without rejection

```

1: function UPDATEBELIEF( $b, a, o$ )
2:    $b' \leftarrow \emptyset$ 
3:   for  $i \leftarrow 1$  to  $|b|$ 
4:      $s_i \leftarrow$  random state in  $b$ 
5:      $s'_i \sim G(s_i, a)$ 
6:      $w_i \leftarrow O(o \mid s'_i, a)$ 
7:   for  $i \leftarrow 1$  to  $|b|$ 
8:     Randomly select  $k$  with probability proportional to  $w_k$ 
9:     Add  $s'_k$  to  $b'$ 
10:  return  $b'$ 

```

to no particles near the true state. This problem, known as *particle deprivation*, can be mitigated to some extent by introducing additional noise to the particles.

6.3 Exact Solution Methods

As discussed earlier, a policy for a POMDP is a mapping from belief states to actions. This section explains how to compute and represent optimal policies.

6.3.1 Alpha Vectors

For now, let us assume that we are interested in computing the optimal policy for a discrete state POMDP with a one-step horizon. We know $U^*(s) = \max_a R(s, a)$, but because we do not know the state exactly in a POMDP, we have

$$U^*(b) = \max_a \sum_s b(s) R(s, a), \quad (6.18)$$

where b is our current belief state. If we let α_a represent $R(\cdot, a)$ as a vector and $\mathbf{b}$ represent our belief state as a vector, then we can rewrite Equation (6.18) as

$$U^*(\mathbf{b}) = \max_a \alpha_a^\top \mathbf{b}. \quad (6.19)$$

The α_a in the equation above is often referred to as an *alpha vector*. We have an alpha vector for each action in this single-step POMDP. These alpha vectors define hyperplanes in belief space. As can be seen in Equation (6.19), the optimal value function is piecewise-linear and convex.

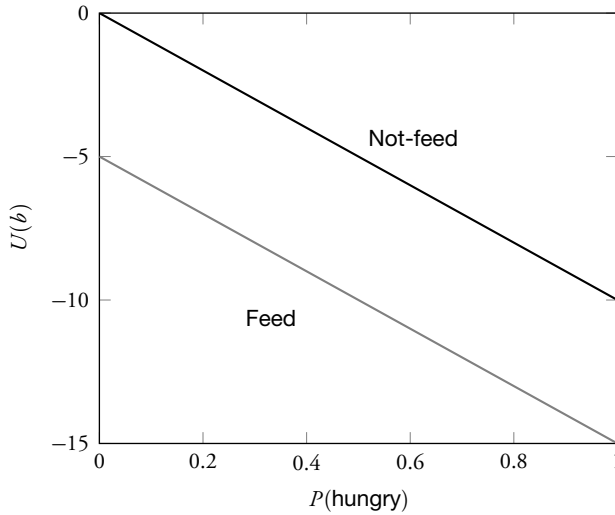


Figure 6.3 Alpha vectors for a one-step version of the crying-baby problem.

Figure 6.3 shows the alpha vectors associated with the crying-baby problem. If the belief vector is represented by the pair $(b(\text{not-hungry}), b(\text{hungry}))$, then the two alpha vectors are

$$\alpha_{\text{not-feed}} = (0, -10) \quad (6.20)$$

$$\alpha_{\text{feed}} = (-5, -15). \quad (6.21)$$

What is apparent from the plot is that regardless of our current beliefs, the one-step optimal policy is to not feed the baby. Given the dynamics of the problem, we do not see any potential benefit of feeding the baby until at least one step later.

6.3.2 Conditional Plans

The alpha vectors introduced in the computation of the optimal one-step policy can be generalized to an arbitrary horizon. In multistep POMDPs, we can think of a policy as a *conditional plan* represented as a tree. We start at the root node, which tells us what action to take at the first time step. After taking that action, we transition to one of the child nodes depending on what we observe. That child node tells us what action to take. We then proceed down the tree.

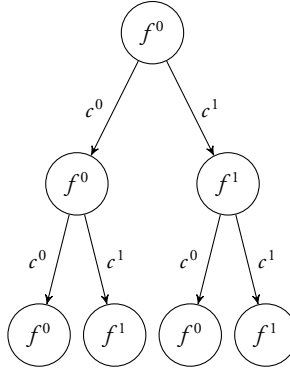


Figure 6.4 Example three-step conditional plan.

Figure 6.4 shows an example of a three-step plan for the crying-baby problem. The root node specifies that we do not feed the baby at the first step. For the second time step, as indicated by the directed edges, we feed the baby if there is crying; otherwise, we do not. At the third time step, we again feed the baby only if there is crying.

We can recursively compute $U^p(s)$, the expected utility associated with conditional plan p when starting in state s :

$$U^p(s) = R(s, a) + \sum_{s'} T(s' | s, a) \sum_o O(o | s', a) U^{p(o)}(s'), \quad (6.22)$$

where a is the action associated with the root node of p and $p(o)$ represents the subplan associated with observation o . We can compute the expected utility associated with a belief state as follows:

$$U^p(b) = \sum_s U^p(s) b(s). \quad (6.23)$$

We can use the alpha vector α_p to represent the vectorized version of U^p . If $\mathbf{b}$ is the belief vector, then we can write

$$U^p(\mathbf{b}) = \alpha_p^\top \mathbf{b}. \quad (6.24)$$

If we maximize over the space of all possible plans up to the planning horizon, then we can find

$$U^*(\mathbf{b}) = \max_p \alpha_p^\top \mathbf{b}. \quad (6.25)$$

Hence, the finite horizon optimal value function is piecewise-linear and convex. We simply execute the action at the root node of the plan that maximizes $\alpha_p^\top \mathbf{b}$.

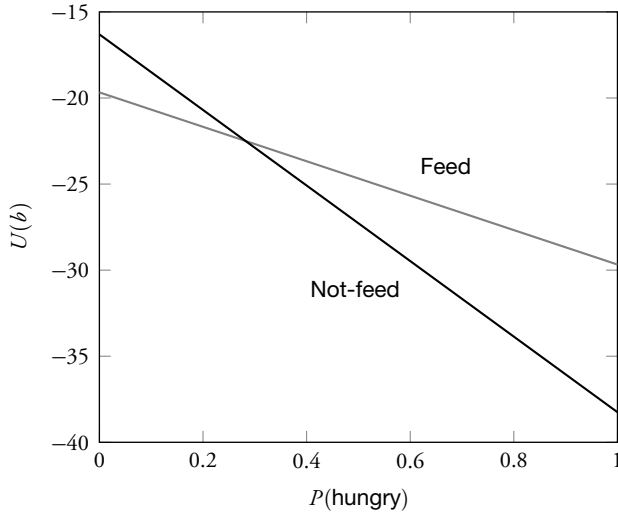


Figure 6.5 Optimal policy for the crying-baby problem.

6.3.3 Value Iteration

It is generally infeasible to enumerate every possible h -step plan to find the one that maximizes Equation (6.25) from the current belief state; the number of nodes in an h -step conditional plan is $(|O|^h - 1)/(|O| - 1)$. Associated with each node are $|A|$ actions. Hence, we have $|A|^{(|O|^h - 1)/(|O| - 1)}$ possible h -step plans. Even for our crying-baby problem with two actions and two observations, there are 2^{63} six-step conditional plans—too many to enumerate.

The idea in POMDP value iteration is to iterate over all the one-step plans and toss out the plans that are not optimal for any initial belief state. The remaining one-step plans are then used to generate potentially optimal two-step plans. Again, plans that are not optimal for any belief state are discarded. The process repeats until the desired horizon is reached. Identifying the plans that are *dominated* at certain belief states by other plans can be done using linear programming.

Figure 6.5 shows the two, nondominated, alpha vectors for the crying-baby problem with a discount factor of 0.9. The two alpha vectors intersect when $P(\text{hungry}) = 0.28206$. As indicated in the figure, we only want to feed the baby if $P(\text{hungry}) > 0.28206$. Of course, for this problem, it would have been easier to simply store this threshold instead of using alpha vectors, but for higher dimensional problems, alpha vectors often provide a compact representation of the policy.

Discarding dominated plans can significantly reduce the computation required to find the optimal set of alpha vectors. For many problems, the majority of the potential plans are dominated by at least one other plan. However, in the worst case, an exact solution for a general finite-horizon POMDP is *PSPACE-complete*, which is a complexity class that includes NP-complete problems and is suspected to include problems even more difficult. General infinite-horizon POMDPs have been shown to be *uncomputable*. Hence, there has been a tremendous amount of research recently on approximation methods, which will be discussed in the remainder of this chapter.

6.4 Offline Methods

Offline POMDP solution methods involve performing all or most of the computation prior to execution. In practice, we are generally restricted to finding only approximately optimal solutions. Some methods represent policies as alpha vectors, whereas others use finite-state controllers.

6.4.1 Fully Observable Value Approximation

A simple approximation technique is called *QMDP*. The idea is to create a set of alpha vectors, one for each action, based on the state-action value function $Q(s, a)$ under full observability. We can use value iteration to compute the alpha vectors. If we initialize $\alpha_a^{(0)}(s) = 0$ for all s , then we can iterate

$$\alpha_a^{(k+1)}(s) = R(s, a) + \gamma \sum_{s'} T(s' | s, a) \max_{a'} \alpha_{a'}^{(k)}(s'). \quad (6.26)$$

Each iteration requires $O(|A|^2|S|^2)$ operations. As $k \rightarrow \infty$, the resulting set of $|A|$ alpha vectors can be used to estimate the value function. The value function at belief state $\mathbf{b}$ is given by $\max_a \alpha_a^\top \mathbf{b}$, and the approximately optimal action is given by $\arg \max_a \alpha_a^\top \mathbf{b}$.

The QMDP method assumes all state uncertainty disappears at the next time step. It can be shown that QMDP provides an upper bound on the value function. In other words, $\max_a \alpha_a^\top \mathbf{b} \geq U^*(\mathbf{b})$ for all $\mathbf{b}$. QMDP tends to have difficulty with problems with information-gathering actions, such as “look over your right shoulder when changing lanes.” However, the method performs extremely well in many real problems in which the particular choice of action has little impact on the reduction in state uncertainty.

6.4.2 Fast Informed Bound

Just as with the QMDP approximation, the *fast informed bound* (FIB) method computes a single alpha vector for each action. However, the fast informed bound takes into account, to some extent, partial observability. Instead of using the iteration of

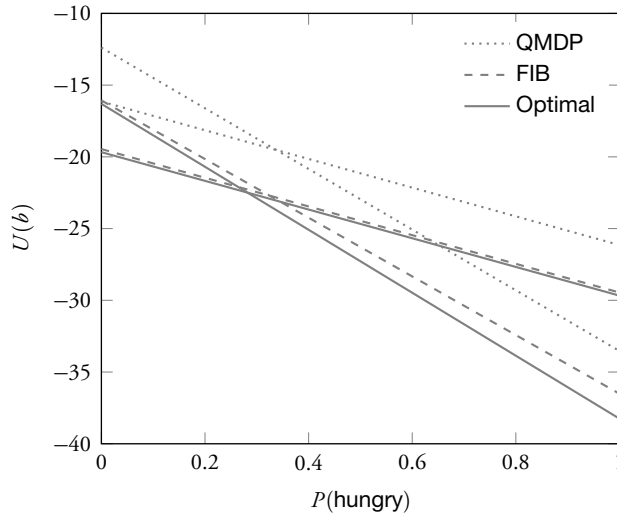


Figure 6.6 QMDP, FIB, and optimal alpha vectors for the crying-baby problem.

Equation (6.26), we use

$$\alpha_a^{(k+1)}(s) = R(s, a) + \gamma \sum_o \max_{a'} \sum_{s'} O(o | s', a) T(s' | s, a) \alpha_{a'}^{(k)}(s'). \quad (6.27)$$

Each iteration requires $O(|A|^2|S|^2|O|)$ operations, only a factor of $|O|$ more than QMDP. At all belief states, the fast informed bound provides an upper bound on the optimal value function that is no higher than that of QMDP. Figure 6.6 compares the alpha vectors associated with QMDP, FIB, and the optimal policy for the crying baby problem.

6.4.3 Point-Based Value Iteration

There is a family of approximation methods that involves backing up alpha vectors associated with a limited number of points in the belief space. Let us denote the set of belief points $B = \{\mathbf{b}_1, \dots, \mathbf{b}_n\}$ and their associated alpha vectors $\Gamma = \{\alpha_1, \dots, \alpha_n\}$. Given these n alpha vectors, we can estimate the value function at any new point $\mathbf{b}$ as follows:

$$U^\Gamma(\mathbf{b}) = \max_{\alpha \in \Gamma} \alpha^\top \mathbf{b} = \max_{\alpha \in \Gamma} \sum_s \alpha(s) b(s). \quad (6.28)$$

For the moment, let us assume that these belief points are given to us; we will discuss how to choose these later in Section 6.4.5. We would like to initialize the alpha vectors in Γ such that $U^\Gamma(\mathbf{b}) \leq U^*(\mathbf{b})$ for all $\mathbf{b}$. One way to compute such a lower bound is to initialize all the components of all n alpha vectors to

$$\max_a \sum_{t=0}^{\infty} \gamma^t \min_s R(s, a) = \frac{1}{1-\gamma} \max_a \min_s R(s, a). \quad (6.29)$$

As we perform backups starting with this initial set of alpha vectors, we guarantee that $U(\mathbf{b})$ at each iteration never decreases for any $\mathbf{b}$.

We may update the value function at belief b based on the n alpha vectors

$$U(b) \leftarrow \max_a \left[R(b, a) + \gamma \sum_o P(o | a, b) U(b') \right], \quad (6.30)$$

where b' is as determined by $\text{UPDATEBELIEF}(b, a, o)$, $U(b')$ is as evaluated by Equation (6.28), and

$$P(o | b, a) = \sum_s b(s) \sum_{s'} O(o | s', a) T(s' | s, a). \quad (6.31)$$

We know from Bayes' rule that

$$b'(s') = \frac{O(o | s', a)}{P(o | b, a)} \sum_s T(s' | s, a) b(s). \quad (6.32)$$

Combining Equations (6.28), (6.30), and (6.32) and simplifying, we get the update

$$U(b) \leftarrow \max_a \left[R(b, a) + \gamma \sum_o \max_{\alpha \in \Gamma} \sum_s b(s) \sum_{s'} O(o | s', a) T(s' | s, a) \alpha(s') \right]. \quad (6.33)$$

Besides simply updating the value at $\mathbf{b}$, we can compute an alpha vector at $\mathbf{b}$ by using Algorithm 6.4. Point-based value iteration approximation algorithms update the alpha vectors at the n belief states until convergence. These n alpha vectors can then be used to approximate the value function anywhere in the belief space.

6.4.4 Randomized Point-Based Value Iteration

The point-based value iteration approach discussed in Section 6.4.3 associates an alpha vector for each of the selected points in the belief space. To reduce the amount of computation required to perform an update of all the belief points, we can attempt to limit the number of alpha vectors representing the value function using the approach outlined in Algorithm 6.5.

Algorithm 6.4 Backup belief

```

1: function BACKUPBELIEF( $\Gamma, \mathbf{b}$ )
2:   for  $a \in A$ 
3:     for  $o \in O$ 
4:        $\mathbf{b}' \leftarrow \text{UPDATEBELIEF}(\mathbf{b}, a, o)$ 
5:        $\alpha_{a,o} \leftarrow \arg \max_{\alpha \in \Gamma} \alpha^\top \mathbf{b}'$ 
6:     for  $s \in S$ 
7:        $\alpha_a(s) \leftarrow R(s, a) + \gamma \sum_{s', o} O(o \mid s', a) T(s' \mid s, a) \alpha_{a,o}(s')$ 
8:    $\alpha \leftarrow \arg \max_{\alpha_a} \alpha_a^\top \mathbf{b}$ 
9:   return  $\alpha$ 

```

The algorithm begins by initializing Γ with a single alpha vector with all components set to Equation (6.29), which lower bounds the value function. Given this Γ and our belief points B , we call $\text{RANDOMIZEDPOINTBASEDUPDATE}(B, \Gamma)$ to create a new set of alpha vectors that provides a tighter lower bound on the value function. These new alpha vectors can be improved on further through another call to this function. The process repeats until convergence.

Each update involves finding a set of alpha vectors Γ' that improves on the value function represented by Γ at the points in B . In other words, the update finds a set Γ' such that $U^{\Gamma'}(\mathbf{b}) \geq U^\Gamma(\mathbf{b})$ for all $\mathbf{b} \in B$. We begin by initializing Γ' to the empty set and the set B' to B . We then take a point $\mathbf{b}$ randomly from B' and call $\text{BACKUPBELIEF}(\mathbf{b}, \Gamma)$ to get a new alpha vector α . If this alpha vector improves the value at $\mathbf{b}$, then we add it to Γ' ; otherwise, we find the alpha vector in Γ that dominates at $\mathbf{b}$ and add it to Γ' . The set B' then becomes the set of points that still have not been improved by Γ' . At each iteration, B' becomes smaller, and the process terminates when B' is empty.

6.4.5 Point Selection

Many point-based value iteration algorithms involve starting with B initialized to a set containing only the initial belief state b_0 and then iteratively expanding that set. One of the simplest ways to expand a set B is to select actions from each belief state in B (based on some exploration strategy from Section 5.1.3) and then add the resulting belief states to B (Algorithm 6.6). This process requires sampling observations from a belief state given an action (Algorithm 6.7).

Other approaches attempt to disperse the points throughout the reachable state space. For example, Algorithm 6.8 iterates through B , tries each available action, and adds the new belief states that are furthest from any of the points already in the set. There are many ways to measure distance between two belief states; the algorithm shown uses the L_1 distance metric, where the distance between b and b' is given by $\sum_s |b(s) - b'(s)|$.

Algorithm 6.5 Randomized point-based backup

```

1: function RANDOMIZEDPOINTBASEDUPDATE( $B, \Gamma$ )
2:    $\Gamma' \leftarrow \emptyset$ 
3:    $B' \leftarrow B$ 
4:   repeat
5:      $\mathbf{b} \leftarrow$  belief point sampled uniformly at random from set  $B'$ 
6:      $\alpha \leftarrow \text{BACKUPBELIEF}(\mathbf{b}, \Gamma)$ 
7:     if  $\alpha^\top \mathbf{b} \geq U^\Gamma(\mathbf{b})$ 
8:       Add  $\alpha$  to  $\Gamma'$ 
9:     else
10:      Add  $\alpha' = \arg \max_{\alpha \in \Gamma} \alpha^\top \mathbf{b}$  to  $\Gamma'$ 
11:       $B' \leftarrow \{\mathbf{b} \in B \mid U^{\Gamma'}(\mathbf{b}) < U^\Gamma(\mathbf{b})\}$ 
12:   until  $B' = \emptyset$ 
13:   return  $\Gamma'$ 

```

Algorithm 6.6 Expand belief points with random actions

```

1: function EXPANDBELIEFPOINTS( $B$ )
2:    $B' \leftarrow B$ 
3:   for  $b \in B$ 
4:      $a \leftarrow$  random action selected uniformly from action space  $\mathcal{A}$ 
5:      $o \leftarrow \text{SAMPLEOBSERVATION}(b, a)$ 
6:      $b' \leftarrow \text{UPDATEBELIEF}(b, a, o)$ 
7:     Add  $b'$  to  $B'$ 
8:   return  $B'$ 

```

Algorithm 6.7 Sample observation

```

1: function SAMPLEOBSERVATION( $b, a$ )
2:    $s \sim b$ 
3:    $s' \leftarrow$  random state selected with probability  $T(s' \mid s, a)$ 
4:    $o \leftarrow$  random observation selected with probability  $O(o \mid a, s')$ 
5:   return  $o$ 

```

Algorithm 6.8 Expand belief points with exploratory actions

```

1: function EXPANDBELIEFPPOINTS( $B$ )
2:    $B' \leftarrow B$ 
3:   for  $b \in B$ 
4:     for  $a \in A$ 
5:        $o \leftarrow \text{SAMPLEOBSERVATION}(b, a)$ 
6:        $b_a \leftarrow \text{UPDATEBELIEF}(b, a, o)$ 
7:        $a \leftarrow \arg \max_a \min_{b' \in B'} \sum_s |b'(s) - b_a(s)|$ 
8:       Add  $b_a$  to  $B'$ 
9:   return  $B'$ 

```

6.4.6 Linear Policies

As discussed in Section 6.2.2, the belief state in a problem with linear-Gaussian dynamics can be represented by a Gaussian distribution $\mathcal{N}(\mu_b, \Sigma_b)$. If the reward function is quadratic, as assumed in Section 4.4, then it can be shown that the optimal policy can be computed exactly offline. In fact, the solution is identical to the perfect observability case outlined in Section 4.4, but the μ_b computed by the Kalman filter is used in place of the true state. With each observation, we simply use the Kalman filter to update our μ_b , and then we matrix multiply μ_b with the policy matrix given in Section 4.4 to determine the optimal action.

6.5 Online Methods

Online methods determine the optimal policy by planning from the current belief state. The belief states reachable from the current state are typically small compared with the full belief space. Many online methods use a depth-first tree-based search up to some horizon. The time complexity of these online algorithms is generally exponential in the horizon. Although online methods require more computation per decision step during execution than offline approaches, online methods are sometimes easier to apply to high-dimensional problems.

6.5.1 Lookahead with Approximate Value Function

We can use the one-step lookahead strategy online to improve on a policy that has been computed offline. If b is the current belief state, then the one-step lookahead policy is given by

$$\pi(b) = \arg \max_a \left[R(b, a) + \gamma \sum_o P(o \mid b, a) U(\text{UPDATEBELIEF}(b, a, o)) \right], \quad (6.34)$$

where U is an approximate value function. This approximate value function may be represented by alpha vectors computed offline using strategies such as QMDP, fast informed bound, or point-based value iteration discussed earlier. Experiments have shown that for many problems, a one-step lookahead can significantly improve performance over the base offline strategy.

The approximate value function may also be estimated by sampling from a rollout policy as introduced in Section 4.6.4 but with modifications to handle partial observability as shown in Algorithm 6.9. The generative model G returns a sampled next state s' and reward r given state s and action a . This algorithm uses a single rollout policy π_0 , but we can use a set of rollout policies and evaluate them in parallel. The policy that results in the largest value is the one that is used to estimate the value at that particular belief state.

Algorithm 6.9 Rollout evaluation

```

1: function ROLLOUT( $b, d, \pi_0$ )
2:   if  $d = 0$ 
3:     return 0
4:    $a \sim \pi_0(b)$ 
5:    $s \sim b$ 
6:    $(s', o, r) \sim G(s, a)$ 
7:    $b' \leftarrow \text{UPDATEBELIEF}(b, a, o)$ 
8:   return  $r + \gamma \text{ROLLOUT}(b', d - 1, \pi_0)$ 
```

An alternative to summing over all possible observations in Equation (6.34) is to use sampling. We can generate n observations for each action through independent calls to $\text{SAMPLEOBSERVATION}(b, a)$ and then compute

$$\pi(b) = \arg \max_a \left[R(b, a) + \gamma \frac{1}{n} \sum_{i=1}^n U(\text{UPDATEBELIEF}(b, a, o_{a,i})) \right]. \quad (6.35)$$

This strategy is particularly useful when the observation space is large.

6.5.2 Forward Search

The one-step lookahead strategy can be extended to look an arbitrary depth into the future. Algorithm 6.10 defines the function $\text{SELECTACTION}(b, d, U)$, which returns the pair (a^*, u^*) defining the best action and the expected utility, given the current belief b , depth d , and approximate value function U .

When $d = 0$, no action is able to be selected, and so a^* is NIL and the utility is $U(b)$. When $d > 0$, we compute the value for every available action and return the best action and its associated value. To compute the value for an action a , we evaluate

$$R(b, a) + \gamma \sum_o P(o \mid b, a) U_{d-1}(\text{UPDATEBELIEF}(b, a, o)). \quad (6.36)$$

In the equation above, $U_{d-1}(b')$ is the expected utility returned by the recursive call $\text{SELECTACTION}(b', d-1)$. The complexity is given by $O(|A|^d |O|^d)$. Algorithm 6.10 can be modified to sample n observations instead of enumerating all $|O|$ of them, similar to what is done in Equation (6.35). The complexity then becomes $O(|A|^d n^d)$.

Algorithm 6.10 Forward search online policy

```

1: function SELECTACTION( $b, d$ )
2:   if  $d = 0$ 
3:     return (NIL,  $U(b)$ )
4:   ( $a^*, u^*$ )  $\leftarrow$  (NIL,  $-\infty$ )
5:   for  $a \in A$ 
6:      $u \leftarrow R(b, a)$ 
7:     for  $o \in O$ 
8:        $b' \leftarrow \text{UPDATEBELIEF}(b, a, o)$ 
9:       ( $a', u'$ )  $\leftarrow \text{SELECTACTION}(b', d-1)$ 
10:       $u \leftarrow u + \gamma P(o \mid b, a) u'$ 
11:   if  $u > u^*$ 
12:     ( $a^*, u^*$ )  $\leftarrow (a, u)$ 
13:   return ( $a^*, u^*$ )

```

6.5.3 Branch and Bound

The branch and bound technique originally introduced in Section 4.6.2 in the context of MDPs can be easily extended to POMDPs. As with the POMDP version of forward search, we have to iterate over the observations and update beliefs—otherwise the algorithm is nearly identical to the MDP version. Again, the ordering of the actions in the for loop is important. To prune as much of the search space as possible, the actions should be enumerated such that their upper bounds decrease in value. In other words, action a_i comes before a_j if $\overline{U}(b, a_i) \geq \overline{U}(b, a_j)$.

We can use QMDP or the fast informed bound as the upper bound function $\overline{U}$. For the lower bound function $\underline{U}$, we can use the value function associated with a *blind policy*, which selects the same action regardless of the current belief state. This value

function can be represented by a set of $|A|$ alpha vectors, which can be computed as follows:

$$\alpha_a^{(k+1)}(s) = R(s, a) + \gamma \sum_{s'} T(s' | s, a) \alpha_a^{(k)}(s'), \quad (6.37)$$

where $\alpha_a^{(0)} = \min_s R(s, a)/(1 - \gamma)$. Equation (6.37) is similar to the QMDP equation in Equation (6.26) except that it does not have a maximization over the alpha vectors on the right-hand side.

So long as $\underline{U}$ and $\overline{U}$ are true lower and upper bounds, the result of the branch and bound algorithm will be the same as the forward search algorithm with $\underline{U}$ as the approximate value function. In practice, branch and bound can significantly reduce the amount of computation required to select an action. The tighter the upper and lower bounds, the more of the search space branch and bound can prune. However, in the worst case, the complexity of branch and bound is no better than that of forward search.

Algorithm 6.11 Branch and bound online policy

```

1: function SELECTACTION( $b, d$ )
2:   if  $d = 0$ 
3:     return (NIL,  $\underline{U}(b)$ )
4:   ( $a^*, \underline{u}$ )  $\leftarrow$  (NIL,  $-\infty$ )
5:   for  $a \in A$ 
6:     if  $\overline{U}(b, a) \leq \underline{u}$ 
7:       return ( $a^*, \underline{u}$ )
8:    $u \leftarrow R(b, a)$ 
9:   for  $o \in O$ 
10:     $b' \leftarrow \text{UPDATEBELIEF}(b, a, o)$ 
11:    ( $a', \underline{u}'$ )  $\leftarrow$  SELECTACTION( $b', d - 1$ )
12:     $u \leftarrow u + \gamma P(o | b, a) \underline{u}'$ 
13:   if  $u > \underline{u}$ 
14:     ( $a^*, \underline{u}$ )  $\leftarrow$  ( $a, u$ )
15:   return ( $a^*, \underline{u}$ )

```

6.5.4 Monte Carlo Tree Search

The Monte Carlo tree search approach for MDPs can be extended to POMDPs, as outlined in Algorithm 6.12. The input to the algorithm is a belief state b , depth d , and rollout policy π_0 . The main difference between the POMDP algorithm and the MDP algorithm in Section 4.6.4 is that the counts and values are associated with *histories* instead of states. A history is a sequence of past observations and actions. For example,

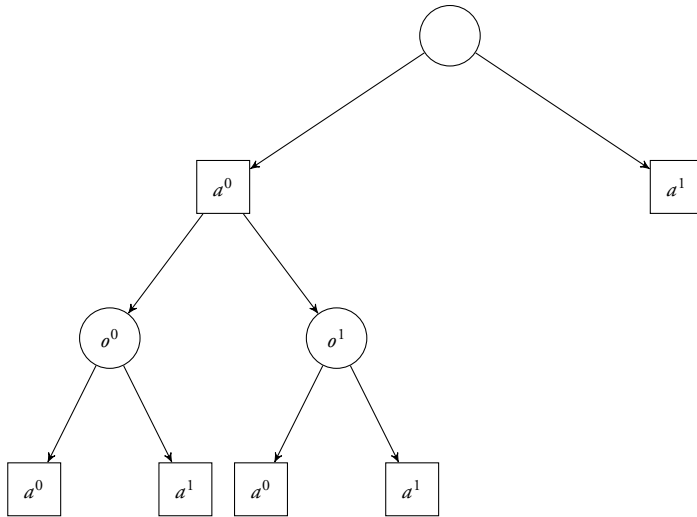


Figure 6.7 Example Monte Carlo tree search history tree.

if we have two actions a^0 and a^1 and two observations o^0 and o^1 , then a possible history could be the sequence $h = a^0 o^1 a^1 o^1 a^0 o^0$. During the execution of the algorithm, we update the value estimates $Q(h, a)$ and counts $N(h, a)$ for a set of history-action pairs.

The histories associated with Q and N may be organized in a tree like the one in Figure 6.7. The root node represents the empty history starting from the initial belief state b . During the execution of the algorithm, the tree structure expands. The layers of the tree alternate between action nodes and observation nodes. Associated with each action node are values $Q(h, a)$ and $N(h, a)$, where the history is determined by the path to the root node. In the algorithm, $N(h) = \sum_a N(h, a)$.

As with the MDP version, the Monte Carlo tree search algorithm is an anytime algorithm. The loop in `SELECTACTION( $b, d$ )` can be terminated at any time, and some solution will be returned. It has been shown that, with a sufficient number of iterations, the algorithm converges to the optimal action.

Prior knowledge can be incorporated into this algorithm through the choice of initialization parameters N_0 and Q_0 as well as the rollout policy. The algorithm does not need to be reinitialized with each decision. The history tree and associated counts and value estimates can be maintained between calls. The observation node associated with the selected action and actual observation becomes the root node at the next time step.

Algorithm 6.12 Monte Carlo tree search

```

1: function SELECTACTION( $b, d$ )
2:    $h \leftarrow \emptyset$ 
3:   loop
4:      $s \sim b$ 
5:     SIMULATE( $s, h, d$ )
6:   return  $\arg \max_a Q(h, a)$ 
7: function SIMULATE( $s, h, d$ )
8:   if  $d = 0$ 
9:     return 0
10:  if  $h \notin T$ 
11:    for  $a \in A(s)$ 
12:       $(N(h, a), Q(h, a)) \leftarrow (N_0(h, a), Q_0(h, a))$ 
13:     $T = T \cup \{h\}$ 
14:    return ROLLOUT( $s, d, \pi_0$ )
15:   $a \leftarrow \arg \max_a \left[ Q(h, a) + c \sqrt{\frac{\log N(h)}{N(h, a)}} \right]$ 
16:   $(s', o, r) \sim G(s, a)$ 
17:   $q \leftarrow r + \gamma \text{SIMULATE}(s', h a o, d - 1)$ 
18:   $N(h, a) \leftarrow N(h, a) + 1$ 
19:   $Q(h, a) \leftarrow Q(h, a) + \frac{q - Q(h, a)}{N(h, a)}$ 
20:  return  $q$ 

```

6.6 Summary

- POMDPs are MDPs over belief states.
- POMDPs are difficult to solve exactly in general but can often be approximated well.
- Policies can be represented as alpha vectors.
- Large problems can often be solved online.

6.7 Further Reading

It was observed in the 1960s that POMDPs can be transformed into MDPs over belief states [1]. Belief state updating in discrete state spaces is a straightforward application of Bayes' rule. A thorough introduction to the Kalman filter and its variants is provided in *Estimation with Applications to Tracking and Navigation* by Bar-Shalom, Li, and Kirubarajan [2]. Arulampalam et al. provide a tutorial on particle filters [3]. *Probabilistic Robotics* by Thrun, Burgard, and Fox discusses different methods for belief updating in the context of robotic applications [4].

Exact solution methods for POMDPs were originally proposed by Smallwood and Sondik [5] and Sondik [6] in the 1970s. There are several surveys of early work on POMDPs [7]–[9]. Kaelbling, Littman, and Cassandra present techniques for identifying dominated plans to improve the efficiency of exact solution methods [10]. Computing exact solutions to POMDPs is intractable in general [11], [12].

Approximation methods for POMDPs have been the focus of considerable research recently. Hauskrecht discusses the relationship between QMDP and the fast informed bound and presents empirical results [13]. Offline approximate POMDP solution algorithms have focused on point-based approximation techniques, as surveyed by Shani, Pineau, and Kaplow [14]. The point-based value iteration (PBVI) algorithm was proposed by Pineau, Gordon, and Thrun [15]. There are other, more involved, point-based value iteration algorithms. Two of the best algorithms, Heuristic Search Value Iteration (HSVI) [16], [17] and Successive Approximations of the Reachable Space under Optimal Policies (SARSOP) [18], involve building search trees through belief space and maintaining upper and lower bounds on the value function. The randomized point-based value iteration algorithm discussed in Section 6.4.4 is based on the Perseus algorithm presented by Spaan and Vlassis [19]. Controller-based solutions have also been explored for concisely representing policies for infinite-horizon problems and removing the need for belief updating during execution [20], [21]. Several online solution methods are surveyed by Ross et al. [22]. Silver and Veness present a Monte Carlo tree search algorithm for POMDPs called Partially Observable Monte Carlo Planning (POMCP) [23].

References

1. K.J. Åström, “Optimal Control of Markov Processes with Incomplete State Information,” *Journal of Mathematical Analysis and Applications*, vol. 10, no. 1, pp. 174–205, 1965. DOI: 10.1016/0022-247X(65)90154-X.
2. Y. Bar-Shalom, X.R. Li, and T. Kirubarajan, *Estimation with Applications to Tracking and Navigation*. New York: Wiley, 2001.
3. M.S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, “A Tutorial on Particle Filters for Online Nonlinear / Non-Gaussian Bayesian Tracking,” *IEEE Transactions on Signal Processing*, vol. 50, no. 2, pp. 174–188, 2002. DOI: 10.1109/78.978374.
4. S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, MA: MIT Press, 2006.
5. R.D. Smallwood and E.J. Sondik, “The Optimal Control of Partially Observable Markov Processes Over a Finite Horizon,” *Operations Research*, vol. 21, no. 5, pp. 1071–1088, 1973.
6. E.J. Sondik, “The Optimal Control of Partially Observable Markov Processes Over the Infinite Horizon: Discounted Costs,” *Operations Research*, vol. 26, no. 2, pp. 282–304, 1978.
7. C.C. White III, “A Survey of Solution Techniques for the Partially Observed Markov Decision Process,” *Annals of Operations Research*, vol. 32, no. 1, pp. 215–230, 1991. DOI: 10.1007/BF02204836.
8. W.S. Lovejoy, “A Survey of Algorithmic Methods for Partially Observed Markov Decision Processes,” *Annals of Operations Research*, vol. 28, no. 1, pp. 47–65, 1991. DOI: 10.1007/BF02055574.
9. G.E. Monahan, “A Survey of Partially Observable Markov Decision Processes: Theory, Models, and Algorithms,” *Management Science*, vol. 28, no. 1, pp. 1–16, 1982.
10. L.P. Kaelbling, M.L. Littman, and A.R. Cassandra, “Planning and Acting in Partially Observable Stochastic Domains,” *Artificial Intelligence*, vol. 101, no. 1–2, pp. 99–134, 1998. DOI: 10.1016/S0004-3702(98)00023-X.
11. C. Papadimitriou and J. Tsitsiklis, “The Complexity of Markov Decision Processes,” *Mathematics of Operation Research*, vol. 12, no. 3, pp. 441–450, 1987. DOI: 10.1287/moor.12.3.441.
12. O. Madani, S. Hanks, and A. Condon, “On the Undecidability of Probabilistic Planning and Related Stochastic Optimization Problems,” *Artificial Intelligence*, vol. 147, no. 1-2, pp. 5–34, 2003. DOI: 10.1016/S0004-3702(02)00378-8.

13. M. Hauskrecht, “Value-Function Approximations for Partially Observable Markov Decision Processes,” *Journal of Artificial Intelligence Research*, vol. 13, pp. 33–94, 2000. doi: 10.1613/jair.678.
14. G. Shani, J. Pineau, and R. Kaplow, “A Survey of Point-Based POMDP Solvers,” *Autonomous Agents and Multi-Agent Systems*, pp. 1–51, 2012. doi: 10.1007/s10458-012-9200-2.
15. J. Pineau, G.J. Gordon, and S. Thrun, “Anytime Point-Based Approximations for Large POMDPs,” *Journal of Artificial Intelligence Research*, vol. 27, pp. 335–380, 2006. doi: 10.1613/jair.2078.
16. T. Smith and R.G. Simmons, “Heuristic Search Value Iteration for POMDPs,” in *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2004.
17. ———, “Point-Based POMDP Algorithms: Improved Analysis and Implementation,” in *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2005.
18. H. Kurniawati, D. Hsu, and W.S. Lee, “SARSOP: Efficient Point-Based POMDP Planning by Approximating Optimally Reachable Belief Spaces,” in *Robotics: Science and Systems*, 2008.
19. M.T.J. Spaan and N.A. Vlassis, “Perseus: Randomized Point-Based Value Iteration for POMDPs,” *Journal of Artificial Intelligence Research*, vol. 24, pp. 195–220, 2005. doi: 10.1613/jair.1659.
20. P. Poupart and C. Boutilier, “Bounded Finite State Controllers,” in *Advances in Neural Information Processing Systems (NIPS)*, 2003.
21. C. Amato, D.S. Bernstein, and S. Zilberstein, “Solving POMDPs Using Quadratically Constrained Linear Programs,” in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2007.
22. S. Ross, J. Pineau, S. Paquet, and B. Chaib-draa, “Online Planning Algorithms for POMDPs,” *Journal of Artificial Intelligence Research*, vol. 32, pp. 663–704, 2008. doi: 10.1613/jair.2567.
23. D. Silver and J. Veness, “Monte-Carlo Planning in Large POMDPs,” in *Advances in Neural Information Processing Systems (NIPS)*, 2010.

7

Cooperative Decision Making

Christopher Amato

Solving problems in which a group of agents must operate collaboratively in sequential environments is an important challenge. As the construction of agents (e.g., robots, sensors, software agents) becomes less costly, more agents can be deployed, but for a team to achieve its full potential, each agent must reason about the others. In this chapter, we discuss models in which agents may have uncertainty about both the state of the environment and the choices of the other agents. Agents seek to optimize a shared objective function but must develop plans of action based on a partial view of the environment. We describe modeling this problem as a decentralized partially observable Markov decision process (Dec-POMDP) and discuss the complexity and salient properties of this model. We also provide an overview of exact and approximate solution methods for Dec-POMDPs, discuss the use of communication between agents in this model, and describe notable subclasses with additional modeling assumptions and reduced complexity.

7.1 Formulation

Multiagent systems can be modeled in a centralized way using MDPs and POMDPs by requiring all agent information and decision making to be centralized at each step. However, many problems require decentralized execution. The Dec-POMDP model is an extension of the MDP and POMDP models that provides decentralized policies for each agent. In a Dec-POMDP, the dynamics of the system and the objective function depend on the actions of all agents, but each agent must make decisions based on local information. We first present the model, discuss an example problem, and outline two forms of solution representations.

7.1.1 Decentralized POMDPs

A Dec-POMDP is defined by the following:

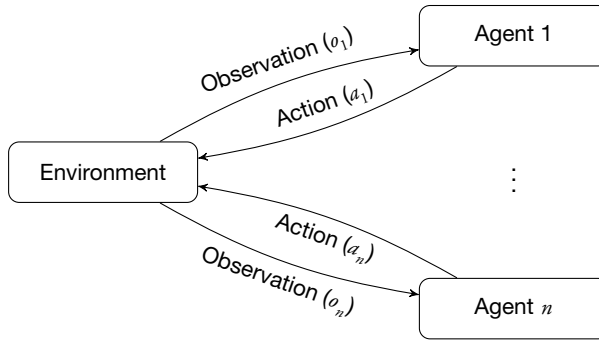


Figure 7.1 Dec-POMDP structure with n agents.

- I , a finite set of agents,
- S , a finite set of states with designated initial state distribution b_0 ,
- A_i , a finite set of actions for each agent i ,
- T , transition probability function $T(s' | s, \mathbf{a})$ that specifies the probability of transitioning from state s to s' when action $\mathbf{a}$ is taken by the agents,
- R , a reward function $R(s, \mathbf{a})$ that specifies the immediate reward for being in state s and taking the actions $\mathbf{a}$,
- Ω_i , a finite set of observations for each agent i , and
- O , observation model $O(\mathbf{o} | s', \mathbf{a})$ that specifies the probability of observing $\mathbf{o}$ in state s' given action $\mathbf{a}$.

As shown in Figure 7.1, a Dec-POMDP involves multiple agents that operate under uncertainty on the basis of different streams of observations. Like an MDP or a POMDP, a Dec-POMDP unfolds over a finite or an infinite sequence of steps. At each step, every agent chooses an action based purely on its local observations, resulting in an immediate reward for the set of agents and an observation for each individual agent. Because the state is not directly observed, it may be beneficial for each agent to remember its observation history. Unlike in POMDPs, in Dec-POMDPs, it is not always possible to calculate an estimate of the system state (a belief state) from the observation history of a single agent (as we will discuss in Section 7.2.1).

A *joint policy* is a set of policies, one for each agent in the problem. A *local policy* for an agent is a mapping from local observation histories to actions. The objective in a Dec-POMDP is to find a joint policy that maximizes expected utility. As with MDPs and POMDPs, we can define utility in different ways, such as the sum of rewards over a finite horizon or the discounted sum of rewards over an infinite horizon (Section 4.1.2).

A generalization of the Dec-POMDP formulation involves specifying independent reward functions for the various agents. If the agents are to maximize their own accumulation of reward, then the problem becomes a partially observable stochastic game

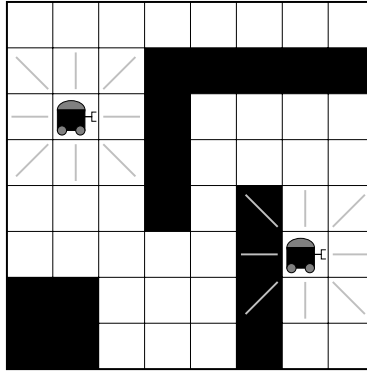


Figure 7.2 Robot navigation problem with two agents.

(POSG). Such problems require a game-theoretic treatment (similar to what was introduced in Section 3.3 for single-shot decisions) and are significantly more difficult to analyze.

7.1.2 Example Problem

One set of domains that can be modeled as Dec-POMDPs is robot navigation and exploration problems. A simple grid-based robot navigation problem is shown in Figure 7.2. The states in this problem correspond to the positions of both robots. The actions are up, down, left, right, and stay in place. The movement actions move the agent one grid cell in the desired direction with probability 0.6 or in one of the other directions or current cell with probability 0.1 each. Movements into a wall result in the robot's staying in place. Choosing to stay in place always keeps the agent in the current location. We assume perfect observation of the grid cells immediately surrounding the agent (as indicated by the gray lines in the figure). As a result, each robot can observe the wall configurations in surrounding squares but not its own actual location. The objective is for the agents to meet as quickly as possible. When both agents occupy the same square, a reward of one is received; otherwise, no reward is received. The initial state is the one shown in the figure.

There are three types of uncertainty in this problem: uncertainty about action outcomes (as in an MDP), uncertainty about sensor information (as in a POMDP), and uncertainty about the information of the other agents. Although the agents observe the surrounding grid squares and can narrow down their own possible locations, the observations usually provide no information about the choices or location of the other agent. As a result, optimal algorithms will often consider all the possible choices and locations for the other agents when generating a solution. As discussed later, centralized

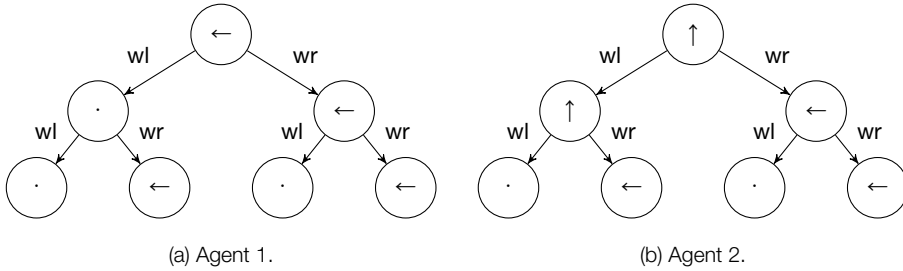


Figure 7.3 Policy tree representation for a two-agent Dec-POMDP.

belief states as used by a POMDP are no longer possible, and solving a Dec-POMDP becomes much more difficult. A solution to this problem would be for each agent to move toward a central location and, if the other agent is not seen, continue to some locations the other agent could be. If the other agent is still not seen, then the agent could move to a location that was agreed on in the solution process and wait for the other agent to arrive. The solution method would produce the policy of which movements an agent should make after seeing different observation histories, optimizing these choices over the uncertainty.

7.1.3 Solution Representations

For finite-horizon problems, local policies can be represented by policy trees. An example is shown in Figure 7.3. Such trees are similar to the policy trees for POMDPs, but now each agent possesses its own tree that is independent of the other agents' trees. To make this more concrete, we consider a simplified version of the example problem above in a 2×2 grid without obstacles. Agent 1 starts in the top right, and agent 2 starts in the bottom left. Actions are represented by arrows or the stop symbol, $\cdot$, (each agent can move in the given direction or stay where it is). Observations are labeled “wl” and “wr” for seeing a wall on the left or the right, respectively. In this representation, an agent takes the action defined at the root node and then, after seeing an observation, chooses the next action that is defined by the respective branch. This sequence continues until the action at a leaf node is executed. For example, agent 1 would first move left, and if a wall is seen on the right, then the agent would move left again. If a wall is now seen on the left, then the agent does not move on the final step. A policy tree is a record of the entire local history for an agent up to some fixed horizon. Because each tree is independent of the others, it can be executed in a decentralized manner. The resulting policies allow the agents to meet in the top left square quickly and with high probability.

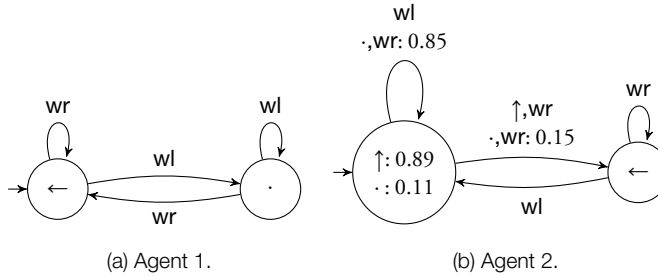


Figure 7.4 Stochastic controller representation for a two-agent Dec-POMDP.

These trees can be evaluated by summing the rewards at each step weighted by the likelihood of transitioning to a given state and observing a given set of observations. For a set of agents, the value of trees $\mathbf{q}$ while starting at state s is given recursively by

$$U(\mathbf{q}, s) = R(\mathbf{a}_{\mathbf{q}}, s) + \sum_{s', \mathbf{o}} P(s' | \mathbf{a}_{\mathbf{q}}, s) P(\mathbf{o} | \mathbf{a}_{\mathbf{q}}, s') U(\mathbf{q}_{\mathbf{o}}, s'), \quad (7.1)$$

where $\mathbf{a}_{\mathbf{q}}$ are the actions defined at the root of trees $\mathbf{q}$, and $\mathbf{q}_{\mathbf{o}}$ are the subtrees of $\mathbf{q}$ that are visited after $\mathbf{o}$ have been seen.

Although this representation is useful for finite-horizon problems, infinite-horizon problems would require trees of infinite height. Another option is to condition action selection on some internal memory state. These solutions can be represented as a set of local finite-state controllers (seen in Figure 7.4). Again, these controllers are similar to those used by POMDPs except each agent possesses its own independent controller.

The controllers operate in a similar way to the policy trees. There is a designated initial node, and following the action selection at that node, the controller transitions to the next node depending on the observation. This process continues for the infinite steps of the problem. We can also consider stochastic controllers, which choose actions and transitions stochastically, as they are able to produce higher quality solutions than deterministic controllers with the same number of nodes. Throughout this chapter, controller states will be referred to as nodes to help distinguish them from system states.

An example of two-node stochastic controllers for the 2×2 version of the example problem can be seen Figure 7.4. Agent 2 begins at node 1, moving up with probability 0.89 and staying in place with probability 0.11. If the agent stayed in place and a wall is then seen on the left (observation “wl”), on the next step, then the controller would transition to node 1, and the agent would use the same distribution of actions again. If a wall was seen on the right instead (observation “wr”), then there is a 0.85 probability that the controller will transition back to node 1 and a 0.15 probability that the controller will transition to node 2 for the next step. The resulting policy again allows the agents to

meet quickly and with high probability in the top left square. The finite-state controller allows an infinite-horizon policy to be represented compactly by remembering some aspects of the agent's history without representing the entire local history.

We can evaluate the joint policy by beginning at the initial node and transitioning through the controller according to the actions taken and observations seen. We define the probability an action a_i will be taken in node q_i by agent i to be $P(a_i | q_i)$. We also have $P(q'_i | q_i, a_i, o_i)$ represent the probability that the controller transitions to node q'_i given that the controller is currently in q_i , takes action a_i , and observes o_i . The value for starting in nodes $\mathbf{q}$ and at state s with action selection and node transition probabilities for each agent, i , is given by the following Bellman equation:

$$U(\mathbf{q}, s) = \sum_{\mathbf{a}} \left(\prod_i P(a_i | q_i) \left[R(s, \mathbf{a}) + \gamma \sum_{s', \mathbf{o}, \mathbf{q}'} P(s' | \mathbf{a}, s) O(\mathbf{o} | s', \mathbf{a}) \prod_j P(q'_j | q_j, a_j, o_j) U(\mathbf{q}', s') \right] \right). \quad (7.2)$$

Note that the values (for either the trees or controllers) can be calculated offline in order to determine a policy for each agent that can then be executed online in a decentralized manner. In fact, as we will discuss below, many algorithms consider this offline planning scenario in which the solution (trees or controllers) is generated offline in a centralized manner and then the policy is executed online in a decentralized manner.

7.2 Properties

The decentralized nature of Dec-POMDPs makes them fundamentally different from POMDPs. We explain some of these differences, discuss the complexity of the general Dec-POMDP model, and describe an extension of the concept of belief states to multiagent problems.

7.2.1 Differences with POMDPs

In a Dec-POMDP, the decisions of each agent affect all the agents in the domain, but because of the decentralized nature of the model, each agent must choose actions based solely on local information. Because each agent receives a separate observation that does not usually provide sufficient information to efficiently reason about the other agents, solving a Dec-POMDP optimally becomes difficult. Each agent may receive a different piece of information that does not allow a common state estimate or any estimate of the other agents' decisions to be calculated. For example, in the robot navigation example problem in Figure 7.2, even though each agent knows the initial location of the other, agent 1's observations (until it becomes adjacent) give it no information about agent 2's

choice of action or location. Therefore, while it may be possible to limit the possible locations the other agent may be in (such as those not in surrounding grid cells), it is usually not possible to generate an estimate of the system state. Exceptions to this are when observations provide this information (e.g., when the other agent is seen) or the policies of the other agents are known (as discussed later).

State estimates are crucial in single-agent problems because they allow the agent's history to be summarized concisely (as belief states), but they are not generally available in Dec-POMDPs. The lack of state estimates (and thus the lack of a concise sufficient statistic) requires agents to remember whole action and observation histories in order to act optimally; therefore, Dec-POMDPs cannot be transformed into belief state MDPs, and we must use a different set of tools to solve them.

7.2.2 Dec-POMDP Complexity

The difference between Dec-POMDPs and POMDPs is seen in the complexity of the finite-horizon problem. A Dec-POMDP with at least two agents is *NEXP-complete*, a category of problem that may require doubly exponential time in practice. This complexity is in contrast to the MDP (P-complete) and the POMDP (PSPACE-complete). As in solving an infinite-horizon POMDP, optimally solving an infinite-horizon Dec-POMDP is *undecidable* because it may require infinite resources (infinitely sized controllers), but ϵ -optimal solutions can be found with finite time and memory. These complexity differences show that introducing multiple decentralized agents causes Dec-POMDPs to be significantly more difficult to solve than POMDPs. The intuition behind this complexity is that agents must consider the possible choices of all other agents in addition to the state and action uncertainty present in order to produce an optimal policy.

7.2.3 Generalized Belief States

As mentioned above, from an agent's perspective, not only is there uncertainty about the state, but there may also be uncertainty about the policies of the other agents. If we consider the possible policies of the other agents as part of the state of the system, then we can form a *generalized belief state* (sometimes called a multiagent belief state). Agents can also consider the generalized belief space that includes all possible distributions over states of the system and policies of the other agents. In a two-agent situation, the value of an agent's policy p at a given generalized belief state b_G is given by

$$U(p, b_G) = \sum_{q,s} b_G(s, q) U(p, q, s), \quad (7.3)$$

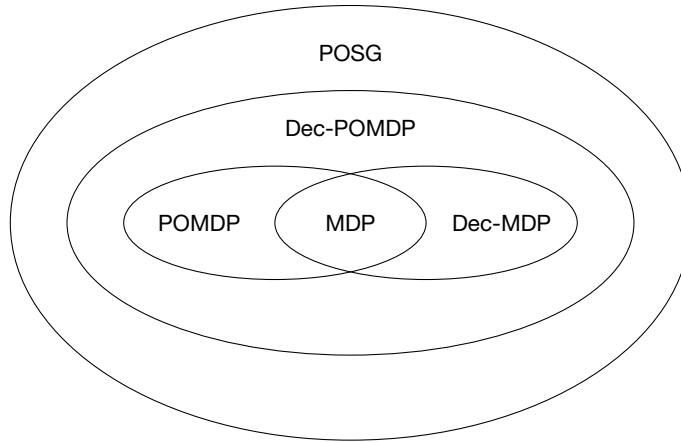


Figure 7.5 Subclasses and superclasses of Dec-POMDPs.

where q represents a policy of the other agent. If all other agents have known policies, then the generalized belief state is the same as a POMDP belief state, and we can solve the Dec-POMDP for the remaining agent as a POMDP (the other agents can be thought of as part of the environment). Unfortunately, when we solve a Dec-POMDP, probability distributions for other agent policies are generally not known. As a result, the generalized belief state cannot usually be calculated. Nevertheless, the idea of the possible policies of the other agents and the generalized belief space can be used in the decision-making process.

7.3 Notable Subclasses

Because of the high worst-case complexity of Dec-POMDPs, a number of Dec-POMDP subclasses have been explored that may be more tractable theoretically or in practice. This section discusses a few of these, including the Dec-MDP, network distributed POMDP (ND-POMDP), and multiagent MDP (MMDP). The relationships among Dec-POMDPs, POSGs, POMDPs, MDPs, and Dec-MDPs are shown in Figure 7.5.

7.3.1 Dec-MDPs

A *decentralized Markov decision process* (Dec-MDP) is a Dec-POMDP with joint full observability. In other words, if the observations of all the agents are combined, then the state of the environment is known exactly. A common example of a Dec-MDP is a problem in which the state consists of the locations of a set of robots and each agent observes its own location perfectly. Therefore, if all these observations are combined, then the locations of all robots would be known. Note that (perhaps counterintuitively)

Dec-MDPs do include observations for each agent that are often noisy indicators of the state. The complexity of Dec-MDPs is the same as Dec-POMDPs; although the true state may be known if observations are shared, this sharing does not take place.

We now discuss factorization in the context of Dec-MDPs, but similar factorization can be done in full Dec-POMDPs. A *factored n -agent Dec-MDP* is a Dec-MDP in which the world state can be factored into $n + 1$ components, $S = S_0 \times S_1 \times \dots \times S_n$. The states in S_i are the *local states* associated with agent i . The S_0 component is a property of the environment and is not affected by any agent actions (and is sometimes omitted). For example, S_0 may be the location of a target in a target-tracking scenario. Similarly, an agent's local state, S_i , might consist of its location in a grid. A factored, n -agent Dec-MDP is said to be *locally fully observable* if each agent fully observes its own state component.

A factored, n -agent Dec-MDP is said to be *transition independent* if the state transition probabilities factorize as follows:

$$T(s' | s, \mathbf{a}) = T_0(s'_0 | s_0) \prod_i T_i(s'_i | s_i, a_i). \quad (7.4)$$

Here, $T_i(s'_i | s_i, a_i)$ represents the probability that the local state of agent i transitions from s_i to s'_i after executing action a_i . The unaffected state transition probability is denoted $T_0(s'_0 | s_0)$. The robot navigation problem is transition independent if the robots never affect each other (i.e., they do not bump into each other when moving and can share the same grid cell).

A factored, n -agent Dec-MDP is said to be *observation independent* if the observation probabilities factorize as follows:

$$O(\mathbf{o} | s, \mathbf{a}) = \prod_i O_i(o_i | s_i, a_i). \quad (7.5)$$

In the equation above, $O_i(o_i | s_i, a_i)$ represents the probability that agent i receives observation o_i in state s_i after executing action a_i . If the robots in the navigation problem cannot observe each other (due to working in different locations or lack of sensors), then the problem becomes observation independent.

A factored, n -agent Dec-MDP is said to be *reward independent* if

$$R(s, \mathbf{a}) = f(R_1(s_1, a_1), \dots, R_n(s_n, a_n)), \quad (7.6)$$

with the constraint that f is some monotonically non-decreasing function. Such a function has the property that $x_i \leq x'_i$ if and only if

$$f(x_1, \dots, x_i, \dots, x_n) \leq f(x_1, \dots, x'_i, \dots, x_n). \quad (7.7)$$

Table 7.1 Complexity of finite-horizon Dec-MDP subclasses.

Independence	Complexity
Transitions, observations and rewards	P-complete
Transitions and observations	NP-complete
Any other subset	NEXP-complete

Under this assumption, the global reward is maximized by maximizing local rewards. Additive local rewards are often used in reward independent models, where

$$R(s, \mathbf{a}) = R_0(s_0) + \sum_i R_i(s_i, a_i). \quad (7.8)$$

Table 7.1 shows the complexity of different subclasses of Dec-MDPs. The simplest case results from having independent transitions, observations, and rewards. It is straightforward to see that, in this case, the problem can be decomposed into n separate MDPs, and their solution can then be combined. When only the transitions and observations are independent, the problem becomes NP-complete. Intuitively, the NP-completeness is because the other agents' policies do not affect an agent's state (only the reward attained at the set of local states). Because independent transitions and observations imply local full observability, an agent's observation history does not provide any additional information about its own state—it is already known. Similarly, an agent's observation history does not provide any additional information about the other agents' states because they are independent. As a result, optimal policies become mappings from local states to actions instead of mappings from observation histories (or local state histories as local states are locally fully observable in this case) to actions. All other combinations of independent transitions, observations, and rewards do not reduce the complexity of the problem, leaving it NEXP-complete in the worst case.

7.3.2 ND-POMDPs

A *networked distributed POMDP* (ND-POMDP) is a Dec-POMDP with transition and observation independence and a special reward structure. The reward structure is represented by a coordination graph or *hypergraph*. A hypergraph is a generalization of a graph in which an edge can connect any number of nodes. The nodes in the ND-POMDP hypergraph correspond to the various agents. The edges relate to interactions between the agents in the reward function. An ND-POMDP associates with each edge j in the hypergraph a reward component R_j that depends on the state and action components to which the edge connects. The reward function in an ND-POMDP is simply the sum of the reward components associated with the edges. This fact allows

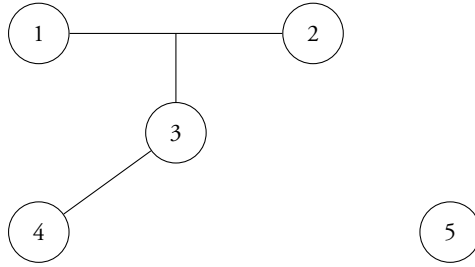


Figure 7.6 Example ND-POMDP structure.

the value function to be factorable in the same way, and solution methods can take advantage of this additional structure.

Figure 7.6 shows an example ND-POMDP structure with five agents. There are three hyper edges: one involving agents 1, 2, and 3; another involving agents 3 and 4; and another involving agent 5 on its own. The reward function decomposes as follows:

$$R_{123}(s_1, s_2, s_3, a_1, a_2, a_3) + R_{34}(s_3, s_4, a_3, a_4) + R_5(s_5, a_5). \quad (7.9)$$

Motivating domains for ND-POMDPs are typically sensor network and target tracking problems.

The ND-POMDP model is similar to the transition and observation independent Dec-MDP model, but it does not make the joint full observability assumption. Even if all observations are shared, the true state of the world may not be known. Furthermore, even with factored transitions and observations, a policy in an ND-POMDP is a mapping from observation histories to actions, unlike the transition and observation Dec-MDP case in which policies are mappings from local states to actions. The worst-case complexity remains the same as a full Dec-POMDP (NEXP-complete), but algorithms for ND-POMDPs are typically much more scalable in the number of agents. Scalability can increase as the hypergraph becomes less connected.

7.3.3 MMDPs

Another notable subclass is the *multiagent Markov decision process* (MMDP). In an MMDP, each agent is able to observe the true state, making the problem fully observable. Because each agent is able to observe the true state, an MMDP can be solved as an MDP (using some coordination mechanisms to ensure policies are consistent with each other) in polynomial time. An example based on the robot navigation problem above for the MMDP case would assume that each robot knows the location of the other robots at each step of the problem. The relationships among MMDPs, Dec-POMDPs, and single-agent models are shown in Figure 7.7.

		Observability	
		Perfect	Imperfect
Number of Agents	Multiple	MMDP	Dec-POMDP
	Single	MDP	POMDP

Figure 7.7 The relationship between Dec-POMDPs and other models.

7.4 Exact Solution Methods

This section discusses optimal algorithms for finite-horizon problems and ϵ -optimal algorithms for infinite-horizon problems. Finite-horizon methods can be used to solve infinite-horizon problems within any ϵ by using a horizon that is large enough to cause further action selection to contribute little to the overall value. POMDP methods are often scalable enough to produce such solutions, but Dec-POMDP methods are not. As a result, specific infinite-horizon methods for Dec-POMDPs, which use finite-state controllers as solution representations, are discussed. Most solution methods assume the problem is solved centrally offline to produce policies that can be executed online in a decentralized manner. These methods can also produce solutions in a decentralized manner with proper coordination mechanisms for the given algorithm.

7.4.1 Dynamic Programming

Algorithm 7.1 is a dynamic programming algorithm for optimally solving a finite-horizon Dec-POMDP. This approach constructs a set of trees for each agent (represented as π) from the last step until the first. At each step, the algorithm exhaustively generates all next step policies, evaluating them, and then pruning policies that are provably suboptimal for each agent i in turn until no more trees can be removed. This generation, evaluation, and pruning continues until the desired horizon T is reached. Dynamic programming in Dec-POMDPs is similar to the value iteration approach for POMDPs, except generalized belief states are used, resulting in a more complicated pruning step.

In the dynamic programming algorithm, a set of T -step policy trees, one for each agent, is generated from the bottom up. More precisely, on the last step of the problem, each agent will perform just a single action, which can be represented as a one-step policy tree. All possible actions for each agent are considered, and each combination of these one-step trees is evaluated at each state of the problem by using Equation (7.1).

Algorithm 7.1 Dynamic programming for Dec-POMDPs

```

1: function DEC_DYNAMIC_PROGRAMMING( $T$ )
2:    $t \leftarrow 0$ 
3:    $\pi_t \leftarrow \emptyset$ 
4:   repeat
5:      $\pi_{t+1} \leftarrow \text{ExhaustiveBackup}(\pi_t)$ 
6:     Compute  $V^{\pi_{t+1}}$ 
7:     repeat
8:        $\hat{\pi}_{t+1} \leftarrow \pi_{t+1}$ 
9:       for  $i \in I$ 
10:         $\hat{\pi}_{t+1} \leftarrow \text{Prune}(\hat{\pi}_{t+1}, i)$ 
11:        Compute  $V^{\hat{\pi}_{t+1}}$ 
12:     until  $|\pi_t| = |\hat{\pi}_t|$ 
13:      $t \leftarrow t + 1$ 
14:   until  $t = T$ 
15:   return  $\pi_t$ 

```

Any action that has lower value than some other action for all states and possible actions of the other agents (the generalized belief space) is then pruned. All two-step policies are then generated for each agent by an *exhaustive backup* of the current trees. That is, for each action and each resulting observation, some one-step tree is chosen. If an agent has $|Q_i|$ one-step trees, $|A_i|$ actions, and $|\Omega_i|$ observations, then there will be $|A_i||Q_i|^{|\Omega_i|}$ two-step trees. After this exhaustive backup of next step trees is completed for each agent, pruning is again used to reduce their number. This process of backing up trees and pruning continues until horizon T is reached.

The resulting set of trees will contain an optimal solution for horizon T and any initial state of the system. This is because we considered all possible trees at each step and removed only those that were not useful no matter what policies the other agents chose at that step. This conservative pruning of trees ensures that we can safely remove trees that will be suboptimal at a given step.

The linear program used to determine whether a tree can be pruned can be represented as follows. For agent i 's given tree q_i and variables ϵ and $x(\mathbf{q}_{-i}, s)$, maximize ϵ , given

$$\sum_{\mathbf{q}_{-i}, s} x(\mathbf{q}_{-i}, s) U(\hat{q}_i, \mathbf{q}_{-i}, s) + \epsilon \leq \sum_{\mathbf{q}_{-i}, s} x(\mathbf{q}_{-i}, s) U(q_i, \mathbf{q}_{-i}, s) \quad \forall \hat{q}_i \quad (7.10)$$

$$\sum_{\mathbf{q}_{-i}, s} x(\mathbf{q}_{-i}, s) = 1 \text{ and } x(\mathbf{q}_{-i}, s) \geq 0 \quad \forall \mathbf{q}_{-i}, s. \quad (7.11)$$

This linear program determines whether agent i tree, q_i is dominated by comparing its value to other trees for that agent, $\hat{q}_i$. The variable $x(\mathbf{q}_{-i}, s)$ is a distribution over

trees of the other agents and system states (the generalized belief state). We maximize ϵ while ensuring the variable x that represents the generalized belief state remains a proper probability distribution and test to see whether there is some distribution of trees that has higher or equal value for all states and policies of the other agents. Because there is always an alternative with at least equal value, regardless of system state and other agent policies, a tree q_i can be pruned if ϵ is non-positive.

The test for dominance is used for trees of a given horizon, ensuring that we consider all possible policies for a given horizon and remove those that are not useful no matter what policies are chosen by the other agents. If policies are removed, then the generalized belief space becomes smaller for the other agents (due to the removal of a possible policy to consider), and more policies may be able to be pruned. Thus, we can keep testing for dominated policies for each agent until no agent is able to prune any further policies. Lines 7–11 in Algorithm 7.1 show that pruning continues for all agents while any agent is able to remove any tree.

Unlike value iteration for POMDPs, the policy trees must be retained because it is no longer possible to recover the policy from the value function. Even with one-step lookahead in the POMDP case, we must calculate the belief state after an action is chosen. Because it is not possible to calculate a belief state in the Dec-POMDP case, and because the actions must be chosen based on local information, the optimal value function is not sufficient to generate a Dec-POMDP policy.

7.4.2 Heuristic Search

Instead of computing the policy trees from the bottom up, as is done by dynamic programming methods, we can construct the trees from the top down starting from a known initial state. This is the approach of multiagent A* (MAA*), which is an optimal algorithm built on heuristic search techniques. The search is conducted by using an upper bound on the value of partially defined policies (using POMDP or MDP solutions) and then choosing the partial joint policy to expand in a best-first ordering. Actions are then added, and an upper bound on this new partial joint policy is found. Again, the highest-valued partial joint policy is chosen, and another action choice is fixed. This process continues until a fully defined joint policy is found that has value higher than any of the partial joint policies. Algorithm 7.2 outlines the approach.

We can grow a joint policy in a top-down fashion by first considering the possible actions each agent can take to begin its policies. MAA* considers all possible combinations of actions that could be taken at that step and constructs a search tree that has these combinations as separate search nodes. In the worst case, a search tree could be constructed that contains all possible policies for each agent by considering all possible combinations of actions at the first step, followed by all possible combinations of actions for each observation at the second step, and so on. Because many of these policies may

Algorithm 7.2 Multiagent A*

```

1: function MULTIAGENTA*( $T, b_0$ )
2:    $\underline{V} \leftarrow -\infty$ 
3:    $\underline{L} \leftarrow \times_i A_i$ 
4:   repeat
5:      $\delta \leftarrow \text{select}(\underline{L})$ 
6:      $\Delta' \leftarrow \text{expand}(\delta)$ 
7:      $\Delta^T \leftarrow \text{fullPols}(\Delta')$ 
8:      $\pi \leftarrow \text{bestFullPol}(\Delta^T)$ 
9:      $v \leftarrow \text{valueOf}(\pi)$ 
10:    if  $v > \underline{V}$ 
11:       $\pi^* \leftarrow \pi$ 
12:       $\underline{V} \leftarrow v$ 
13:       $\text{prune}(\underline{L}, \underline{V})$ 
14:       $\Delta' \leftarrow \Delta' \setminus \Delta^T$ 
15:       $\underline{L} \leftarrow \underline{L} \setminus \delta \cup \Delta'$ 
16:    until  $\underline{L}$  is empty
17:  return  $\pi^*$ 

```

be suboptimal, a more intelligent approach for determining which choices to make is desired.

To assist in choosing better actions, MAA* incorporates a heuristic value for continuing execution for a given number of steps after a partial policy has been executed. That is, using a search based on the A* heuristic search method, we can estimate the value of a set of horizon T policies from a set of horizon t trees using the value of those trees up to horizon t calculated with Equation (7.1) and then some heuristic value for the value of continuing to horizon T .

More formally, we will call some set of horizon $t < T$ policy trees $\mathbf{q}$ a *partial policy* and a set of policies Δ^{T-t} a *completion policy*. A completion policy consists of appending $T - t$ policies to each leaf (last action) in the partial policy. Given a partial policy and a completion policy, we could evaluate them at a state s as follows:

$$U(\{\mathbf{q}^t, \Delta^{T-t}\}, s) = U(\mathbf{q}^t, s) + U(\Delta^{T-t} \mid \mathbf{q}^t, s). \quad (7.12)$$

Rather than explicitly considering completion policies to append, we can instead estimate the value a completion policy would produce. Therefore, we can produce an estimated value $\hat{U}$ that a partial policy has for the full horizon T as

$$\hat{U}(\mathbf{q}^t, s) = U(\mathbf{q}^t, s) + \hat{U}_{\mathbf{q}^t, s}^{T-t}, \quad (7.13)$$

where $\hat{V}_{\mathbf{q}^t,s}^{T-t}$ is the estimate for the value of continuing until horizon T after executing $\mathbf{q}$ starting at s .

There are many ways we can calculate $V_{\mathbf{q}^t,s}^{T-t}$, but to ensure an optimal policy is produced, we require the estimated value to be at least as high as the optimal value of continuing ($\hat{V} \geq V^*$). MAA* considers relaxing problem assumptions by using MDP or POMDP policies to produce the heuristic values. $V_{\text{POMDP}}^*(b)$ can be defined as the optimal value of a POMDP policy starting at belief b (i.e., a centralized solution in which all observations for all agents are known and actions can be chosen based on this centralized information). Similarly, $V_{\text{MDP}}^*(s)$ can be defined as the optimal value of an MDP policy starting at state s (i.e., a centralized policy assuming the state of the problem can be seen by all agents for the remainder of the problem). It can be shown that $V_{\text{Dec-POMDP}}^* \leq V_{\text{POMDP}}^* \leq V_{\text{MDP}}^*$, which intuitively holds because policies are less constrained as more information is available to the agents. MAA* then evaluates a partial policy by evaluating the policy until its given horizon t and then assumes either the belief state (in the V_{POMDP}^* case) or the state (in the V_{MDP}^* case) is known to the agents thereafter starting from the leaf nodes of $\mathbf{q}^t$.

The search in MAA* then chooses to grow the partial policy with the highest heuristic value. Growing a policy appends actions for each possible observation after the leaves of the current policy. The expansion of this search node adds an exponential number of nodes to the search tree at each step. Each of these new policies is then evaluated to produce an updated heuristic value.

A lower bound on the value of an optimal joint policy is also maintained. If growing a policy results in a horizon T policy, then the value of this policy is compared with the lower bound, updating it if the value of this policy is greater. Subsequently, any partial policy with estimated value less than the lower bound can then be pruned. This pruning can occur because the estimated value of a policy is an upper bound on its true value, indicating that it will never have value higher than the policy that produced the lower bound. The search completes when there are no more partial policies to expand. In this case, a set of $\mathbf{q}^T$ policies has been generated with higher value than the heuristic values of any partial policy.

We now describe Algorithm 7.2 in more detail. A lower bound value, $\underline{V}$, which represents the value of the best known joint policy, is initialized to negative infinity. An open list L , which represents the partial policies that are available to be expanded, is initialized to all joint actions for the agents. At each step, the partial joint policy (node in the search tree) with the highest estimated value $\hat{V}$ is selected. This partial policy is then expanded, generating all next step policies for that joint policy (all children in the search tree). This set is called Δ' . The set of full horizon T joint policies is now gathered into Δ^T , each of which is evaluated, and the one with the highest value v is chosen. If this value is higher than the value of the best known full policy, then the

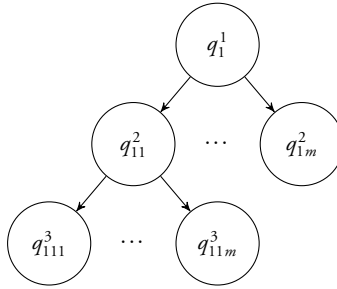


Figure 7.8 An example search tree in MAA*.

lower bound value and pointer to best policy are updated. Also, any partial policy in the open list that has estimated upper bound value (using the Q_{MDP} or Q_{POMDP} heuristics) lower than $\underline{V}$ is pruned. The full trees are removed from the set of expanded nodes, and the selected node is removed from the open list. The remaining expanded nodes are then added to the open list. This algorithm continues until the open list is empty and returns an optimal joint policy π^* .

An example of this search is shown in Figure 7.8. Each search node represents a joint policy for a given horizon. Subscripts represent the indices of the search node and each of the ancestor nodes in the search tree, whereas superscripts represent the horizon of the joint policy. Ancestors are indexed to signify that partial policies are shared for the appropriate horizon. Note that m represents the number of possible next-step joint policies, which is $|A_{\max}|^n |\Omega_{\max}|$, given the largest action and observation sets among the n agents $A_{\max}, \Omega_{\max}$. The available partial policies (the open list) are represented by the leaves of the tree, whereas the nodes that have already been expanded are shown as internal nodes.

7.4.3 Policy Iteration

Because the infinite-horizon problem is undecidable, it may not be possible to generate a solution with the exact optimal value. As a consequence, methods focus on producing solutions within ϵ of the optimal value.

The policy iteration approach for Dec-POMDPs is similar to the finite-horizon dynamic programming algorithm, except finite-state controllers are used as policy representations (like the policy iteration approach for POMDPs). Starting from an initial controller for each agent, nodes are added at each step by using an exhaustive backup to produce any possible next-step policy for each agent. Pruning can then be completed to remove a node in an agent's controller if it has lower value than beginning in another node for all states of the system and all possible controllers of the other agents.

Algorithm 7.3 Policy iteration for Dec-POMDPs

```

1: function DEC-POLICY-ITERATION( $\pi_0, \epsilon$ )
2:    $t \leftarrow 0$ 
3:   repeat
4:      $\pi_{t+1} \leftarrow \text{ExhaustiveBackup}(\pi_t)$ 
5:     Compute  $V^{\pi_{t+1}}$ 
6:     repeat
7:        $\hat{\pi}_{t+1} \leftarrow \pi_{t+1}$ 
8:       for  $i \in I$ 
9:          $\hat{\pi}_{t+1} \leftarrow \text{Prune}(\hat{\pi}_{t+1}, i)$ 
10:      UpdateController( $\hat{\pi}_{t+1}, i$ )
11:      Compute  $V^{\hat{\pi}_{t+1}}$ 
12:   until  $|\pi_t| = |\hat{\pi}_t|$ 
13:    $t \leftarrow t + 1$ 
14:   until  $\frac{\gamma^{t+1} |R_{\max}|}{1-\gamma} \leq \epsilon$ 
15:   return  $\pi_t$ 

```

These exhaustive backups and pruning steps continue until the solution is provably within ϵ of an optimal solution. This algorithm can produce an ϵ -optimal policy in a finite number of steps. The details of policy iteration follow.

The policy iteration algorithm is shown in Algorithm 7.3. The input is an initial joint controller, π_0 , and a parameter, ϵ . At each step, evaluation, backup, and pruning occur. The controller is evaluated using Equation (7.2). Next, an exhaustive backup is performed to add nodes to the local controllers. An exhaustive backup adds nodes to the local controllers for all agents at once. Similar to the finite-horizon case, for each agent i , $|A_i||Q_i|^{|\Omega_i|}$ nodes are added to the local controller, one for each one-step policy. Note that repeated application of exhaustive backups amounts to a brute force search in the space of deterministic policies. Continuing these exhaustive backups converges to optimality but is obviously quite inefficient.

To increase the efficiency of the algorithm, pruning takes place. Recall that planning takes place offline, so the controllers for each agent are known at each step, but agents will not know which node of their controller any of the other agents will be in during execution. As a result, pruning must be completed over the generalized belief space (using a linear program that is similar to that described for finite-horizon dynamic programming). That is, a node for an agent's controller can only be pruned if there is some combination of nodes that has higher value for all states of the system and at all nodes of the other agents' controllers. If this condition holds, then edges to the removed node are redirected to the dominating nodes. Because a node may be dominated by a distribution of other nodes, the resulting transitions may be stochastic rather than

deterministic. The updated controller is evaluated, and pruning continues until no agent can remove any further nodes.

In contrast to the single-agent case, there is no Bellman residual for testing convergence to ϵ -optimality. We resort to a simpler test based on the discount rate and the number of iterations so far. Let $|R_{\max}|$ be the largest absolute value of an immediate reward possible in the Dec-POMDP. The algorithm terminates after iteration t if $\gamma^{t+1}|R_{\max}|/(1-\gamma) \leq \epsilon$. At this point, due to discounting, the value of any policy after step t is less than ϵ .

7.5 Approximate Solution Methods

In this section, we discuss approximate dynamic programming methods for finite-horizon problems and fixed-size controller-based methods for infinite-horizon problems. The algorithms in this section do not possess error bounds.

7.5.1 Memory-Bounded Dynamic Programming

The major limitation of dynamic programming approaches is the explosion of memory and time requirements as the horizon grows. This explosion occurs because each step requires generating and evaluating all joint policy trees (sets of policy trees for each agent) before performing the pruning step. Approximate dynamic programming techniques can mitigate this problem by keeping a fixed number of policy trees for each agent at each step, using a parameter called *MaxTrees*. This approach, called *memory-bounded dynamic programming* (MBDP), is outlined in Algorithm 7.4.

MBDP merges top-down (heuristic search) and bottom-up (dynamic programming) approaches by using heuristics (referred to as H in the algorithm) to choose top-down policies for each agent up to a given horizon. That is, dynamic programming proceeds as before, but after each backup, *MaxTrees* belief states are generated using heuristics to perform top-down sampling until the current step of dynamic programming is reached ($T - t$ if dynamic programming is at step t). It is then assumed that the resulting belief states are revealed to the agents, and only the trees that have the highest value at these belief states are retained. During execution, the belief state will not truly be revealed to the agents, but the hope is that high-valued decentralized policies can still be produced using this strategy. MBDP is conducted in an iterative fashion similar to traditional dynamic programming. For example, in a problem of horizon T , heuristic policies can be used for the first $T - 1$ steps, and dynamic programming can find the best one-step trees (actions) for the resulting beliefs. Then, heuristic policies can be used for the first $T - 2$ steps, and the *MaxTrees* one-step trees can be built up to two steps by dynamic programming. This process continues until *MaxTrees* horizon T trees have been constructed. Heuristics that have been used include MDP and random policies.

Algorithm 7.4 Memory-bounded dynamic programming (MBDP)

```

1: function MBDP( $MaxTrees, T, H$ )
2:    $t \leftarrow 0$ 
3:    $\pi_t \leftarrow \emptyset$ 
4:   repeat
5:      $\pi_{t+1} \leftarrow \text{ExhaustiveBackup}(\pi_t)$ 
6:     Compute  $V^{\pi_{t+1}}$ 
7:      $\hat{\pi}_{t+1} \leftarrow \emptyset$ 
8:     for  $k \in MaxTrees$ 
9:        $b_k \leftarrow \text{GenerateBelief}(H, T - t - 1)$ 
10:       $\hat{\pi}_{t+1} \leftarrow \hat{\pi}_{t+1} \cup \arg \max_{\pi_{t+1}} V^{\pi_{t+1}}(b_k)$ 
11:     $t \leftarrow t + 1$ 
12:     $\pi_{t+1} \leftarrow \hat{\pi}_{t+1}$ 
13:  until  $t = T$ 
14:  return  $\pi_t$ 

```

Because only a fixed number of trees is retained at each step, the result is a suboptimal but much more scalable algorithm. In fact, because the number of policies retained at each step is bounded by $MaxTrees$, MBDP has time and space complexity linear in the horizon.

7.5.2 Joint Equilibrium Search

As an alternative to MBDP-based approaches, a method called joint equilibrium search for policies (JESP) utilizes alternating best response. JESP is shown in Algorithm 7.5. Initial policies are generated for all agents and then all but one are held fixed. The remaining agent can then calculate a best response (local optimum) to the fixed policies. This agent's policy then becomes fixed, and the next agent calculates a best response. This process continues until no agents change their policies. The result is a policy that is only locally optimal, but it may be high valued. JESP can be made more efficient by incorporating dynamic programming in the policy generation. Note that JESP can be thought of as finding a Nash equilibrium (as discussed in Section 3.3) in the cooperative game represented as the Dec-POMDP.

7.6 Communication

Communication can be implicitly represented using observations, but more explicit representations of communication have also been developed. Free and instantaneous communication is equivalent to centralization because all agents can have access to all observations at each step. When communication is delayed or has a cost, agents

Algorithm 7.5 Joint equilibrium search for strategies (JESP) without DP

```

1: function JESP( $\pi$ )
2:    $k = 0$ 
3:    $\pi^k \leftarrow \pi$ 
4:   repeat
5:      $\pi^{k+1} \leftarrow \pi^k$ 
6:      $k \leftarrow k + 1$ 
7:     for  $i \in I$ 
8:       Compute  $V^{\pi^k}$ 
9:        $\pi^k(i) \leftarrow \text{BestResponse}(\pi^k, -i)$ 
10:  until  $\pi^k = \pi^{k-1}$ 
11:  return  $\pi^k$ 

```

must reason about what and when to communicate. The complexity classes of the Dec-POMDP model with different types of observability and communication are shown in Table 7.2.

One extension of the general Dec-POMDP model to explicitly include communication is the decentralized partially observable Markov decision process with communication (Dec-POMDP-Com). A Dec-POMDP-Com is a Dec-POMDP, where each agent can send a message at each step. The reward at each step is a function of the joint state, joint action, and set of messages sent by the agents.

Producing an optimal solution of a Dec-POMDP-Com has the same complexity as solving the general Dec-POMDP model (NEXP-complete), and similar algorithms can be used to solve it as well. It can be shown that the optimal value of communicating an agent's history since the last communication will have value at least as high as any other set of possible messages, assuming fixed communication costs. As a result, many communication approaches assume that observation histories are used during communication. Many methods assume communication decisions are made by each agent separately, but some models assume agents can force all others to send their observation histories (allowing a POMDP belief state to be calculated).

A natural approach to solving a Dec-POMDP with communication is to produce a centralized (POMDP) plan and communicate when the observations received by an agent would cause it to choose a different action from the one prescribed by the centralized plan. This approach can be thought of as the agents agreeing on a policy before execution, and when it is noticed (by an agent's local observations) that this policy could be improved, communication takes place. This approach does not explicitly consider communication cost or delay but can limit the number of times communication takes place.

Table 7.2 The effect of communication on model complexity with different observability.

Observability	General Communication	Free Communication
Full	MMDP (P)	MMDP (P)
Joint Full	Dec-MDP (NEXP)	MMDP (P)
Partial	Dec-POMDP (NEXP)	MPOMDP (PSPACE)

7.7 Summary

- Dec-POMDPs can represent cooperative multiagent problems with action outcome uncertainty, observation uncertainty, and costly, lossy, or no communication.
- Like MDPs and POMDPs, Dec-POMDPs model sequential problems using a decision-theoretic approach that utilizes probabilities for uncertainties and values for outcomes.
- Unlike the single-agent models, each agent must make choices based solely on its own observation history.
- Solving finite-horizon Dec-POMDPs is NEXP-complete.
- Many subclasses of Dec-POMDPs are more efficient in theory or practice.
- Algorithms have been developed that can produce optimal or ϵ -optimal solutions for finite and infinite-horizon Dec-POMDPs.
- More scalable approximate algorithms have also been developed.
- Communication can also be explicitly modeled to assist in determining when and what to communicate to improve performance.

7.8 Further Reading

General overviews of Dec-POMDPs with additional algorithms and models are provided by Seuken and Zilberstein [1], Oliehoek [2], and Goldman and Zilberstein [3]. The complexity of the general Dec-POMDP was proven by Bernstein et al. [4]. A model similar to a Dec-POMDP is the multiagent team decision problem (MTDP) [5]. Finite-horizon dynamic programming is presented by Hansen, Bernstein, and Zilberstein [6], and infinite-horizon policy iteration is described by Bernstein et al. [7]. Multiagent A* is presented by Szer, Charpillet, and Zilberstein [8].

Dec-MDPs with independent transitions and observations were first discussed and solved by Becker et al. [9]. Dec-MDPs with event-driven interactions, considering limited transition dependence, were also discussed [10]. Allen and Zilberstein discussed a number of different modeling assumptions and resulting complexity [11]. ND-POMDPs were first proposed by Nair et al. [12], and MMDPs are presented by Boutilier [13]. Approximate finite-horizon algorithms MBDP and JESP are de-

scribed by Seuken and Zilberstein [14] and Nair et al. [15], respectively. Approximate infinite-horizon algorithms can be found in the literature [16]–[18].

The Dec-POMDP-Com model was presented by Goldman and Zilberstein [3]. The idea of using a centralized policy as a basis for communication was described by Roth, Simmons, and Veloso [19]. Forced synchronizing communication was discussed by Nair and Tambe [20].

Optimal finite-horizon algorithms have been improved in a few directions. Pruning can be used while conducting the backup to limit the subtrees that need to be considered [21]. Other work has also compressed policies rather than agent histories, improving the efficiency of the linear program used for pruning [22]. MAA* has been improved in several ways, including the incorporation of new heuristics and improved search [23]–[25]. Recently, two approaches have been developed for generating more concise sufficient statistics for offline planning, proving the sufficiency of distributions of states and joint histories [26] and converting Dec-POMDPs into POMDPs using decentralizable policies [27].

Approximate algorithms have also seen additional improvements. A number of approaches have improved on MBDP by compressing observations [28], replacing the exhaustive backup with a branch-and-bound search in the space of joint policy trees [29], as well as constraint optimization [30] and linear programming [31] to scale up the selection of the trees at each step. Additional fixed-size controller approaches have been developed that utilize an alternative representation of the controller as a Mealy machine to increase performance [32], employ expectation maximization for parameter optimization [33], or make use of more structured periodic controllers and an improved search technique [34].

Algorithms for subclasses have also been developed. The algorithm for solving transition and observation independent Dec-MDPs was the coverage set method [9]. Additional methods have also been developed to solve independent transition and observation Dec-MDPs more efficiently. These methods include a bilinear programming algorithm [35], a mixture of heuristic search and constraint optimization [36], and recasting the problem as a continuous MDP [37]. Optimal and approximate methods for solving ND-POMDPs have been proposed [12]. Other ND-POMDP methods have also been developed, such as those producing quality bounded solutions [38] and using finite-state controllers for agent policies [39].

Other models of communication are discussed in the literature [3], [5]. Communication has been studied in the context of locally fully observable Dec-MDPs with independent transitions and observations. This problem can be modeled with a cost of communication to receive the local states of the other agents and solved with a set of heuristic approaches [40]. Myopic communication, where an agent decides whether to communicate based on the assumption that communication can take place on this step or never again, has also been shown to work well in many cases [41]. Other types

of communication explored include stochastically delayed communication [42] and communication for online planning in Dec-POMDPs [43].

Factored models have been studied in the context of Dec-POMDPs. General factored models have been described and solved [44]. Witwicki and Durfee summarized different types of factored models and their complexity [45], and improved algorithms have been developed [46], [47].

In general, the research community has focused on planning methods for models that are similar to Dec-POMDPs, but a few learning methods have been explored. These include model-free reinforcement learning methods using gradient-based methods to improve the policies [48], [49] and using communication to learn solutions in ND-POMDPs [50].

Additional solution methods for general Dec-POMDPs have also been proposed. These include a mixed integer linear programming approach for Dec-POMDPs [51] and sampling methods [52], [53].

A number of applications have been studied, including multi-robot coordination in the form of space exploration rovers [54], helicopter flights [5] and navigation [55]–[57], load balancing for decentralized queues [58], network congestion control [59], multiaccess broadcast channels [60], network routing [61], sensor network management [12], target tracking [12], [62], and weather phenomena [63].

References

1. S. Seuken and S. Zilberstein, “Formal Models and Algorithms for Decentralized Control of Multiple Agents,” *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 17, no. 2, pp. 190–250, 2008. DOI: 10.1007/s10458-007-9026-5.
2. F.A. Oliehoek, “Decentralized POMDPs,” in *Reinforcement Learning: State of the Art*, M. Wiering and M. van Otterlo, eds., vol. 12, Berlin: Springer, 2012.
3. C.V. Goldman and S. Zilberstein, “Decentralized Control of Cooperative Systems: Categorization and Complexity Analysis,” *Journal of Artificial Intelligence Research*, vol. 22, pp. 143–174, 2004. DOI: 10.1613/jair.1427.
4. D.S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein, “The Complexity of Decentralized Control of Markov Decision Processes,” *Mathematics of Operations Research*, vol. 27, no. 4, pp. 819–840, 2002. DOI: 10.1287/moor.27.4.819.297.
5. D.V. Pynadath and M. Tambe, “The Communicative Multiagent Team Decision Problem: Analyzing Teamwork Theories and Models,” *Journal of Artificial Intelligence Research*, vol. 16, pp. 389–423, 2002. DOI: 10.1613/jair.1024.
6. E.A. Hansen, D.S. Bernstein, and S. Zilberstein, “Dynamic Programming for Partially Observable Stochastic Games,” in *AAAI Conference on Artificial Intelligence (AAAI)*, 2004.

7. D.S. Bernstein, C. Amato, E.A. Hansen, and S. Zilberstein, “Policy Iteration for Decentralized Control of Markov Decision Processes,” *Journal of Artificial Intelligence Research*, vol. 34, pp. 89–132, 2009. DOI: 10.1613/jair.2667.
8. D. Szer, F. Charpillet, and S. Zilberstein, “MAA*: A Heuristic Search Algorithm for Solving Decentralized POMDPs,” in *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2005.
9. R. Becker, S. Zilberstein, V. Lesser, and C.V. Goldman, “Solving Transition-Independent Decentralized Markov Decision Processes,” *Journal of Artificial Intelligence Research*, vol. 22, pp. 423–455, 2004. DOI: 10.1613/jair.1497.
10. R. Becker, V. Lesser, and S. Zilberstein, “Decentralized Markov Decision Processes with Event-Driven Interactions,” in *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2004.
11. M. Allen and S. Zilberstein, “Complexity of Decentralized Control: Special Cases,” in *Advances in Neural Information Processing Systems (NIPS)*, 2009.
12. R. Nair, P. Varakantham, M. Tambe, and M. Yokoo, “Networked Distributed POMDPs: A Synthesis of Distributed Constraint Optimization and POMDPs,” in *AAAI Conference on Artificial Intelligence (AAAI)*, 2005.
13. C. Boutilier, “Sequential Optimality and Coordination in Multiagent Systems,” in *International Joint Conference on Artificial Intelligence (IJCAI)*, 1999.
14. S. Seuken and S. Zilberstein, “Memory-Bounded Dynamic Programming for DEC-POMDPs,” in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2007.
15. R. Nair, D. Pynadath, M. Yokoo, M. Tambe, and S. Marsella, “Taming Decentralized POMDPs: Towards Efficient Policy Computation for Multiagent Settings,” in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.
16. D. Szer and F. Charpillet, “An Optimal Best-First Search Algorithm for Solving Infinite Horizon DEC-POMDPs,” in *European Conference on Machine Learning (ECML)*, 2005.
17. D.S. Bernstein, E.A. Hansen, and S. Zilberstein, “Bounded Policy Iteration for Decentralized POMDPs,” in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2005.
18. C. Amato, D.S. Bernstein, and S. Zilberstein, “Optimizing Fixed-Size Stochastic Controllers for POMDPs and Decentralized POMDPs,” *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 21, no. 3, pp. 293–320, 2010. DOI: 10.1007/s10458-009-9103-z.
19. M. Roth, R. Simmons, and M.M. Veloso, “Reasoning About Joint Beliefs for Execution-Time Communication Decisions,” in *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2005.

20. R. Nair and M. Tambe, “Communication for Improving Policy Computation in Distributed POMDPs,” in *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2004.
21. C. Amato, J.S. Dibangoye, and S. Zilberstein, “Incremental Policy Generation for Finite-Horizon DEC-POMDPs,” in *International Conference on Automated Planning and Scheduling (ICAPS)*, 2009.
22. A. Boularias and B. Chaib-draa, “Exact Dynamic Programming for Decentralized POMDPs with Lossless Policy Compression,” in *International Conference on Automated Planning and Scheduling (ICAPS)*, 2008.
23. F.A. Oliehoek, M.T.J. Spaan, and N. Vlassis, “Optimal and Approximate Q-Value Functions for Decentralized POMDPs,” *Journal of Artificial Intelligence Research*, vol. 32, pp. 289–353, 2008.
24. F.A. Oliehoek, M.T.J. Spaan, and C. Amato, “Scaling Up Optimal Heuristic Search in Dec-POMDPs via Incremental Expansion,” in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2011.
25. F.A. Oliehoek, M.T.J. Spaan, C. Amato, and S. Whiteson, “Incremental Clustering and Expansion for Faster Optimal Planning in Dec-POMDPs,” *Journal of Artificial Intelligence Research*, vol. 46, pp. 449–509, 2013. doi: 10.1613/jair.3804.
26. F.A. Oliehoek, “Sufficient Plan-Time Statistics for Decentralized POMDPs,” in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2013.
27. J.S. Dibangoye, C. Amato, O. Buffet, and F. Charpillet, “Optimally Solving Dec-POMDPs as Continuous-State MDPs,” in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2013.
28. A. Carlin and S. Zilberstein, “Value-Based Observation Compression for DEC-POMDPs,” in *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2008.
29. J.S. Dibangoye, A.-I. Mouaddib, and B. Chaib-draa, “Point-Based Incremental Pruning Heuristic for Solving Finite-Horizon DEC-POMDPs,” in *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2009.
30. A. Kumar and S. Zilberstein, “Point-Based Backup for Decentralized POMDPs: Complexity and New Algorithms,” in *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2010.
31. F. Wu, S. Zilberstein, and X. Chen, “Point-Based Policy Generation for Decentralized POMDPs,” in *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2010.
32. C. Amato, B. Bonet, and S. Zilberstein, “Finite-State Controllers Based on Mealy Machines for Centralized and Decentralized POMDPs,” in *AAAI Conference on Artificial Intelligence (AAAI)*, 2010.

33. A. Kumar and S. Zilberstein, “Anytime Planning for Decentralized POMDPs Using Expectation Maximization,” in *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2010.
34. J.K. Pajarinen and J. Peltonen, “Periodic Finite State Controllers for Efficient POMDP and DEC-POMDP Planning,” in *Advances in Neural Information Processing Systems (NIPS)*, 2011.
35. M. Petrik and S. Zilberstein, “A Bilinear Programming Approach for Multiagent Planning,” *Journal of Artificial Intelligence Research*, vol. 35, pp. 235–274, 2009. DOI: 10.1613/jair.2673.
36. J.S. Dibangoye, C. Amato, and A. Doniec, “Scaling Up Decentralized MDPs Through Heuristic Search,” in *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2012.
37. J.S. Dibangoye, C. Amato, A. Doniec, and F. Charpillat, “Producing Efficient Error-Bounded Solutions for Transition Independent Decentralized MDPs,” in *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2013.
38. P. Varakantham, J. Marecki, Y. Yabu, M. Tambe, and M. Yokoo, “Letting Loose a SPIDER on a Network of POMDPs: Generating Quality Guaranteed Policies,” in *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2007.
39. J. Marecki, T. Gupta, P. Varakantham, M. Tambe, and M. Yokoo, “Not All Agents Are Equal: Scaling up Distributed POMDPs for Agent Networks,” in *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2008.
40. P. Xuan and V. Lesser, “Multi-Agent Policies: From Centralized Ones to Decentralized Ones,” in *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2002.
41. R. Becker, A. Carlin, V. Lesser, and S. Zilberstein, “Analyzing Myopic Approaches for Multi-Agent Communication,” *Computational Intelligence*, vol. 25, no. 1, pp. 31–50, 2009. DOI: 10.1111/j.1467-8640.2008.01329.x.
42. M.T.J. Spaan, F.A. Oliehoek, and N. Vlassis, “Multiagent Planning Under Uncertainty with Stochastic Communication Delays,” in *International Conference on Automated Planning and Scheduling (ICAPS)*, 2008.
43. F. Wu, S. Zilberstein, and X. Chen, “Multi-Agent Online Planning with Communication,” in *International Conference on Automated Planning and Scheduling (ICAPS)*, 2009.
44. F.A. Oliehoek, M.T.J. Spaan, S. Whiteson, and N. Vlassis, “Exploiting Locality of Interaction in Factored Dec-POMDPs,” in *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2008.

45. S.J. Witwicki and E.H. Durfee, “Towards a Unifying Characterization for Quantifying Weak Coupling in Dec-POMDPs,” in *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2011.
46. S.J. Witwicki, F.A. Oliehoek, and L.P. Kaelbling, “Heuristic Search of Multiagent Influence Space,” in *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2012.
47. F.A. Oliehoek, S. Whiteson, and M.T.J. Spaan, “Approximate Solutions for Factored Dec-POMDPs with Many Agents,” in *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2013.
48. A. Dutech, O. Buffet, and F. Charpillet, “Multi-Agent Systems by Incremental Gradient Reinforcement Learning,” in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2001.
49. L. Peshkin, K.-E. Kim, N. Meuleau, and L.P. Kaelbling, “Learning to Cooperate via Policy Search,” in *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2000.
50. C. Zhang and V.R. Lesser, “Coordinated Multi-Agent Reinforcement Learning in Networked Distributed POMDPs,” in *AAAI Conference on Artificial Intelligence (AAAI)*, 2011.
51. R. Aras, A. Dutech, and F. Charpillet, “Mixed Integer Linear Programming for Exact Finite-Horizon Planning in Decentralized POMDPs,” in *International Conference on Automated Planning and Scheduling (ICAPS)*, 2007.
52. C. Amato and S. Zilberstein, “Achieving Goals in Decentralized POMDPs,” in *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2009.
53. F.A. Oliehoek, J.F. Kooi, and N. Vlassis, “The Cross-Entropy Method for Policy Search in Decentralized POMDPs,” *Informatica*, vol. 32, no. 4, pp. 341–357, 2008.
54. D.S. Bernstein, S. Zilberstein, R. Washington, and J.L. Bresina, “Planetary Rover Control as a Markov Decision Process,” in *International Symposium on Artificial Intelligence, Robotics and Automation in Space*, 2001.
55. R. Emery-Montemerlo, G. Gordon, J. Schneider, and S. Thrun, “Game Theoretic Control for Robot Teams,” in *IEEE International Conference on Robotics and Automation (ICRA)*, 2005.
56. M.T.J. Spaan and F.S. Melo, “Interaction-Driven Markov Games for Decentralized Multiagent Planning Under Uncertainty,” in *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2008.

57. L. Matignon, L. Jeanpierre, and A.-I. Mouaddib, “Coordinated Multi-Robot Exploration Under Communication Constraints Using Decentralized Markov Decision Processes,” in *AAAI Conference on Artificial Intelligence (AAAI)*, 2012.
58. R. Cogill, M. Rotkowitz, B. Van Roy, and S. Lall, “An Approximate Dynamic Programming Approach to Decentralized Control of Stochastic Systems,” in *Allerton Conference on Communication, Control, and Computing*, 2004.
59. K. Winstein and H. Balakrishnan, “TCP Ex Machina: Computer-Generated Congestion Control,” in *ACM Special Interest Group on Data Communication (SIGCOMM)*, 2013.
60. J.M. Ooi and G.W. Wornell, “Decentralized Control of a Multiple Access Broadcast Channel: Performance Bounds,” in *IEEE Conference on Decision and Control (CDC)*, 1996.
61. L. Peshkin and V. Savova, “Reinforcement Learning for Adaptive Routing,” in *International Joint Conference on Neural Networks (IJCNN)*, 2002.
62. A. Kumar and S. Zilberstein, “Constraint-Based Dynamic Programming for Decentralized POMDPs with Structured Interactions,” in *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2009.
63. —, “Event-Detecting Multi-Agent MDPs: Complexity and Constant-Factor Approximation,” in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2009.

II

APPLICATION

8

Probabilistic Surveillance Video Search

Jason R. Thornton

Driven by recent advances in digital video camera design, video management systems, and efficient archiving, security personnel at large facilities and urban sites increasingly deploy comprehensive video surveillance systems [1]. Such systems support the ability for operators to monitor large areas in real time and the ability for operators to conduct forensic review after a key incident or report. While the fully attentive human visual system is adept at interpreting video content, it is also limited in capacity; as a result, operator review of large amounts of surveillance video can be a slow and tedious process. Automatic video search technology can help relieve some of the burden on human operators. This technology can direct the attention of security personnel or investigators to potentially useful video content [2]. Although there are a number of ways to analyze and index video (e.g., based on observed actions, motion patterns, or object appearances), this chapter focuses on the particular problem of attribute-based person search [3]. We will present a probabilistic approach to the problem that is especially suited to the uncertainty of video observations, and we discuss its performance in realistic surveillance environments.

8.1 Attribute-Based Person Search

The problem of searching for a person recorded within surveillance video may be divided into two subproblems: biometric search or attribute-based search. Biometric search refers to the process of looking for a person whose physiological characteristics (such as facial appearance) indicate a precise match to a known identity. Although automatic recognition based on face [4] or other biometric signatures [5] can be performed under certain conditions, this approach requires access to a previously captured biometric template against which to perform matching. This approach cannot be used when analysts or investigators are working only with a description of a person of interest or a sample image that does not adequately capture a unique biometric signature.

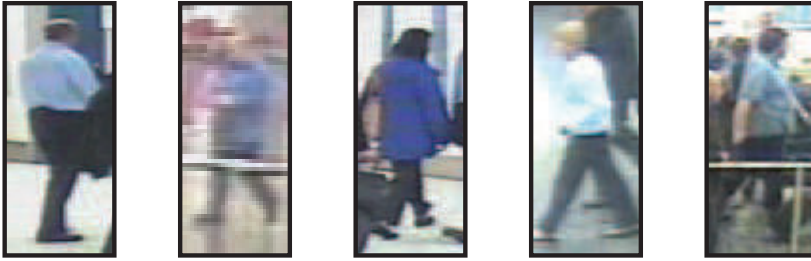


Figure 8.1 Appearance variations among images that fit a particular attribute description (blue shirt and black pants).

An alternative approach is attribute-based search, in which the system looks for persons who match a basic appearance profile, including attributes that are observable at a distance, such as hair color, clothing type and color, gender, and accompanying bags or other carried items. These descriptions are sometimes called “soft biometrics” because they are temporary in nature and taken together do not necessarily describe a unique individual in the area under surveillance. However, these profiles are usually specific enough to significantly narrow the subset of persons who require further inspection.

Automatic attribute-based search is a difficult problem because there can be significant variations in appearance among images that match the same attribute profile. These variations are due to determining factors such as pose, clothing style, lighting conditions, view angle, and naturally occurring differences within the human population. Figure 8.1 shows a sample set of images exhibiting the basic attributes “blue shirt and black pants.” Although these are all matches to the same description, inspection of the observed pixel values within these images reveals considerable differences. An accurate search technique must successfully interpret image content despite these differences.

Although the problem of attribute-based search has been addressed before (with a particular focus on face attributes in close-range video [6]), in this chapter, we consider a probabilistic approach to the problem. The chief advantage of probabilistic techniques is that they provide an elegant way to account for collective variations in appearance, especially when these variations are dependent on latent (unobserved) factors of image composition.

8.1.1 Applications

Before considering solutions to the attribute search problem, we highlight its value for security applications. There are many cases in which investigators or security personnel must monitor multiple distributed camera views for relevant pieces of information related to pedestrian activity. Example scenarios include monitoring of large public facilities, the areas surrounding critical infrastructure, or the many urban areas under the

surveillance of law enforcement. In these cases, investigatory actions are often triggered by either witness or officer descriptions of a person of interest. Systems capable of performing attribute-based search would be able to scan many hours of surveillance video for apparent matches to that description, dramatically reducing the tedious video review process.

In addition to forensic video review, effective attribute search can be used to monitor a collection of live camera feeds in real time for potential matches. In this mode of operation, the system constantly scans incoming feeds to look for matches to a supplied description and updates a list of alerts when a sufficiently strong match appears. Analysts then review this list to acquire cues about geo-location of potential persons of interest. Note that in any application, operators have some tolerance to search errors because they can review and then confirm or deny the validity of the search results. However, the extent to which an attribute search tool can speed up surveillance video review depends heavily upon the accuracy of the search technique.

8.1.2 Person Detection

The search technique described in this chapter has two main components: detection and scoring. The first component ingests raw surveillance video and attempts to detect all instances of moving persons within each camera view. This process is designed to run in real time as video is recorded, storing all detections as time-stamped and location-stamped records in a database for later reference. Rather than process every frame of video, the system analyzes one sample frame per second because of the high redundancy in content between consecutive frames. It also detects only persons in motion because there is no need to store stationary subjects repeatedly to the database.

The method for extracting detections from a single frame applies a standard sliding window approach to evaluate candidate detection locations within the scene. Each candidate location is specified by the position of a bounding box (x, y) and its dimensions (w, h) . Because the technique assumes a fixed aspect ratio of height to width for the bounding box dimensions, there are effectively three parameter values to sweep through during search.

At every candidate location, the system evaluates that subsection of the image to check whether it satisfies three criteria: it must exhibit the shape of a person, it must fit into the ground plane constraints of the scene, and it must contain a high ratio of pixels exhibiting motion (as opposed to a static background). The combination of all three criteria leads to fairly robust moving-person detection [3], with a relatively low number of false positives (i.e., non-person clutter) stored to the database.

Figure 8.2 depicts the detections for sample frames of video captured within two different airport surveillance environments. In these cases, the detection process has successfully captured every example of pedestrian motion. Note that the method does



Figure 8.2 Detections of moving persons in sample video frames from two different airport environments. The first frame is from the Performance Evaluation of Tracking and Surveillance (PETS) dataset [7], and the second is from the i-LIDS Multiple Camera Tracking Dataset [8].

not always detect every instance of a person moving through the processed camera views (especially those instances in which there is heavy occlusion by other scene components). However, it is likely to capture at least a subset of the frame-by-frame appearances of each person, so that there is some representation of each individual within the database.

8.1.3 Retrieval and Scoring

Once a database of detections has been constructed, we need a method for finding matches from that database for a given search request. The search request is composed of three parts: a search time window, a subset of cameras or regions within cameras over which to search, and an attribute profile. The first two criteria determine the records retrieved from the database based on their time and location stamps. Next, each candidate record is scored for its degree of match to the attribute profile. Finally, the records with scores exceeding some baseline threshold are sorted based on descending score, and the top set of matches is returned as the search results.

The most critical part of the search process is the scoring mechanism. It must be accurate enough to provide a good measure of degree of relevance, ranking clear matches higher than near-matches and near-matches higher than clear mismatches, while allowing for rapid evaluation to minimize wait time. The model described in the following section uses principles of probabilistic reasoning to achieve the desired scoring behavior.

8.2 Probabilistic Appearance Model

In this section, we formulate a probabilistic model to describe the relationship between an attribute description and a surveillance video observation of someone who matches that description. The model is *generative* in the sense that it describes the likelihood that one observed state (the attribute profile) will lead to the generation of another observed state (the set of digital pixel values that comprise a surveillance image chip). These pieces of information are collectively referred to as the observed states O of the system.

The many different ways in which an attribute profile can generate a set of observed pixel values are due to variations among both the human population and the circumstances of video collection. We can account for some of this variation explicitly by adding a set of hidden states H to the generative model in order to represent important factors, such as the position of the body components within the observed image chip. Besides providing a more defined model structure, the hidden states of the generative model decompose the full joint probability distribution into a collection of more tractable conditional distributions depending on key states of the image composition.

For a given instantiation of the observed variable set, we compute a match score s by maximizing the joint probability of the model with respect to the hidden states:

$$s = \max_H p(O, H). \quad (8.1)$$

Intuitively, this approach finds the most likely explanation of the observed data in the form of the hidden states that explain image formation before scoring the relevance of an image chip. When the observed image is a good fit to the attribute description, there should exist a set of hidden state values that lead to relatively high generative probabilities. Conversely, when the observed image is not a good match, there should exist no hidden state values that lead to high-probability model evaluations. When properly formulated, a generative model is an effective way to interpret the likelihood of observed image data while accounting for expected variations.

8.2.1 Observed States

First, we define the observed states of the model more precisely. The attribute vector $\mathbf{A}$ comprises a set of appearance-based properties that are visible at a distance, including gender, clothing color, and bag information. Table 8.1 lists the individual variables contained within $\mathbf{A}$. In this definition, there are six categorical variables, each of which can be set to a value of *unspecified* to accommodate cases in which only limited information is available.

Table 8.1 Variables within attribute set A.

Variable	Description	Possible values
g	Gender	Unspecified, male, or female
h	Head color (hair or hat)	Unspecified, or one of 12 color categories
t	Torso clothing color	Unspecified, or one of 12 color categories
l	Lower body clothing color	Unspecified, or one of 12 color categories
b_t	Primary bag type	Unspecified, or one of three types
b_c	Primary bag color	Unspecified, or one of 12 color categories

The color values are defined to include the following 12 common perceptual categories: white, gray, black, yellow, orange, pink, red, green, blue, purple, brown, and beige. As we discuss later in this section, this palette can be extended to incorporate any value within the color space or mixtures of multiple colors. The bag types are divided into the three categories most commonly found in transportation settings such as airports: backpacks, hand-carried bags, and large rolled luggage.

Besides the attribute profile, the other major observed state in the generative model is the image chip to be evaluated. Rather than using the pixel values directly, we can apply a feature extraction step to the image to compute features that capture summary information about color and shape. For instance, every pixel value can be assigned a color category based on its location in the color space.

Let $\mathbf{X}_k$ represent the three-dimensional value of the k th pixel in the hue-saturation-value (HSV) color space. The HSV color space is defined on a cylindrical coordinate system and is designed so that the cyclical hue axis corresponds well to perceptual color differentiation by the human visual system. We model each of the 12 perceptual color categories as a probability density function within the HSV color space.

Figure 8.3 depicts the statistical dependencies of the color model. C_k represents the color category variable for pixel k , which can take any integer value from 1 to 12. Parameter set ϕ contains a mean vector μ and covariance matrix Σ for each of the 12 categories. These parameter values are learned by fitting densities to samples assigned to each category by test subjects. They collectively form the color model that is used to evaluate every pixel in the image.

We assume the prior probability of each color category is uniform, and the conditional probability of the pixel observation is given by the quasi-normal density

$$p(\mathbf{X}_k | C_k = i, \phi) = \phi(\phi_i) \exp(-0.5 \cdot \mathbf{d}(\mathbf{X}_k, \mu_i)^\top \Sigma_i^{-1} \mathbf{d}(\mathbf{X}_k, \mu_i)), \quad (8.2)$$

where $\mathbf{d}(\cdot)$ is a vector difference operator, modified to work on the HSV cylindrical coordinate system so that the difference on the cyclical hue axis (which ranges from 0

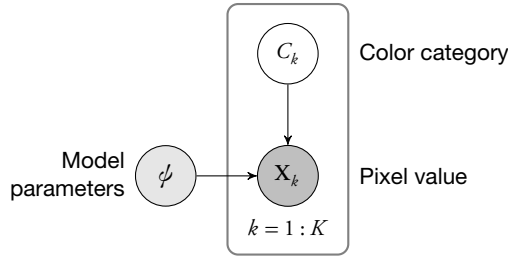


Figure 8.3 Color model relating a hidden color category to an observed pixel value for each of the K pixel values contained in an image.

to 1) is given by

$$d(X_k^{hue}, \mu_i^{hue}) = \text{mod}(X_k^{hue} - \mu_i^{hue} + 0.5, 1) - 0.5. \quad (8.3)$$

Because this is a truncated distribution (defined over a finite region of the cylindrical coordinate space), constant $\phi(\psi_i)$ is set to normalize the distribution to sum to one.

Using this model, we assign each pixel a color category according to the maximum likelihood method:

$$C_k = \arg \max_i p(\mathbf{X}_k | C_k = i, \psi_i). \quad (8.4)$$

The resulting features (i.e., pixel-wise color labels) must only be extracted once when the image of the moving person is detected, and these labels are stored to a database.

Although we do not discuss the details here, we can apply a similar process to extract other primitives that capture information about local edges or textures surrounding each pixel. For example, gradient extraction filters may be applied to an image in order to assign each pixel a category based on edge magnitude and orientation. Like the color categories, these features represent observed states with some level of conceptual meaning; these states relate to the hidden states of image composition within the generative model.

8.2.2 Basic Model Structure

The generative model described in this section has a hierarchical structure, with key factors of the image formation encoded as latent variables in the hierarchy. Figure 8.4 depicts the variables and the dependencies between them as a graphical model. First, the image is partitioned into its component parts (e.g., head, torso, etc.). Then the model defines a visual theme for each component or a distribution over observable features, according to the label assigned it by the attribute profile.

Vector $\mathbf{Z}$ represents the latent variable that partitions the image into its component parts: head, torso, lower body, and (optionally) bag. Each of these component regions is

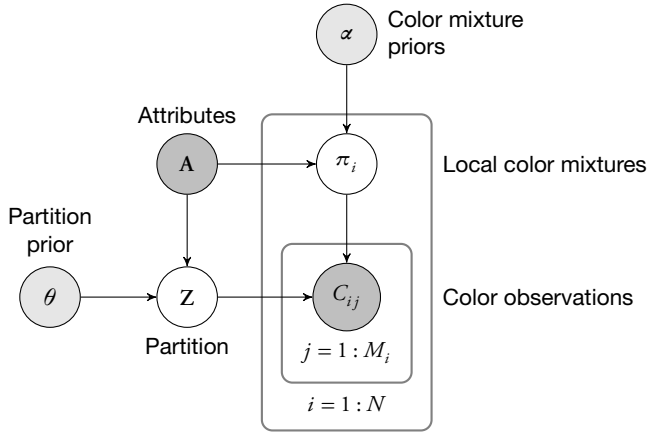


Figure 8.4 Graphical model visualization of the basic generative model for image formation based on an attribute description. Darkly shaded nodes represent observed variables, lightly shaded nodes represent parameters, and unshaded nodes represent hidden (or latent) states.

specified by a bounding box, as depicted by the sample partitions in Figure 8.5. The first portion of this vector, $\mathbf{z}_{body}$, encodes the two-dimensional position of the body within the image chip because the detected image chip does not form a tight bound around the person:

$$\mathbf{z}_{body} = [x_{body}, y_{body}, w_{body}, h_{body}]. \quad (8.5)$$

The four values specify x -position, y -position, width, and height in units relative to the width and height of the full image chip. Representing these values as ratios of the image chip size makes them invariant to the image resolution at which the person is detected.

In addition to whole body position, there are also separate bounding box rectangles for the components of interest: $\mathbf{z}_{body}$, $\mathbf{z}_{head}$, $\mathbf{z}_{torso}$, and $\mathbf{z}_{bag}$ (if needed). Just as $\mathbf{z}_{body}$ is defined relative to the entire image chip, each of these components is defined relative to the position and size of $\mathbf{z}_{body}$. To define a distribution over the partition variables, the model groups all non-bag components into a single vector,

$$\mathbf{z}_{base} = [\mathbf{z}_{body}, \mathbf{z}_{head}, \mathbf{z}_{torso}, \mathbf{z}_{lower}], \quad (8.6)$$

and assumes that these values are jointly distributed according to a truncated normal distribution with mean vector μ_{base} and covariance matrix Σ_{base} . Selecting these values from a joint distribution accounts for correlations between the component positions (e.g., a head position left of center usually corresponds to a torso position left of center).

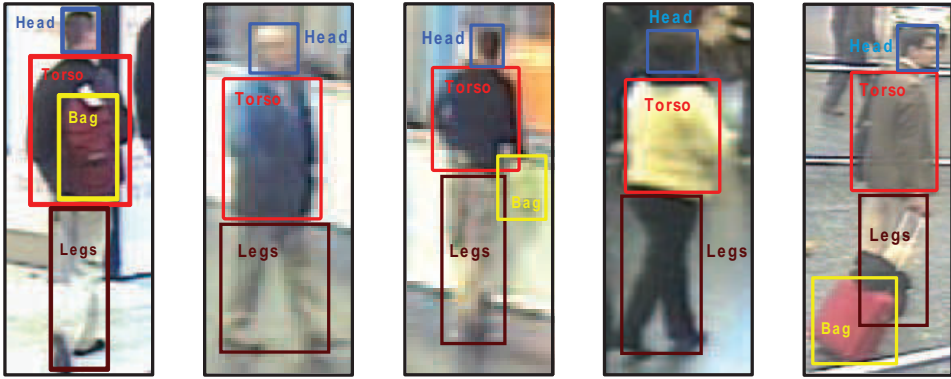


Figure 8.5 Example body component partitions, including accompanying bags. In the generative model, variable Z specifies the joint positions of these components using bounding boxes.

The full state vector Z comprises the basic position vector appended to the bag position vector when bag information is specified by the attribute profile:

$$Z = [\mathbf{z}_{base}, \mathbf{z}_{bag}]. \quad (8.7)$$

Unlike the body positions, the bag position is not sufficiently described by a single unimodal distribution because there is higher variability in the shape of bags and the way in which they are transported. In airport environments, for instance, there are at least three distinct common bag types: backpacks, hand-carried bags such as purses or small luggage, and larger luggage that is typically transported by rolling. Figure 8.5 shows one example of each bag type.

In the attribute set, variable b_t specifies the bag type, and so the model employs three different distributions (conditioned on b_t). In each case, bag position is modeled as a nonparametric kernel density learned from training examples. Each density is converted into a fast lookup table spanning the possible values of the $\mathbf{z}_{bag}$ rectangle. Each of the three lookup tables is denoted by a discrete vector of values $\mathbf{T}$ so that the full set of parameters governing the selection of the partition state is

$$\theta = \{\mu_{base}, \Sigma_{base}, \mathbf{T}_{backpack}, \mathbf{T}_{hand-carried}, \mathbf{T}_{rolled-luggage}\}. \quad (8.8)$$

With all of the relevant partition variables defined, the conditional distribution of the partition state given the attribute set and the partition parameters is given by

$$p(Z | \mathbf{A}, \theta) = \mathcal{N}(\mathbf{z}_{base} | \mu_{base}, \Sigma_{base}) \cdot T_{b_t}(\mathbf{z}_{bag}), \quad (8.9)$$

where bag type b_t from the attribute set is used to select the appropriate lookup table for determining the probability of Z_{bag} . When no bag information is specified, the algorithm selects a uniform probability since bag position becomes irrelevant in this case.

Once the component positions have been established, the model governs the observed features within each component region according to the attribute profile. To do this, the model takes a simplified form of the model used in well-known latent topic models, which have been previously applied to the interpretation of textual [9] and visual [10] data. In this application, of course, the *topics* (or mixtures of observed features) are visual.

To illustrate the concept, we consider the generation of observed color information at the pixel level, although there are other pixel-level features that the model can incorporate as well. First, we define a discrete number of elemental color categories, selected to match the N_c basic perceptual color options of the attribute profile. Each of the N component regions is associated with a mixture over these color categories, represented by latent topic vector π_i . This real-valued state vector, which sums to one, assigns a proportional weight to each color category, indicating its likelihood of observation within the local component.

It is important to note that the observed color categories within a component depend on the attribute description for that component, but this relationship is not deterministic. For example, a torso with a “dark red shirt” label is likely to contain a mixture dominated by red and black color categories. However, the exact mixture depends on a number of other factors, such as clothing style, material composition, and lighting and shadow effects. To account for this variation, the color topic state vector is drawn from a Dirichlet prior distribution. As discussed in Section 2.3.2, the Dirichlet density is often used as a prior distribution over the parameters of a categorical variable. Consequently, the color mixture π_i is drawn according to

$$p(\pi_i | \omega) = \phi(\omega) \prod_{k=1}^{N_c} \pi_i(k)^{\omega(k)-1}, \quad (8.10)$$

where ω is the vector of Dirichlet *pseudocount* parameters and $\phi(\omega)$ is the Dirichlet normalization factor.

Because the Dirichlet parameters are different for each color specified by the attribute profile, we represent all parameters of the color mixture priors in a matrix α of size N_c by N_c . The k th row of this matrix, denoted by $\alpha(k)$, gives the Dirichlet parameter vector corresponding to the k th color. If we take the color topic of the torso component π_{torso} as an example, the prior probability of this topic depends on the torso clothing

color t given in the attribute profile $\mathbf{A}$, so that

$$p(\pi_{t_{orso}} | \mathbf{A}, \alpha) = \phi(\alpha(t)) \prod_{k=1}^{N_c} \pi_{t_{orso}}(k)^{\alpha(t)(k)-1}. \quad (8.11)$$

We define the conditional probabilities for each π_i component in a similar way by substituting the relevant variable from the attribute profile for t in the above expression.

After we select the color topic for each local component, the color observation at each pixel is drawn according to the topic of the component containing it. This generation of observed color states is performed only at foreground pixels (i.e., pixels that have been labeled as part of the person or accompanying objects, not the static background). Let C_q represent the color observation of the q th foreground pixel within the image chip, which is a categorical variable that can take one of the N_c color categories. The probability of C_q , conditioned on the partition $\mathbf{Z}$ and the set of topics $\{\pi_1, \dots, \pi_n\}$, is given by

$$P(C_q | \mathbf{Z}, \pi_1, \dots, \pi_n) = \pi_i(C_q), \quad (8.12)$$

where component index i is derived from a selector function

$$i = S(\mathbf{Z}, q) \quad (8.13)$$

that maps each pixel index to the component that contains it, according to the partition. Effectively, the partition determines which color topic applies, and then that topic becomes the probability mass function for drawing the pixel-level observations.

Because in this model each individual color observation is drawn independently of the other pixel-level observations, given the topic, it is sufficient to express the observed states as histograms, or categorical frequency counts. Let function $y(\mathbf{Z}, i, j)$ total all the observations of color category j within the i th component (according to partition $\mathbf{Z}$). Then the histogram corresponding to component i is defined as

$$\mathbf{Y}_i = [y(\mathbf{Z}, i, 1), y(\mathbf{Z}, i, 2), \dots, y(\mathbf{Z}, i, N_c)]. \quad (8.14)$$

Instead of drawing individual samples from the topic, we can now use the model structure depicted in Figure 8.6. In this equivalent but simplified structure, a single histogram is generated at each local component according to the mixture specified by the topic. To remove the effect of image resolution, we normalize each histogram so that the sum across all counts equals 100 because the only information that needs to be preserved is the relative frequency of each category. For 100 independent draws from a given probability mass function, the probability of the resulting frequency histogram is given by the multinomial distribution:

$$p(\mathbf{Y}_i | \mathbf{Z}, \pi_i) = \frac{100!}{\prod_{k=1}^{N_c} y(\mathbf{Z}, i, k)!} \prod_{k=1}^{N_c} \pi_i(k)^{y(\mathbf{Z}, i, k)}. \quad (8.15)$$

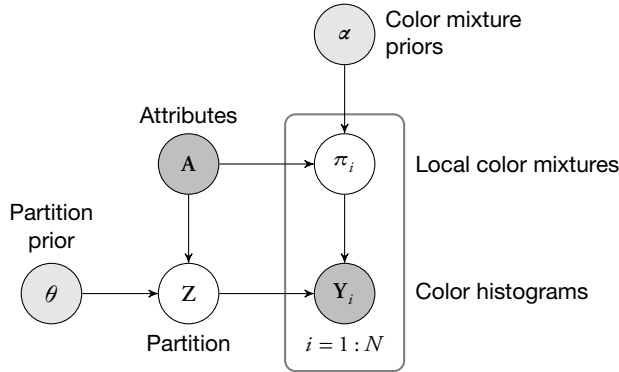


Figure 8.6 Graphical model visualization of the generative model, using histograms of color observations instead of individual pixel-level observations.

We have now fully defined the foundation for a simple generative image model. The major conceptual mechanism of the model is a selection of body and accompanying bag component positions, followed by the selection of feature-based “topics” within each component, and finally a selection of the observed pixel-level states. In the following subsection, we consider extensions to this basic model structure.

8.2.3 Model Extensions

Although the basic generative model provides an effective foundation for attribute-based search, we can add flexibility to the model by incorporating additional forms of attribute profile inputs or pixel-level observations. Each of the following model extensions can add either precision or accuracy to the attribute search process.

8.2.3.1 Gender

The attribute set described in Table 8.1 includes an observed variable g that indicates the gender of the person of interest. To add this consideration into the model, we need some way to derive an observable metric from the image chip relating to apparent gender.

The person detection process outlined in Section 8.1.2 computes a set of features that characterize local gradient information (known as histograms of oriented gradients features [11]). These features are passed into a classifier that is trained to recognize human-like contours, an important criterion for detection. Because these features also capture some information relating to apparent gender (e.g., hairstyle, clothing style, body frame), we can reuse them to build a gender classifier. The resulting classifier produces a

real-valued classification score G , which ranges along the interval $[-G_{\max}, G_{\max}]$. Scores that approach the two extrema of this interval indicate strong evidence for male or female, respectively, whereas scores near the center at zero indicate relatively weak or inconclusive evidence.

Figure 8.7 depicts an extension of the basic generative model that includes the gender classifier output as an observed state. To define a conditional probability distribution over G , we first define two functions that remap the value of G by using the gender specification g from the attribute profile.

The first function $s(G, g)$ alters the sign of G according to g , so that positive values always indicate a better match to the specified gender and negative values always indicate a worse match. If g is unspecified by the attribute profile, then $s(G, g)$ maps to zero (because the value of G does not matter in this case).

The second function maps the value of G to an interval ranging from zero to one,

$$m(G, g) = \frac{G_{\max} - s(G, g)}{2 \cdot G_{\max}}, \quad (8.16)$$

so that a value of zero corresponds to the strongest possible evidence of a profile match and a value of one corresponds to the strongest possible evidence of a profile mismatch.

The model defines the conditional probability of the gender score, given the specified gender attribute, as a truncated exponential distribution over the remapped score:

$$p(G \mid \mathbf{A}, \lambda) = \phi(\lambda) \cdot e^{-\lambda \cdot m(G, g)}. \quad (8.17)$$

Parameter λ determines the shape of the distribution, and $\phi(\lambda)$ normalizes the truncated exponential probability density function so that it integrates to one. Note that this distribution yields relatively high probability values near zero (strong match evidence) but declines rapidly as the score increases toward one (strong mismatch evidence).

Because λ determines the rate of exponential change, it effectively controls the influence of this observation compared with the rest of the observations in the model. In other words, higher λ values cause the gender analysis to have a larger effect on the full joint probability of the model and, therefore, a larger effect on the overall match score of the image chip. For this reason, λ can be treated as a tunable parameter of the matching algorithm, set to emphasize the relative importance of gender compared to other attribute profile elements.

The gender score is an example of an observed state that can be derived from the entire image chip by means of a pretrained classifier. Although we do not discuss the details here, similar types of classifiers can be used to extract other relevant states, such as a person's height or build. The generative model can incorporate these states as well by using equivalent branch structures and conditional probability functions.

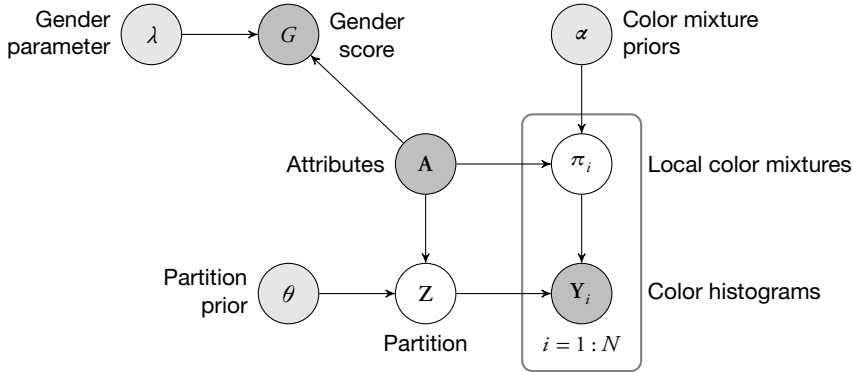


Figure 8.7 Model extension to include an observed gender indicator G and the parameter λ of its statistical dependence on the attribute set.

8.2.3.2 Color Variability

The model employs a predefined set of common perceptual colors to enable categorical feature extraction and color topic representations. However, we do not necessarily have to limit the color specifications of the attribute profile to these categorical options. Often the description of a person of interest will contain specific color shades (e.g., light blue) or color definitions that do not map well to any of the predefined options.

As an alternative to categorical specifications, we allow colors to be defined by any point in the RGB or HSV color space (such precise values can be selected easily from a color palette interface). We then project this point onto the basis of common perceptual colors using the color space distribution learned for each one. The result is a vector of proportional coefficients ρ , with length equal to the number of color categories N_c , that sums to one.

Equation (8.11) defines the conditional probability of a color topic within one image component given the Dirichlet parameter set. In that case, where the specified color is given by a categorical variable t , the relevant vector of Dirichlet parameters is selected by taking a single row $\alpha(t)$ from the parameter matrix α . Alternatively, when the color is represented proportionally by ρ , we use a weighted combination to compute the Dirichlet parameters:

$$\alpha(\rho) = \sum_{k=1}^{N_c} \rho(k) \alpha(k). \quad (8.18)$$

In practice, this approach is an effective way to add flexibility to the color specification process without requiring large category sets (and many corresponding color models). This approach also provides a way to handle multiple colors specified for the same image

component. In this case, the multiple color values are mixed by assigning them equal weights within the ρ vector.

8.2.3.3 Edge Primitives

The basic model structure uses color features at the pixel level to characterize the observed states within each image component. Although color information is an important cue, it is not the only useful low-level primitive. Especially to help differentiate between types of components, we may wish to incorporate features based on shapes, edges, gradients, or textures.

As an example, we consider the extraction of edge information using filter bank processing. It is possible to extract edge or gradient characteristics surrounding each pixel within an image chip by applying a set of filters that measure intensity changes at different scales and orientations (such as Gabor filters [12]) and analyzing the joint responses. Partitioning the response space categorizes the local edge or gradient type observed at each location. These features are useful for image interpretation because different image components (e.g., bags versus torsos) may exhibit different mixtures of edge types.

Once the edge features have been defined, we can represent the *edge topic* of each image component as a latent variable in the model. This model extension is analogous to the way color topics are incorporated into the basic model because both are representations of expected frequencies of pixel-level states. Figure 8.8 outlines an extended version of the base model, with a parallel branch for the generation of edge states. Vector ω_i represents the edge topic for the i th component, and β is an N by N_e matrix of Dirichlet weights, where N is the number of image components and N_e is the number of edge categories.

The probability of an edge topic is given by a Dirichlet density with weights selected from the row of β corresponding to the component index (as opposed to the color specification used for α row selection):

$$p(\omega_i | \beta) = \phi(\beta(i)) \prod_{k=1}^{N_e} \omega_i(k)^{\beta(i)(k)-1}. \quad (8.19)$$

Vector E_i stores a histogram of edge observation counts, where, just as with the color histograms, the counts are a function $\epsilon(Z, i, k)$ of the partition Z , the component index i , and the feature category k :

$$E_i = [\epsilon(Z, i, 1), \epsilon(Z, i, 2), \dots, \epsilon(Z, i, N_e)]. \quad (8.20)$$

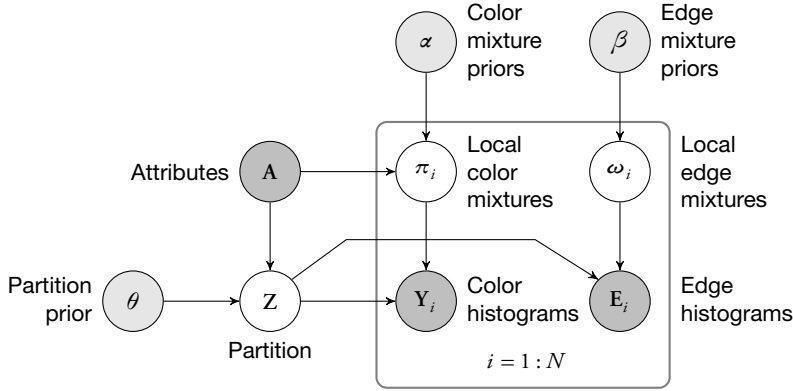


Figure 8.8 Model extension to include observed edge states using a variable dependence branch that parallels the color feature branch.

Finally, the conditional probability of the edge histogram (normalized to a total count of 100) is given by the multinomial distribution

$$p(E_i | Z, \omega_i) = \frac{100!}{\prod_{k=1}^{N_e} \epsilon(Z, i, k)!} \prod_{k=1}^{N_e} \omega_i(k)^{\epsilon(Z, i, k)}. \quad (8.21)$$

Including an extra primitive type (such as edge features) into the model enables more accurate image interpretation because the matching process accounts for multiple observed factors simultaneously. This combination of feature types can resolve ambiguities that manifest in individual feature types. Any categorical pixel-based feature can be incorporated into the model by extending branches that parallel those in Figure 8.8.

8.3 Learning and Inference Techniques

The model presented in this chapter has three fundamental types of variables: parameters, observed states, and hidden states. The model parameters are estimated during an initial training stage and then held constant during model application. In this section, we describe the process of learning the model parameters using maximum likelihood estimation with respect to a training dataset. Once the parameter values have been selected, the model may be used to estimate the likelihood of observing an image given an attribute profile.

We present an efficient procedure for performing inference on the hidden states of the model to estimate this joint probability value and assign a final match score. Because the model has a nontrivial variable dependence structure, like most models of practical application, the maximum likelihood estimates of both parameter values

and hidden states do not necessarily have closed-form solutions. Therefore, we rely on several approximate estimation techniques to converge on a solution.

8.3.1 Parameter Learning

To facilitate learning of model parameters, we must have access to a dataset of annotated training images. For instance, we can take example moving person detections from multiple source videos and annotate some ground truth regarding the hidden states of each example. In particular, if we label the partition of each image into its component parts as depicted in Figure 8.5, along with the primary color and component type descriptors, then we have enough data to learn the key conditional distributions of the model. Assigning labels to training data at this level of detail takes some time and effort, but the process is not prohibitively resource-intensive because it requires selecting only a few bounding boxes and categories for each image chip. The resulting data are sufficient for learning the model parameters, as described in the rest of this subsection.

8.3.1.1 Partition Parameters

Figure 8.9 expresses the partition parameter learning problem as a plate diagram. Vector $\mathbf{Z}_j$ represents the j th partition observation within the labeled dataset, which contains a total of M training examples. Recall from Section 8.2.2 that the partition vector consists of two parts:

$$\mathbf{Z} = [\mathbf{z}_{base}, \mathbf{z}_{bag}], \quad (8.22)$$

representing the base body components (whole body, head, torso, and lower body) and the bag component, if it exists.

The value of $\mathbf{z}_{base}$ follows a multivariate normal distribution, whereas the value of $\mathbf{z}_{bag}$ follows a nonparametric distribution according to bag type. Consequently, the parameter set θ contains the following five (multidimensional) parameters:

$$\theta = [\mu, \Sigma, \mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3]. \quad (8.23)$$

The first two specify the mean vector and covariance matrix for $\mathbf{z}_{base}$, and the last three represent lookup tables for the probabilities of $\mathbf{z}_{bag}$ associated with each of the three major bag types.

To train the model for body component positions, we assemble a dataset containing only the $\mathbf{z}_{base}$ portion of each labeled partition example:

$$D_{base} = \{\mathbf{z}_j : \mathbf{z}_j = \mathbf{z}_{base} \text{ from } \mathbf{Z}_j\}_{j=1}^M. \quad (8.24)$$

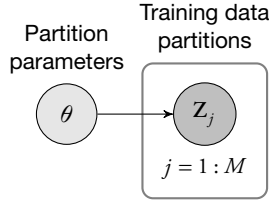


Figure 8.9 Plate diagram for the partition parameter learning problem. The likelihood of many observed partition values depends on a single parameter set.

Our objective is to maximize the probability of D_{base} with respect to the multivariate normal parameters, assuming each training sample is statistically independent. In the case of a normal distribution, the maximum likelihood estimate of the mean vector μ is well known to be the sample mean,

$$\hat{\mu} = \arg \max_{\mu} p(D_{base} | \mu, \Sigma) = \frac{1}{M} \sum_{j=1}^M \mathbf{z}_j, \quad (8.25)$$

and the maximum likelihood estimate of Σ is the sample covariance matrix,

$$\hat{\Sigma} = \arg \max_{\Sigma} p(D_{base} | \hat{\mu}, \Sigma) = \frac{1}{M-1} \sum_{j=1}^M (\mathbf{z}_j - \hat{\mu})(\mathbf{z}_j - \hat{\mu})^\top. \quad (8.26)$$

Learning this part of the model is therefore a straightforward process of substituting the training data values into a closed-form solution for our parameter estimates.

Because the bag positions do not tend to follow unimodal Gaussian distributions, these require a different type of learning. To distinguish between bag types, we define three different datasets, D_1 , D_2 , and D_3 , corresponding to the three possible values of bag type variable b_t from the generative model. Each dataset contains all samples of a particular bag type annotated within the training data:

$$D_i = \{\mathbf{z}_j : \mathbf{z}_j = \mathbf{z}_{bag} \text{ and } b_t = i\}_{j=1}^{M_i}. \quad (8.27)$$

Because bag positions are not well characterized by traditional parametric densities, we instead use nonparametric kernel densities to determine probability. The density of the bag position is set as a sum of M_i Gaussian-shaped kernels, with each kernel centered over its corresponding training sample:

$$p(\mathbf{z}_{bag}) = \frac{1}{M_i} \sum_{j=1}^{M_i} K_G(\mathbf{z}_{bag} - \mathbf{z}_j), \quad (8.28)$$

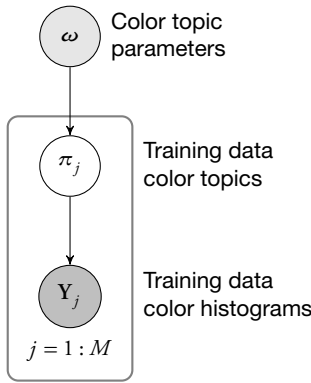


Figure 8.10 Plate diagram for the Dirichlet parameter learning problem, isolated to a single weight vector (corresponding to a particular attribute color).

where K_G represents a multivariate Gaussian kernel of constant size.

Finally, the resulting density is partitioned into a lookup table T_i to dramatically speed up probability evaluation. It is possible to form such a lookup table because the bag position variable spans a relatively low-dimensional space with only four axes (x -position, y -position, width, height), which can be adequately represented by a coarsely partitioned lookup table.

8.3.1.2 Dirichlet Parameters

The other major parameters in the model are the Dirichlet weights associated with each component color specification. We can learn each weight vector ω (corresponding to one row in the α matrix) separately by focusing on one color category at a time. For a given category, we can collect a dataset of any local components within the training images that have been annotated with that color label. The resulting dataset consists of M observed color histograms, one for each component:

$$D = \{Y_j\}_{j=1}^M. \quad (8.29)$$

Figure 8.10 depicts the learning problem for a single ω weight vector. For each of the M training examples, this parameter vector determines the likelihood of generating the latent topic vector π_j , which in turn determines the likelihood of generating the observed histogram Y_j . The training data are only partially observed because there is no way to collect annotations on the hidden topic variables. Therefore, we wish to express the likelihood of observation Y_j directly in terms of parameter ω by marginalizing out

variable π_j :

$$p(\mathbf{Y}_j | \boldsymbol{\omega}) = \int p(\mathbf{Y}_j | \pi_j) p(\pi_j | \boldsymbol{\omega}) d\pi_j. \quad (8.30)$$

Because the two conditional distributions inside this integral have multinomial and Dirichlet forms, respectively, we can substitute the expressions in Equations (8.10) and (8.15), resulting in

$$p(\mathbf{Y}_j | \boldsymbol{\omega}) = \int \left[\frac{100!}{\prod_{k=1}^{N_c} Y_j(k)!} \prod_{k=1}^{N_c} \pi_j(k)^{Y_j(k)} \right] \left[\phi(\boldsymbol{\omega}) \prod_{k=1}^{N_c} \pi_j(k)^{\omega(k)-1} \right] d\pi_j \quad (8.31)$$

$$= \frac{100! \phi(\boldsymbol{\omega})}{\prod_{k=1}^{N_c} Y_j(k)!} \int \left[\prod_{k=1}^{N_c} \pi_j(k)^{Y_j(k)+\omega(k)-1} \right] d\pi_j. \quad (8.32)$$

Integrating out the latent topic variable, as in the above equation, yields a compound Dirichlet-multinomial distribution. The resulting closed-form distribution (sometimes referred to as a *Polya distribution* [13]) is given by

$$p(\mathbf{Y}_j | \boldsymbol{\omega}) = \frac{100!}{\prod_{k=1}^{N_c} Y_j(k)!} \cdot \frac{\Gamma\left(\sum_{k=1}^{N_c} \omega(k)\right)}{\Gamma\left(\sum_{k=1}^{N_c} (\omega(k) + Y_j(k))\right)} \cdot \prod_{k=1}^{N_c} \frac{\Gamma(\omega(k) + Y_j(k))}{\Gamma(\omega(k))}, \quad (8.33)$$

with Γ representing the gamma function.

The objective of the learning process is to find the maximum log-likelihood estimate of parameter $\boldsymbol{\omega}$ with respect to the M independent samples of the training dataset:

$$\hat{\boldsymbol{\omega}} = \arg \max_{\boldsymbol{\omega}} \log p(D | \boldsymbol{\omega}) = \arg \max_{\boldsymbol{\omega}} \sum_{j=1}^M \log p(\mathbf{Y}_j | \boldsymbol{\omega}). \quad (8.34)$$

Substituting the expression from Equation (8.33) and dropping any terms that do not depend on $\boldsymbol{\omega}$ give the final objective function for parameter optimization:

$$\begin{aligned} \hat{\boldsymbol{\omega}} = \arg \max_{\boldsymbol{\omega}} & \left[M \log \Gamma\left(\sum_{k=1}^{N_c} \omega(k)\right) - \sum_{j=1}^M \log \Gamma\left(\sum_{k=1}^{N_c} (\omega(k) + Y_j(k))\right) \right. \\ & \left. + \sum_{j=1}^M \sum_{k=1}^{N_c} \log \Gamma(\omega(k) + Y_j(k)) - M \sum_{k=1}^{N_c} \log \Gamma(\omega(k)) \right]. \quad (8.35) \end{aligned}$$

Because there is no closed-form solution to the above optimization problem, the maximum likelihood estimate of ω must instead be approximated using iterative numerical techniques. The many candidate methods we may apply to this task have different convergence properties and practical advantages. The following two approaches are common:

- Fixed-point iteration techniques like Newton-Raphson optimization, in which an initial estimate is updated repeatedly using the same update rule until convergence. In the case of Newton-Raphson, the update rule is

$$\hat{\omega}^{(n+1)} = \hat{\omega}^{(n)} - \mathbf{H}_{\omega}(\log p(D | \omega))^{-1} \cdot \frac{\partial}{\partial \omega} \log p(D | \omega), \quad (8.36)$$

where $\hat{\omega}^{(n)}$ is the current parameter estimate in the iteration, and $\mathbf{H}_{\omega}$ is the Hessian matrix with respect to ω . The necessary first- and second-order partial derivatives can be derived from the log-likelihood function given in Equation (8.35)).

- Direct search optimization methods like the Nelder-Mead technique or the simulated annealing technique [14]. These iterative procedures do not make use of function derivatives; instead, they rely on heuristics for determining the location of new sample values based on the evaluation of previous samples.

Either approach may require many evaluations of large functionals, which can be computationally costly. However, the learning must only be performed once for a particular training set before the model is used to perform any attribute-based searches.

We may also apply the same learning process in parallel for any other pixel-based feature primitives incorporated into the model, such as the edge-based features discussed in Section 8.2.3.3. Because the conditional distributions for each feature generation branch retain the same form (Dirichlet and multinomial), the only adjustment to the learning process is the formation of different training datasets extracted from the annotated images.

In the end, we have parameter values that reflect the variations in appearance captured in the training dataset. These values will lead the model to perform accurate interpretation of new images that exhibit similar characteristics.

8.3.2 Hidden State Inference

To extract a match score between an attribute profile and an image chip, we use the generative probabilistic model to evaluate the likelihood of these joint observations (i.e., the likelihood that an image exhibiting the specified attributes would generate the observed features). Besides the model parameters, the generative model contains a set of observed variables O and a set of hidden image formation variables H . Figure 8.11 displays the basic model structure along with its partition into observed and hidden variable sets.

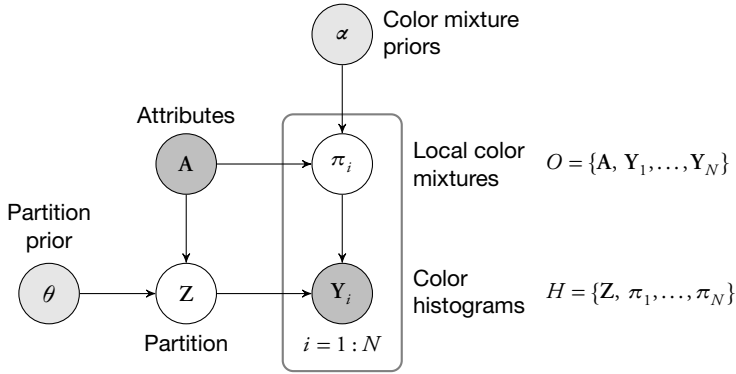


Figure 8.11 Basic generative model structure, partitioned into hidden and observed variables.

One way to evaluate the match score is to compute the marginal likelihood of the observed variables, after integrating out the hidden states:

$$s = p(O) = \int p(O, H) dH. \quad (8.37)$$

Conceptually, this is similar to taking the average likelihood across all possible modes of image formation. The major drawback to this method is the computational cost associated with the marginalization. The integral over the hidden states of the model has no closed-form solution, and numerical integration over a high-dimensional space like this one is computationally intensive.

Instead, we use an alternative match score evaluation based on the maximum likelihood estimate of the hidden variables:

$$s = \max_H p(O, H). \quad (8.38)$$

Conceptually, this is equivalent to finding the single most likely explanation of the data (in the form of latent formation states) and evaluating the model at that point in the state space. This approach turns inference into a more tractable optimization problem because the goal is to maximize the joint probability distribution with respect to the hidden variables.

Although the maximum likelihood approach is more manageable than the marginal likelihood approach, it still does not have a closed-form solution. However, we can find a partial solution by focusing on subsets of the full hidden variable set. First, we factor the joint distribution according to the variable dependencies of the basic model

structure:

$$p(O, H) = p(\mathbf{Z} | \mathbf{A}, \boldsymbol{\theta}) \prod_{i=1}^N p(\pi_i | \mathbf{A}, \boldsymbol{\alpha}) \cdot p(\mathbf{Y}_i | \mathbf{Z}, \pi_i). \quad (8.39)$$

If we assume a fixed estimate for $\hat{\mathbf{Z}}$, then we can optimize over each π_i separately by taking

$$\hat{\pi}_i = \arg \max_{\pi_i} p(O, H) = \arg \max_{\pi_i} p(\pi_i | \mathbf{A}, \boldsymbol{\alpha}) \cdot p(\mathbf{Y}_i | \hat{\mathbf{Z}}, \pi_i) \quad (8.40)$$

subject to

$$\sum_{k=1}^{N_c} \pi_i(k) = 1. \quad (8.41)$$

This assumption simplifies the problem down to finding a single state vector that maximizes the product of two conditional likelihoods. Substituting the definitions of these likelihoods provided in Equations (8.11) and (8.15) gives

$$\hat{\pi}_i = \arg \max_{\pi_i} \left[\phi(\boldsymbol{\omega}_i) \prod_{k=1}^{N_c} \pi_i(k)^{\omega_i(k)-1} \right] \left[\frac{100!}{\prod_{k=1}^{N_c} Y_i(k)!} \prod_{k=1}^{N_c} \pi_i(k)^{Y_i(k)} \right], \quad (8.42)$$

where $\boldsymbol{\omega}_i$ is the row of the $\boldsymbol{\alpha}$ matrix corresponding to the color specification for the i th component, and $\phi(\boldsymbol{\omega}_i)$ is a normalization term.

Because the maximization is with respect to π_i , we can drop any terms that do not depend on this variable and combine the remaining terms to get

$$\hat{\pi}_i = \arg \max_{\pi_i} \prod_{k=1}^{N_c} \pi_i(k)^{Y_i(k) + \omega_i(k) - 1}, \quad (8.43)$$

still subject to the constraint that all elements of vector π_i sum to one.

It is straightforward to derive a closed-form solution to this constrained optimization problem by using Lagrange multipliers, resulting in the following formula for the maximum likelihood estimate of the color mixture states:

$$\hat{\pi}_i = \left[\frac{Y_i(1) + \omega_i(1) - 1}{\sum_{k=1}^{N_c} (Y_i(k) + \omega_i(k) - 1)}, \dots, \frac{Y_i(N_c) + \omega_i(N_c) - 1}{\sum_{k=1}^{N_c} (Y_i(k) + \omega_i(k) - 1)} \right], \quad (8.44)$$

where the color histograms represented by $\mathbf{Y}_i$ are computed according to the partition estimate $\hat{\mathbf{Z}}$. Now that we can derive closed-form estimates for all $\hat{\pi}_i$ based on $\hat{\mathbf{Z}}$, we need a method for optimizing the joint probability distribution with respect to the partition as well.

Because the partition variable essentially controls a selector function for forming feature histograms, the joint likelihood that we wish to maximize is not differentiable with respect to the partition state. This limits the range of applicable numerical techniques for estimating the optimal state value to direct search techniques. Because running an exhaustive search over all combinations of component positions is not practical, we may instead use a form of greedy search referred to as iterated conditional modes (ICM) [15].

First, we set the initial estimate of the partition variable to its most likely a priori value, the mean of the normal distribution given by parameter μ_{base} combined (if necessary) with the maximum likelihood bag position according to the relevant lookup table. Then for each component in turn (full body, head, torso, lower, bag), we search for the value of that component's bounding box that maximizes the joint probability distribution of the model, conditioned on all other component positions being fixed at their current estimates. In this way, ICM takes a relatively high-dimensional search space and breaks it down into a series of more manageable four-dimensional search spaces. Although not guaranteed to converge on the global maximum, this approximate optimization technique tends to work well in practice.

8.3.3 Scoring Algorithm

This subsection outlines the specific steps of the scoring algorithm. First, for convenience, we adjust the definition of the match score to equal the maximum log-likelihood of the joint probability distribution:

$$\hat{s} = \log p(O, \hat{H}). \quad (8.45)$$

Because the log function is monotonic, the maximum likelihood estimates of the hidden states do not change. This log-likelihood can be expressed as

$$\hat{s} = s_0 + \sum_{i=1}^N s_i, \quad (8.46)$$

where

$$s_0 = \log p(\hat{\mathbf{Z}} | \mathbf{A}, \theta) = \log \mathcal{N}(\hat{\mathbf{z}}_{base} | \mu_{base}, \Sigma_{base}) + \log T_{b_t}(\hat{\mathbf{z}}_{bag}) \quad (8.47)$$

is called the *partition score*, and

$$s_i = \log p(\hat{\pi}_i | \mathbf{A}, \alpha) + \log p(\mathbf{Y}_i | \hat{\mathbf{Z}}, \hat{\pi}_i) \quad (8.48)$$

is called the *i*th *component score*.

Substituting the relevant conditional probability definitions into the component score and simplifying gives

$$s_i = - \sum_{k=1}^{N_c} \log Y_i(k)! + \sum_{k=1}^{N_c} (Y_i(k) + \omega_i(k) - 1) \log \hat{\pi}_i(k) + \phi_i, \quad (8.49)$$

where the ϕ_i term is a normalization constant that does not depend on the image being evaluated and therefore has no significance in the scoring process. The goal of the scoring algorithm is to maximize the sum of the partition score and all component scores with respect to the hidden state estimates.

Algorithm 8.1 details the steps of the match score computation algorithm. First, the algorithm sets an initial estimate of the partition and the corresponding maximum likelihood estimates of the color topic states; these are used to set the initial values of $\hat{s}_0$ and $\hat{s}_1$ through $\hat{s}_N$. Until convergence or the maximum number of iterations is reached, the algorithm cycles through the head, torso, lower body, and bag positions and updates the estimate for $\hat{z}_i$, the corresponding bounding box within the partition vector. The position is only updated if the algorithm can find a value within a local spatial neighborhood of the existing estimate that leads to a higher log-likelihood over all hidden variables (including updated color topic states). When all updates are complete, the final match score is the sum of the resulting partition and component scores.

It is straightforward to revise the scoring algorithm for the model extensions discussed in Section 8.2.3. To incorporate gender, we simply add an extra term to the final match score by evaluating the conditional probability in Equation (8.17) because this does not depend on the estimation of any hidden states. To incorporate other feature primitive types (such as edges), we redefine component score s_i as a sum of log-likelihoods across the conditional probability chains of each feature type. The algorithm flow remains the same.

Figure 8.12 illustrates the iterative process of hidden state estimation. In this example, the observed image is paired with a matching attribute profile so that the resulting latent state estimates successfully reflect image formation. The figure shows sample partition estimate values during the optimization process from initial value (far left) to convergence (far right). Note that the partition state starts at the average bounding box across all images (learned during the training process).

As ICM optimization progresses, the individual component positions begin to migrate toward better-fitting bounding boxes, in which a better fit equates to an increase in the joint probability of the partition and the maximum likelihood latent topic vectors. Using this technique, when the image chip represents a match to the attribute profile, the estimation process tends to significantly increase the final optimized score (as the algorithm finds a good “explanation” of the data according to the probabilistic model); however, estimation does not typically improve the score much when the image is

Algorithm 8.1 Scoring based on iterative optimization

-
- 1: **Input:** attribute profile, image, model parameters
 - 2: **Output:** $\hat{s}$, estimated match score
 - 3: Initialize partition estimate: $\hat{\mathbf{z}}_{base} \leftarrow \boldsymbol{\mu}_{base}$, $\hat{\mathbf{z}}_{bag} \leftarrow \arg \max_{\mathbf{z}_{bag}} T_{b_t}(\mathbf{z}_{bag})$
 - 4: Set partition score $\hat{s}_0$ based on $\hat{\mathbf{Z}}$ (Equation (8.47))
 - 5: **for** $i \leftarrow 1$ **to** N
 - 6: Compute histogram $\mathbf{Y}_i$ based on initial partition $\hat{\mathbf{Z}}$ (Equation (8.14))
 - 7: Estimate latent topic vector $\hat{\boldsymbol{\pi}}_i$ based on $\mathbf{Y}_i$ (Equation (8.44))
 - 8: Set component score $\hat{s}_i$ based on $\hat{\boldsymbol{\pi}}_i$ (Equation (8.49))
 - 9: **repeat**
 - 10: **for** $i \leftarrow 1$ **to** N
 - 11: **for** positions $\mathbf{x} \in \text{Neighborhood}(\hat{\mathbf{z}}_i)$
 - 12: Set modified partition score s_0 based on $\mathbf{x}$ (Equation (8.47))
 - 13: Update histogram $\mathbf{Y}_i$ based on candidate position $\mathbf{x}$ (Equation (8.14))
 - 14: Estimate latent topic vector $\hat{\boldsymbol{\pi}}_i$ based on $\mathbf{Y}_i$ (Equation (8.44))
 - 15: Set modified component score s_i based on $\hat{\boldsymbol{\pi}}_i$ (Equation (8.49))
 - 16: **if** $s_0 + s_i > \hat{s}_0 + \hat{s}_i$
 - 17: $\hat{\mathbf{z}}_i \leftarrow \mathbf{x}$
 - 18: $\hat{s}_0 \leftarrow s_0$
 - 19: $\hat{s}_i \leftarrow s_i$
 - 20: **until** $\hat{\mathbf{Z}}$ has converged or maximum number of iterations is exceeded
 - 21: $\hat{s} \leftarrow \hat{s}_0 + \sum_{i=1}^N \hat{s}_i$
-



Figure 8.12 Illustration of the iterative hidden state estimation for a sample image (and matching attribute profile). Far left: initial estimate of the partition state. Far right: final converged estimate of the partition state, leading to an increased match score. Not pictured: corresponding estimates of the hidden mixture states $\{\pi_i\}_{i=1}^N$, which determine the conditional probability of the observations within each component.

a mismatch. The behavior of the scoring algorithm in either case is critical to the effectiveness of this approach.

8.4 Performance

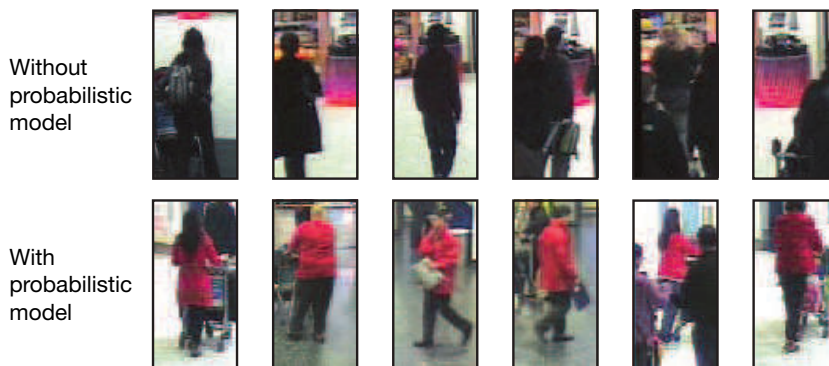
To gauge the value of the generative model approach, we may apply this technique to attribute search tasks within multiple video surveillance environments. Viewing the results across a range of search requests and comparing to ground truth information about video content can give us a quantitative and qualitative sense of performance. In this section, we apply the proposed search technique to the following two representative data sources to characterize its performance:

- Two hours of surveillance video collected at Gatwick Airport in London [8], distributed across three cameras at different locations within one terminal of the airport. There is significant variation in the density of the observed crowds, with regular arrivals causing spikes in the activity level. Applying the moving-person detection process described in Section 8.1.2 on this dataset at a rate of 1 Hz results in a database of approximately 14,000 detected instances of moving persons.
- A dataset of surveillance video collected at a major U.S. airport. This dataset consists of about five total hours of video from six different camera views, arranged within one terminal of the airport. Applying the detection process at 1 Hz results in approximately 175,000 instances of detected moving persons.

Experimentation with these datasets helps to reveal two aspects of performance critical to practical application of the attribute-based search technique. The first is search accuracy, which impacts the ability of the operator to successfully find the person of interest when reviewing results. The second is search timing, or the duration of the required score computations, which must be relatively short to avoid long wait times once an operator has initiated the search.

8.4.1 Search Accuracy

Figure 8.13 shows examples of search results, using one lower specificity attribute profile and one higher specificity attribute profile. For each search case, the top row depicts the best six matches according to a simple (nonprobabilistic) color-matching method. In this method, the color class labels for all the foreground pixels of the image chip are accumulated into a single color histogram. The histogram is scored based on its similarity to the target histogram, which is formed from the colors in the attribute description. In other words, the simple approach searches for the right mixture of foreground colors without using a probabilistic model to account for the hidden states of image formation. As can be seen in the figure, this approach does not typically return good matches to the profile, often picking up on image regions unrelated to detected moving persons.



(a) Red torso / black lower body.



(b) Black and white torso / dark blue lower body / red luggage.

Figure 8.13 Examples of color-based attribute search results for both a simple color-matching heuristic and the probabilistic generative model technique. The simple heuristic tends to result in poor matches, but the probabilistic model does better at matching images to specified attributes.

The bottom row of each search case in Figure 8.13 shows the results from using the probabilistic generative model described in this chapter. Because the model can account for unobserved factors such as the spatial arrangement and component-specific distributions of color and edge primitives, it does significantly better at finding accurate matches to the description. For the first, less specific search case, five of the six top results are complete matches, whereas the other result is a partial match (resulting from the presence of other pedestrian clutter). For the second, more specific search case, the model returns a single significant match (the only result scoring above some minimum threshold), which is the only correct match in the data. Qualitatively, this comparison demonstrates the value of using a probabilistic generative model over simple color-matching approaches for the evaluation of matches. However, examples of this sort do not give a quantitative measure of performance.

To get a quantitative characterization of search performance, we select a particular area surveilled during one of the airport video data collections and label ground truth for all pedestrian activity occurring within that area over a fixed time span. The surveillance video of this (approximately 50 square meter) region has image resolution along the height of pedestrians ranging from about 80 pixels to 120 pixels, which is sufficient for accurate moving person detection. Labeling the location and attribute-based appearance of each person who passes through this region leads to about 1000 frame appearances of 150 unique individuals. We discuss several metrics of performance on this dataset.

8.4.1.1 Detection

For each person passing through the region of interest of the labeled dataset, we count a correct detection if the detection algorithm described in Section 8.1.2 flags that person in at least one analyzed frame of video. We count any detection that does not correspond to an entire person as a false positive. False-positive sources in the test environment include shadow effects, parts of pedestrians, and luggage carts.

When we used threshold values that are optimized for crowded indoor environments, the observed probability of detection is 145 out of 150 (or approximately 97%), with the missed detections due to either constant occlusion or fleeting intersections with the region of interest. The corresponding false-positive rate is about once per 200 seconds.

In practice, the detection algorithm supports good search capability because it finds at least one instance of almost every pedestrian who passes through the scene with an unobstructed view to the camera while pulling in false positives at a low enough rate that they constitute a small percentage of records in the detection database. It is worth noting that the detection performance depends heavily on the characteristics of the scene, particularly the density of the observed crowds and the resolution of the imagery.

8.4.1.2 Appearance Model

The probabilistic appearance model provides a mechanism to score the likelihood that each image chip depicts a person with the specified attribute set. When the model functions correctly, all examples of persons matching the provided description will appear at the top of the match list, which is ranked in order of descending likelihood scores.

We may evaluate the accuracy of the model by running multiple sample searches over the labeled dataset and comparing the actual results to the expected results. The legend of Figure 8.14 lists a set of attribute profiles for 11 such searches, selected at random by taking descriptions from the ground truth labels that match one or more persons who appear within the video.

For each test search, a performance curve by varying the number of top results returned by the search. Raising the number of results increases the chance of finding all true matches, but it also increases the occurrence of false positives or returned image chips that do not fully match the specified profile. The vertical axis in Figure 8.14 plots *recall*, or the percentage of all true matches returned by the search, whereas the horizontal axis plots the number of returned false positives, normalized by the total number of false-positive individuals in the database. Note that by these metrics, an algorithm that assigns random match scores to image chips will have an expected performance represented by the dotted line in Figure 8.14.

As expected, all search results using the proposed model perform significantly better than the random-scoring baseline. However, there is noticeable variation in error rate depending on the particular attribute profile. Five of the 11 sample searches (represented by the red line in Figure 8.14) find all true matches before returning any false positives and are therefore examples of perfect search accuracy. Other searches return multiple false positives before recovering all matches.

The more specific search queries, especially those with only one true match among all pedestrians, tend to show better results because these profiles contain the most discriminating information. More generic descriptions tend to pull in some false positives along with all of the true positives. Because the scoring algorithm produces a real-valued match score related to the probability of the observation, the false-positive results (which have relatively high scores) still tend to be close matches to the attribute profile, often deviating in minor ways from the description.

8.4.2 Search Timing

To have operational relevance, an attribute-based search must execute in a reasonable amount of time. Because the search is not finished until a match score has been assigned to every candidate image chip, the time to execute a search is directly related to the

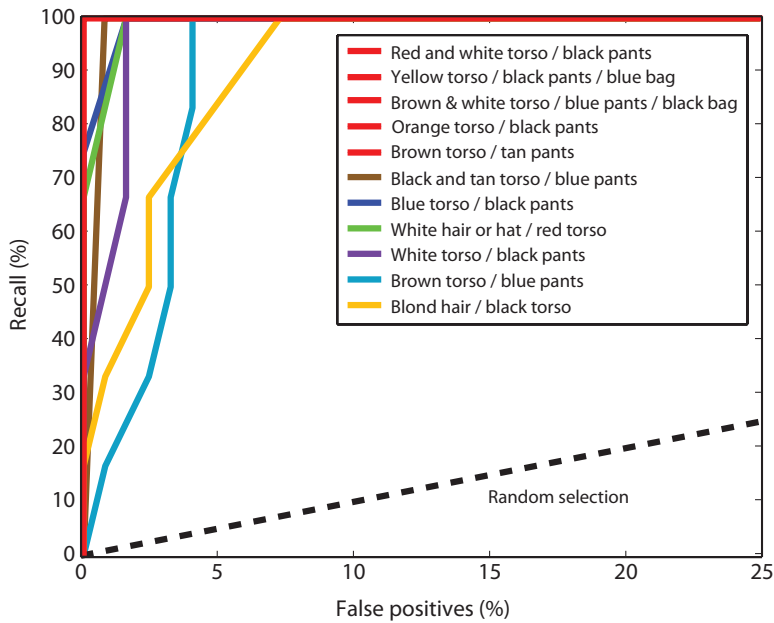


Figure 8.14 Attribute-matching accuracy across 11 search examples. The dotted line represents expected performance of a random selection algorithm. The first five searches in the legend have perfect accuracy.

Table 8.2 Processing times by dataset.

Task	Gatwick Airport	U.S. Airport
Average score computation time	57 μ s	50 μ s
Average search time per camera hour	0.4 s	1.75 s

average time to perform an individual match score computation. This operation must be performed quickly, because a typical search may involve tens or hundreds of thousands of candidates from the detection database (if not more).

While the scoring algorithm outlined in Algorithm 8.1 is designed to converge quickly to an approximate optimization of the hidden states of the model, we can take the following steps in the implementation of this algorithm to help ensure sufficiently fast computation:

- Perform a preliminary check to see whether the feature primitives representing the image chip are at all close to the expected observations corresponding to the attribute profile. For example, if the local color histograms diverge significantly from any of the colors specified by the attribute profile, we can automatically assign a low match and refrain from executing the hidden state optimization procedure under the assumption that no hidden state values will result in a sufficiently high match score to display it to the operator. For many search criteria, this preliminary check reduces the total number of required match score computations by more than one half.
- Eliminate repetitive evaluations of relatively expensive functions, like the log-factorial in Equation (8.49). Because in this application the log-factorial can only be evaluated on the set of integers from 0 to 100, it is much more computationally efficient to precompute these values and store them ahead of time.
- Impose a reasonable limit on the number of iterations of the optimization algorithm. Limiting to a few iterations is typically sufficient for the estimation to near convergence while preventing superfluous iterations that have relatively minor effects on the final match score.

Table 8.2 gives the average score computation times for both airport surveillance datasets. These times were averaged across many attribute search trials because processing time is partially a function of the attribute profile. For either dataset, the average time required for a single score computation is about 50 μ s; therefore, up to 20,000 images can be scored every second.

For each dataset, Table 8.2 also shows the mean time required to process an hour of detections from one camera view. For the Gatwick dataset, which has a lower density of pedestrian flow (and therefore fewer detections overall per hour of data), it takes less than

half a second to search one camera hour of video. For the U.S. airport dataset, which captures significantly more pedestrians within relatively wide camera views, it takes close to two seconds per camera hour. However, in either case, the scoring algorithm works efficiently enough to perform searches over tens of hours of camera data, and millions of records, within a minute or two.

8.5 Interactive Search Tool

The concept of probabilistic attribute-based image evaluation is most useful when incorporated as part of an interactive search capability. In this section, we describe one implementation of such a tool and show examples of its functionality on sample video data. The tool connects directly to a database of moving-person detections (created automatically as incoming video is analyzed) and provides a way to search over those detections.

The tool has the following functionality:

- It allows the operator to input a set of search criteria through a graphical user interface, including both the attribute profile and the time and location constraints of the search.
- It retrieves all records from the detection database that fall within the specified time period and location constraints.
- It applies the probabilistic model described in Section 8.2 to score how well each record matches the specified attribute profile.
- It ranks the match results in order of descending score to extract the top N matches from the database (as long as the match scores exceed some baseline threshold).
- It filters the list of top-scoring matches by using a form of non-maxima suppression to eliminate redundant appearances of the same person in consecutive frames of video.
- It displays the top matches back to the operator as a set of browsable image chips.
- Finally, it allows the operator to review associated video by selecting individual image chips from the top match set.

The resulting system provides an effective way for operators to interactively explore the database of observed pedestrians within a facility by conducting a series of search and video review steps.

Figure 8.15 shows a screenshot of the search tool. Launching a search brings up a search criteria input menu, divided into an attribute profile section (upper half) and a search localization section (lower half). Within the attribute profile section, the operator may specify any subset of attribute options represented in the menus; any inputs not provided by the operator are left as unspecified by default, and they will not factor into the search results.

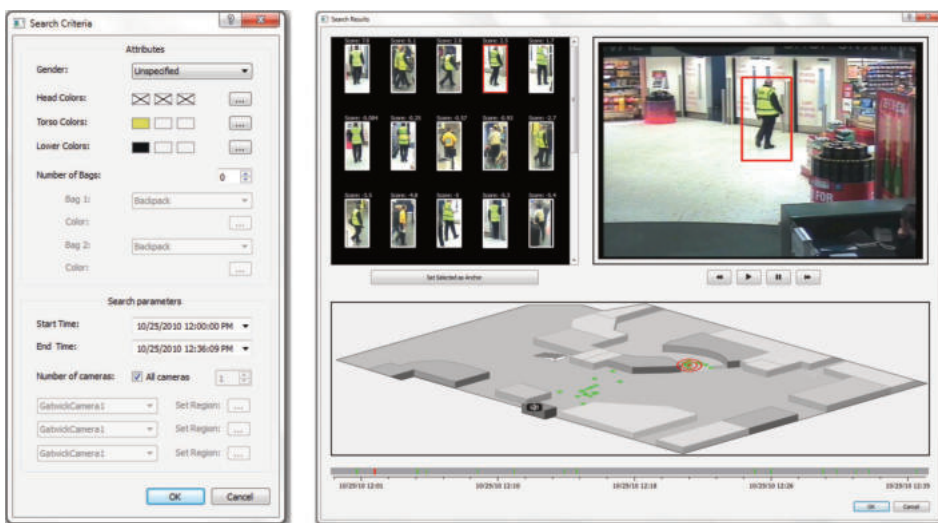


Figure 8.15 Sample screenshot of the interactive search tool. Left: Search criteria input interface, in which the operator has specified yellow shirt color and black pants. Right: Search results exploration interface, displaying top matches for these search criteria. Image chips are arranged in order of descending match score, from left to right and top to bottom. This search was performed over approximately 30 minutes of data from an airport dataset.

In addition to the attribute profile, the operator selects a start time and end time for conducting the search, which may span up to multiple days of archived video. The operator may also choose the subset of camera views over which to search, focusing on particular regions within individual camera scenes (e.g., doorways or pedestrian passageways) if desired. Once all attribute and search localization criteria have been entered, the search process launches. The time required to finish this processing is typically at least a few seconds and scales in rough proportion to the area and time window of the search and the density of pedestrian traffic.

The search results interface has three panels, as depicted on the right side of Figure 8.15. The first panel contains a set of image chips representing the top matches to the attribute search criteria. Although these chips are filtered to avoid redundant displays, the same individual may appear multiple times with some separation in time and space between the appearances. The operator may scroll through these image chip results and click on any of them to retrieve the corresponding video content.

The second panel displays the video frame from which the selected match was detected and allows the operator to view the video with a standard set of playback controls. The third panel displays both a timeline and a map of the facility and contains estimated times and locations for each of the top matches. The operator has the option to retrieve video by selecting an individual detection directly from the map or from the timeline, in addition to selecting from the image chip set.

In practice, there is no guarantee that all search results will be successful matches to the attribute description. However, due to the probabilistic scoring process, those results that are not complete matches often tend to be *near* matches, in which only one aspect of the attribute profile is a mismatch. There is also no guarantee that the most relevant observation to the analyst will appear as the first (most highly scored) image chip because of limitations in the type of attribute inputs accepted by the software or limitations of the generative appearance model. However, if the useful content appears somewhere among the top match results, then it is typically a much easier and quicker process for the operator to browse through a set of image chips and focus in on the relevant content than it is for that operator to manually scan large amounts of raw video.

8.6 Summary

Video search based on content descriptions is a useful capability for multiple surveillance scenarios. First, this capability can be used forensically to search through archived video and recover observations related to a witness report or incident investigation. Second, this capability can be used to monitor live video feeds for apparent matches to a given description. In either case, automating the search process allows an operator to sort through large quantities of video data and invest time and attention in only the most relevant segments.

This chapter discussed a particular type of content search based on the attributes of a person and associated clothing and bags observed at a distance. Although this type of video interpretation can typically be performed at high accuracy by human visual perception, it is difficult to automate this process because of the variations in appearance for a given description. In fact, there are many different manifestations (in pixel values) for observations that match an attribute profile depending on factors such as view angle, lighting conditions, and clothing composition.

Solutions based on probabilistic modeling approaches are a good fit for this problem because they address the uncertainty of the observations introduced by the ambiguous high-level descriptions. Such models offer a mechanism to characterize the nature of the uncertainty and structure it according to key latent states of the interpretation problem. We reviewed an example of a generative appearance model that defines a joint probability distribution over the observed pixel-level features and a collection of hidden states of image formation, conditioned on the attribute profile. The hidden states of the model can be inferred quickly using approximate optimization techniques, leading to a more accurate assessment of the degree of match than if these states were not considered during evaluation. In addition, the parameters of the generative model are learned directly from (partially labeled) data.

Experimental trials using sample video from several environments demonstrate the value of employing such a model. Simple color-matching methods do not perform well because they do not adequately account for the expected joint distribution of the observation set, making them susceptible to poor image interpretations. However, scoring based on evaluations of a probabilistic model works well in these trials. When coupled with a search interface, it enables timely and accurate interactive searches on test video from airports.

However, the performance of any probabilistic video-interpretation method will depend on the characteristics of the scene under surveillance and the conditions of video capture. Common challenges that lead to failure include the following:

- Poor resolution on the scene components of interest (which typically require on the order of hundreds of total pixels to resolve attribute-based details) or insufficient lighting to record scene components at reasonable contrast with the background. Accurate interpretation cannot be performed if the necessary information is not captured by the surveillance system.
- High crowd density or occlusion levels. Moving-pedestrian detection becomes difficult when the camera has only a heavily obstructed view of the pedestrians that prevents the creation of a comprehensive database of person observations over which search can be performed. Detection is especially problematic when the video depicts a solid mass of people moving through the scene, denying a clear line of sight to each person. In these cases, it is sometimes reasonable to rely on

video analysis at other locations within the same facility where crowd levels are more dispersed.

- Significant differences in clothing style or other appearance factors from those represented in the training set. For instance, the expected distributions over observed features is somewhat dependent on local styles of dress and weather conditions. Because a misalignment between the model and the evaluated observations can lead to degraded image interpretation performance, one way to solve this problem is to repeat the model learning process using a more representative dataset.

Most challenges can be addressed by strategic selection of the camera views on which to perform video analysis or through refinements to the probabilistic model structure and parameters.

Finally, the approach outlined in this chapter can be extended to the problem of attribute-based search of other scene component types besides pedestrians. Useful search capabilities for security personnel include those based on vehicle descriptions (color and type); high-fidelity luggage, bag, or package descriptions; or facial attributes captured in close-proximity video. In any of these cases, probabilistic models can be used to effectively represent and manage uncertainty during the image interpretation task.

References

1. J. Vlahos, “Surveillance Society: New High-Tech Cameras Are Watching You,” *Popular Mechanics*, vol. 139, no. 1, pp. 64–69, 2008.
2. N. Gagvani, “Challenges in Video Analytics,” in *Embedded Computer Vision*, B. Kisačanin, S.S. Bhattacharyya, and S. Chai, eds., London: Springer, 2009.
3. J. Thornton, J. Baran-Gale, D. Butler, M. Chan, and H. Zwahlen, “Person Attribute Search for Large-Area Video Surveillance,” in *IEEE International Conference on Technologies for Homeland Security*, 2011.
4. P. Phillips, P. Flynn, T. Scruggs, K. Bowyer, J. Chang, K. Hoffman, J. Marques, J. Min, and W. Worek, “Overview of the Face Recognition Grand Challenge,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005.
5. J. Boyd and J. Little, “Biometric Gait Recognition,” in *Advanced Studies in Biometrics: Revised Selected Lectures and Papers*, M. Tistarelli, J. Bigun, and E. Grosso, eds., Berlin: Springer, 2005.
6. M. Demirkus, K. Garg, and S. Guler, “Automated Person Categorization for Video Surveillance Using Soft Biometrics,” in *SPIE Biometric Technology for Human Identification*, 2010.

7. J. Ferryman and D. Tweed, “An Overview of the PETS 2007 Dataset,” in *IEEE International Workshop on Performance Evaluation of Tracking and Surveillance*, 2007.
8. K. Sage, A. Nilski, and I. Sillett, “Latest Developments in the ILids Performance Standard: New Imaging Modalities,” *IEEE Aerospace and Electronic Systems Magazine*, vol. 25, no. 7, pp. 4–11, 2010. DOI: 10.1109/MAES.2010.5546288.
9. D. Blei, A. Ng, and M. Jordan, “Latent Dirichlet Allocation,” *Journal of Machine Learning Research*, vol. 3, pp. 993–1022, 2003.
10. L. Fei-Fei and P. Perona, “A Bayesian Hierarchical Model for Learning Natural Scene Categories,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005.
11. N. Dalal and B. Triggs, “Histograms of Oriented Gradients for Human Detection,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005.
12. J. Daugman, “Uncertainty Relation for Resolution in Space, Spatial Frequency, and Orientation Optimized by Two-Dimensional Visual Cortical Filters,” *Journal of the Optical Society of America*, vol. 2, no. 7, pp. 1160–1169, 1985. DOI: 10.1364/JOSAA.2.001160.
13. C. Elkan, “Clustering Documents with an Exponential-Family Approximation of the Dirichlet Compound Multinomial Distribution,” in *International Conference on Machine Learning (ICML)*, 2006.
14. S. Kirkpatrick, C.G. Jr., and M. Vecchi, “Optimization by Simulated Annealing,” *Science*, vol. 220, no. 4598, pp. 671–680, 1983. DOI: 10.1126/science.220.4598.671.
15. J. Huang, Q. Morris, and B. Frey, “Iterated Conditional Modes for Crosshybridization Compensation in DNA Microarray Data,” University of Toronto, Tech. Rep. PSI TR 2004-031, 2004.

9

Dynamic Models for Speech Applications

Pedro A. Torres-Carrasquillo

This chapter provides an overview of speech applications and the role of basic modeling and decision techniques in these applications. We will focus on the area of speech recognition involving extracting information from speech. Classically, the main piece of information extracted from a speech signal is the sequence of words. In this chapter, we will describe the basic speech recognition application as well as other concepts such as gender, language, and speaker recognition. We will expand on the description of hidden Markov models (HMMs) and Gaussian mixture models (GMM) presented in Chapter 2 and discuss how they are typically employed in various speech applications.

9.1 Modeling Speech Signals

Speech processing is the term used to refer to the process by which a speech signal produced by a human is analyzed by a computer or another automated system to extract information of interest. The process of extracting information from a speech signal is usually conducted by performing short-term frequency analysis of the signal and modeling these frequency components of the signal. These components can be modeled directly and used by a pattern recognizer, e.g., a dynamic model like an HMM, or used as a building block for more linguistically significant units like phonemes. These linguistically significant components can then be used to model even higher level units such as words or sentences. Table 9.1 presents a hierarchy of speech units or tokens used for different applications along with their typical time span.

At a high level, the speech production mechanism is usually modeled following the source-filter paradigm [1], [2]. An excitation signal drives a filter that shapes the incoming signal and produces the speech signal. The speech signal produced is highly dynamic with local and nonstationary statistics. The nonstationary characteristics of the speech signal result in a number of assumptions needed to model the signal. For example, short-term analysis is employed to exploit the short-term stationarity of the signal.

Table 9.1 Hierarchy of speech units.

Unit	Time span
Spectral (cepstral based)	10 to 20 ms
Prosodic	50 to 100 ms
Phonetic	100 ms
Idiolectal (phones)	> 200 ms
Words	> 200 ms
Idiolectal (words)	> 400 ms
Semantic	> 1 s

The speech signal conveys information at various levels, ranging from the smallest unit, the phoneme, to the level of words and sentences. Additionally, the signal carries rhythm information and information about the characteristics of the individual speaker. In the following sections, we will describe how statistical modeling of the speech signal is typically done and applications that exploit these different levels of information to extract the information of interest.

There are many applications within the field of speech processing. These applications include coding of the speech signal, information extraction from the speech signal, and speech-to-speech translation. In this section, we limit our discussion to speech recognition and information extraction. The process for conducting speech recognition and information extraction includes feature extraction, classification, and decision making.

9.1.1 Feature Extraction

Although many features have been used in the field of speech processing, discussing all of these is beyond the scope of this book. We focus on the most widely used set of features within the field of speech recognition, the cepstral features. A cepstral feature is defined as the inverse Fourier transform of the logarithm of the Fourier transform of the signal. In the case of a speech signal, we are usually referring to a windowed version of the signal or the short-term Fourier transform. The most common variant of the cepstral feature set is the mel-scale cepstral feature set [3], [4].

9.1.2 Hidden Markov Models

Hidden Markov models (HMMs), as defined in Section 2.1.7, are stochastic models that allow for the inclusion of temporal information. HMMs have been used for a large number of applications within the speech processing field, particularly for recognition systems [5], [6].

HMMs are extremely flexible and allow for modeling the highly variable components observed in a speech signal. In the case of speech recognition, a left-to-right, three-state HMM architecture is commonly used. This architecture allows for either a return to the current state or a transition to the state to the right of the current state. The generation of an observation is typically modeled by a probability distribution with a Gaussian or mixture model. An HMM requires defining the set of transition probabilities among states, the observation distribution associated with each state, and an initial state distribution.

Once the architecture and parameter set that define the HMM are decided, we typically focus on the three basic problems of interest for data modeling:

- adjusting the parameters with respect to the training data,
- computing the most likely state sequence, and
- computing the likelihood of an observation sequence given the model.

A detailed description of each of these problems is beyond the scope of this chapter, but a number of algorithms are commonly used to achieve a solution either in the maximum likelihood sense or using discriminative techniques [7]–[10].

9.1.3 Gaussian Mixture Models

As introduced in Section 2.1.2, a Gaussian mixture model is a representation of a continuous probability distribution composed of a set of M normal components. The density is given by

$$p(\mathbf{x} \mid \mu_1, \Sigma_1, \dots, \mu_M, \Sigma_M, \alpha_1, \dots, \alpha_M) = \sum_{i=1}^M \alpha_i \mathcal{N}(\mathbf{x} \mid \mu_i, \Sigma_i). \quad (9.1)$$

The parameters of the model are defined by $\theta = (\mu_{1:M}, \Sigma_{1:M}, \rho_{1:M})$. One way to generate a sample from a GMM is to select one of the M components according to the distribution defined by parameters $\rho_{1:M}$ and then sample from the distribution associated with that component.

GMMs have an important role within the recognition component of speech processing. First, the GMM is typically the distribution of choice for the observation set in an HMM. Additionally, the GMM technique has been the leading approach in the area of speaker recognition for more than a decade. More recently, GMM-based language identification has been shown to provide excellent performance. GMM-based recognition has been used over the last few years to conduct depression classification and speech activity detection.

9.1.4 Expectation-Maximization Algorithm

The training for the GMM is commonly performed by the expectation-maximization (EM) algorithm [9]. The EM algorithm is widely used across multiple fields and for a variety of problems. The EM algorithm has a number of desirable properties including convergence to at least a local maximum and computational simplicity. Other methods, such as gradient descent, tend to be computationally expensive.

EM is generally concerned with estimating the parameters of a probabilistic model when there are missing data or unobserved variables. In the context of learning the parameters of a GMM, the missing data might be the component used to generate the sample. If the observed dataset is X and the unobserved dataset is Y , then EM aims to find the model parameter θ that maximizes $P(X, Y | \theta)$ —or, more commonly due to computational convenience, $\log P(X, Y | \theta)$. The process outlined in Algorithm 9.1 begins with an initial guess of θ , denoted $\hat{\theta}$. Given $\hat{\theta}$, the algorithm computes

$$\arg \max_{\theta} \sum_Y P(Y | X, \hat{\theta}) \log P(Y, X | \theta) \quad (9.2)$$

and sets $\hat{\theta}$ to this value for use in the next iteration. The process repeats until convergence.

Algorithm 9.1 Expectation-maximization estimation

```

1: function EXPECTATIONMAXIMIZATION
2:   Initialize  $\hat{\theta}$ 
3:   loop
4:      $\hat{\theta} \leftarrow \arg \max_{\theta} \sum_Y P(Y | X, \hat{\theta}) \log P(Y, X | \theta)$ 

```

If the distributions involved are from the exponential family, as is the case with GMMs, then there is a closed-form solution to Equation (9.2). Bilmes provides an overview of how to compute the parameters of a GMM using EM [11]. When applying EM to GMMs, the covariance matrix for each mixture component is often assumed to be diagonal. Diagonal matrices tend to work well for most speech applications and simplifies the computation. The parameter $\hat{\theta}$ can be initialized in different ways, such as random assignment or k -means clustering.

9.2 Speech Recognition

Speech recognition is the process by which computers analyze a speech signal and produce a sequence of words, a transcript, of the spoken signal. The incoming speech signal is converted into a stream of feature vectors as described earlier. The resulting stream of feature vectors is usually decoded into a set of units or tokens with linguistic meaning.

These tokens tend to be either phones or sub-word units obtained via acoustic decoding. Typically, the acoustic decoding is combined with additional linguistic knowledge about the language of interest. This additional knowledge is used to constrain the possible set of acoustic sequences.

Speech recognition involves the application of Bayes' rule to various pieces of information, including the set of words W , also known as the language vocabulary, and the stream of feature vectors X produced for the speech signal. Speech recognition produces a mapping from the input acoustic sequence, $X = x_{1:n}$, to the set of words, $W = w_{1:m}$, where n is usually much larger than m .

HMMs are the model of choice used for speech recognition. Typically, an HMM is created for all the phones in a given language, and the phone-HMMs are combined to produced word-HMMs via forward-backward estimation. Decoding of the observation X into words is computationally expensive, and various algorithms are available to perform this process. The Viterbi algorithm is conventionally employed for this task [5], [12]. The acoustic model is commonly trained on hours of data spanning multiple pronunciations of different words and phones.

The decoding process incorporates linguistic knowledge about the language in two ways. First, the pronunciation is modeled to constrain the different alternatives over which words in the language of interest can be pronounced. This step is critical in the search process because it reduces the number of viable decoding alternatives and reduces the computation required. The pronunciation model is usually built by including all the likely canonical pronunciations for a word in a given language. This model is also known as the pronunciation dictionary.

Second, the language model constraints the potential word sequences likely to be observed in a language. The language model also plays a large role in the search process to determine the hypothesized word sequences. The language model is trained by computing the probabilities of word sequences across a large corpus. Traditionally, trigrams (sequences of three words) are employed as a trade-off between the amount of training data needed to robustly estimate the sequence statistics and the discriminative potential of the language model. In the case of the language model, hundreds or even thousands of hours are used to train the language model. Mathematically, this process is usually conducted as follows:

$$\arg \max_W p(W | X) = \arg \max_W \frac{p(X | W)p(W)}{p(X)} = \arg \max_W p(X | W)p(W). \quad (9.3)$$

The pronunciation and language model are incorporated using

$$\arg \max_W p(W | X) = \arg \max_W p(X | A)p(A | W)p(W), \quad (9.4)$$

where $p(X | A)$ is the set of HMMs modeling the acoustics of the language, $p(A | W)$ is the pronunciation model for the acoustic sequence given all possible pronunciations, and $p(W)$ is the language model constraining the sequences likely observed in a language.

Advances in speech have come at a consistent pace, incorporating knowledge across multiple research fields. Multidisciplinary ideas from psychology, linguistics, computer science, and pattern recognition have proved critical to improve performance over the last few decades. Usually, advances are obtained by constraining the conditions over which a problem is solved and then relaxing these conditions as improved performance is obtained. A common performance metric in the field of speech recognition is word error rate. Word error rate is usually measured as the addition of the three possible types of error observed. These errors include words that are deleted compared to the reference, words that are added to those in the reference, and words that are substituted when compared to the reference. The errors are obtained by aligning the reference (known) transcript to the one obtained from the system via dynamic programming.

A number of excellent sources address the issues related to speech recognition algorithms [13], [14]. In this chapter, we have not discussed a number of important issues related to how computational issues are handled and how words that are never observed during training are handled (out-of-vocabulary words). A large number of techniques have been developed over the years to reduce the search space over which models are decoded. Most of these techniques include pruning viable alternatives by eliminating the least promising decoding paths.

There are several current trends in automated speech recognition research. First, as additional computing power becomes available, the trend is to include more complex models with a large number of parameters. For example, many state-of-the-art systems involve HMMs with millions of parameters for the observation distribution. Second, the use of discriminative techniques for training the models is typical. In the case of discriminative techniques, the emphasis usually is to optimize the decision region instead of focusing on maximizing the likelihood of the models. Third, new techniques dealing with neural networks are emerging. One of the most recent alternatives being studied in the field is the use of deep neural networks [15]. Deep neural networks exploit complex neural network architectures and training techniques to address the speech recognition problem. A fourth area of interest is the extension to additional languages. Given the need of systems to be trained on large amounts of transcribed data, work is underway to obtain data-driven units that can be used for applications in low-resource languages [16], [17]. In the case of low-resource languages, limited amounts of training data are available, and the interest is to explore bootstrapping data from additional richer languages.

9.3 Topic Identification

Topic identification is the process of identifying the topic in a spoken document. Over the years, a number of alternatives have been considered for performing the topic identification tasks. Currently, most of the techniques have a speech recognition preprocessing stage for extracting words or sub-word units. Each of these methods has advantages and disadvantages. In the case of word-based systems, we tend to be able to use specialized vocabulary that could be of interest for a particular application at the cost of higher computational complexity and limited language coverage. In the case of phonetic systems, we tend to have better flexibility for adapting to newer conditions and languages at the cost of more errorful tokenization.

The output of the system, in the form of words or phone sequences, in the preprocessing stage is then processed to obtain a set of features that can be useful for topic identification. A widely used method for using the speech recognition output is the bag-of-words approach. The bag-of-words represents a vectorized count of words (or tokens) observed in the decoded speech. Typically, the set of words is filtered to eliminate words that are not descriptive of the content in the spoken document. The filtered set of words includes elements such as filled pauses (e.g., ah, uh, um) or nondescriptive words (e.g., the, of, that).

The final classification stage consists of a measurement of distance between the decoded input speech vector and a set of representative vectors of the topics of interest. In this stage, techniques range from simple distance metrics like cosine distance to more involved methods like inverse document frequency [18] and latent semantic analysis [19]. The inverse document frequency technique relies on the concept that the most frequent words or tokens are not likely to carry as much discriminative power as those units that are not observed as frequently, and a combination of a set of these infrequent units is likely a good indicator of the topic. Latent semantic analysis attempts to constrain the variability in the feature space to a smaller number of dimensions.

Recently, an emerging area of research within topic identification involves systems that can be trained without any supervision such that the sub-word units used are derived from the data instead of from previously transcribed data [17], [20]. The advantage of systems using such an approach that does not require transcriptions for training includes the extension to new languages. Extending speech recognition and topic identification to languages for which only small amounts of untranscribed data are available is an active topic. An additional advantage of these techniques is the flexibility in adapting to new acoustical and environmental conditions.

9.4 Language Recognition

Language identification is the process of identifying the language spoken in a speech utterance. Over the years, a number of applications have been proposed for language identification. The first set of proposed applications includes those in which the language identification process is used as a preprocessing step for future automatic processing. Examples of this type of application include the initial stage of a speech recognition system where the hypothesized language is used to determine which models are used to decode the signal, and similarly, machine translation, where language identification is used to determine the whole set of models that are used for translating the source speech into a known target set.

Algorithms have evolved rapidly over the last decade. For most of the previous decade, the dominant technology was based on phone n -grams. The phone n -gram system, also known as phonotactics, is the process of studying the sequence of phones that are obtained when the speech signal is decoded. The basic principle is that these sequences will be different across languages and facilitate discrimination. Typically, these systems are constructed in three stages.

The first stage is the phone-decoding stage. In this stage, the incoming speech signal is decoded into a sequence of phones. This decoding step is similar to the first stage in the speech recognition system with minor differences. In this case, as in the speech recognition case, $p(X | A)$ is also an HMM, but it is decoded in open-loop fashion instead of constrained by pronunciation or the language model. The result is that the incoming speech is tokenized into a sequence of phones without any linguistic knowledge used to reduce the search. Concretely, the observation $X = x_{1:n}$ is mapped to a sequence of phones $P = p_{1:m}$ using the set of HMMs that had been trained on phonemes. This process can be language specific or universal. It is important to mention that there does not need to be a relation between the language or languages over which the phonetic inventory is constructed and the list of target languages that we are interested in identifying [16].

The second stage is the language model stage. Similar to the idea presented earlier for speech recognition, the language models trained in this stage are based on language-specific data. The number of language models trained is usually related to the set of target languages. This stage is key to discriminating across languages. The discrimination power exploited by the language model is based on the fact that the frequency of sequences of the tokens observed varies greatly across languages and in some cases is almost unique to a given language. Across the years, researchers have looked at sequences ranging from bigrams to 5-grams, with trigrams typically employed by most systems. An example of a language modeling technique that is widely used by various researchers including Zissman [21] is the interpolated language model. An example of an interpolated bigram

($n = 2$) language model is shown below. In this case, the sequence is modeled using

$$P(p_t | p_{t-1}) = \alpha_2 P(p_t | p_{t-1}) + \alpha_1 P(p_t) + \alpha_0 P_0. \quad (9.5)$$

Here, $P(p_t | p_{t-1})$ is the conditional probability of observing phone p_t given that phone p_{t-1} has been observed and $\alpha_{0,2}$ are weights related to how confident we are in the given n -gram.

The final stage in phonotactic systems is a post-processor component, sometimes referred to as the backend. The backend can be a simple score normalization process or can be trained, like a linear classifier, on a held-out set and used to model the distributions of the language model scores across the different classes. The decision in this case is just the maximum language model score across all the classes of interest. Additional performance gains are usually obtained by combining multiple language-specific phonotactic components.

Recent work on acoustic or spectral systems has exploited information at a lower level in the speech hierarchy. Systems in this category include GMMs, support vector machines (SVM), or a combination of both. In the case of GMM systems, over the last decade, performance has greatly improved by three main factors:

- shifted-delta cepstral features, which account for temporal information [22];
- higher order models, with systems including up to 2048 mixture components; and
- the introduction of discriminative training.

Discriminatively trained systems have been dominant in this area over the last few years. More recently, systems based on combining GMMs with SVMs have resulted in additional gains over GMM-only systems. As an example, a language identification system based on GMMs usually assumes knowledge of a large set of observations labeled by language. In this case, we built a GMM for each language of interest and compute the likelihood for each class following

$$\hat{\ell} = \arg \max_{\ell} \log p(X | \theta_{\ell}), \quad (9.6)$$

where

$$\log p(X | \theta_{\ell}) = \frac{1}{T} \sum_{t=1}^T \log p(x_t | \theta_{\ell}), \quad (9.7)$$

where ℓ is the language for which the set of parameters θ_{ℓ} produces the highest likelihood for observation X .

State-of-the-art performance in this area is currently obtained by employing sub-space compensation methods, particularly the technique known as i-vectors [23]. The i-vector algorithm is a result of extending factor analysis methods to language identification. In this approach, a language-independent GMM is trained and used to create a supervector (concatenation of mean vectors) for each incoming training speech utterance. Each supervector is then projected into a lower dimensional space using a matrix T obtained from all the available training data. The matrix T is related to the eigenvector matrix obtained by principal component analysis of the covariance matrix of the training data. Typically, the dimensionality of the i-vector subspace is 200 to 600 dimensions. Once the data have been projected into the i-vector subspace, either simple distance metrics or conventional classifiers (e.g., SVN or GMM) are used for classification.

Although phonotactic and acoustic (spectral-based) algorithms have been the dominant technologies in language identification, other techniques have been used with more limited success. The most notable ideas in this case include prosodics and word n -grams. Prosodic-based systems attempt to exploit the differences in rhythm observed across different language classes. These systems typically look at features based on the fundamental frequency (pitch) and its derivatives along with syllabic rate. Word n -gram systems have shown excellent performance but are usually limited to target languages on which well-trained systems are available. The limited availability of full-scale, well-trained systems on a large set of languages results in limiting the set of possible applications for this technique [24]–[26].

In the field of language identification, the National Institute of Standards and Technology (NIST) has conducted language-recognition evaluations since the mid-1990s [27]. The motivation is to assess the state of the technology by providing a uniform set of data over which different systems can be compared. The evaluation paradigm is based on creating a previously unseen dataset and having participating sites deliver classification decisions without knowledge of the truth labels of the data. Figure 9.1 shows results for Lincoln Laboratory systems over the years. These systems are usually the result of combining various systems, such as a phonotactic system being combined with various cepstral-based systems [28], [29].

9.5 Speaker Identification

Speaker identification is the process of identifying speakers by their voice. There are two main applications of interest. The first is speaker verification, in which there is a speech sample and a claimed identity. In this case, the speech sample is processed and scored against the model for the claimed speaker, and verification is granted if the produced score is above a certain threshold. The second application of interest is the identification case, which lends its name to the general area. In the identification case, there are two typical scenarios:

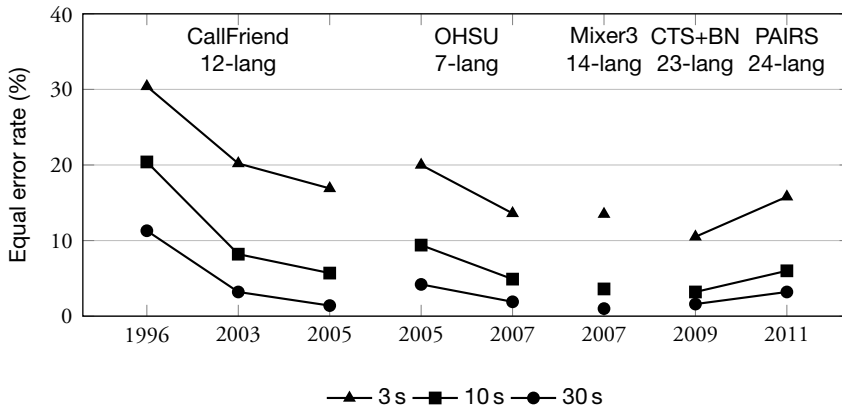


Figure 9.1 Performance trend of language-recognition systems on National Institute of Standards and Technology corpora.

1. A *closed-set* scenario is one in which a speaker is known to be someone within a predetermined set of models. The chosen speaker is usually the one that scores the highest within the set of models.
2. An *open-set* or *out-of-set* scenario is one in which the speaker is not guaranteed to be within the predetermined set of models. In this scenario, the speaker is chosen to be within the model set only if the score is above a certain threshold.

Over the past 20 years, the area of speaker identification has been dominated by algorithms based on GMMs that probabilistically model the salient characteristics of the speakers. The foundational work in this area [30], [31] relies on using a universal background model to model the general speaker population and serve as a competing hypothesis for the hypothesis test. Additionally, the universal background model allows for a computational advantage when multiple speakers are considered [31].

Recent advances have resulted in improved performance over the conventional GMM approach. Initially, gains in performance were obtained by combining systems based on GMMs with other systems that provided complementary information. The first of these systems was based on employing classifiers, such as SVMs, to model the speaker characteristics [32]. Later on, the GMM- and SVM-based systems were combined with another class of systems that not only relied on a different set of classifiers but focused on a different set of features altogether [33]. The new high-level features relied on information extracted beyond conventional cepstral features into longer-term units such as phones, words, and word usage [33], [34]. The general idea is that if enough speaker data are available to train the models, then speaker-specific behavior could be observed at the phone and word-usage level. Additionally, another type of system was

developed by exploiting the power of SVMs in combination with features derived from speech-recognition systems.

More recently, a new class of systems has been proposed that addresses one of the main causes affecting the performance of speaker identification, namely, channel variability. It is well known [31] that the performance of speaker identification is adversely affected by having the enrollment data, used to train a speaker model, come from a different source from the testing data used to evaluate the system performance. In particular, systems based on factor analysis and nuisance attribute projection specifically exploit the availability of multiple enrollment utterances for each of a large set of speakers to extract the channel information that needs to be compensated for. In the case of nuisance attribute projection, the basic idea is to model a speaker utterance as the additive combination of a generic speaker component based on the alignment of the speaker utterance to a universal background model and an offset related to the channel information. This channel information is constrained to a small subspace and then eliminated from the original speech. The initial proposal for systems using factor analysis has been extended further to a technique known as joint factor analysis (JFA), which attempts to mitigate not only the channel variation but also the variability across the different speakers.

The most recent extension of the JFA framework has resulted in a new approach named the i-vectors [35]. The concept of i-vectors relies on the same basic principles behind the JFA approach but is a different philosophical view of variability. In the i-vector approach, the underlying assumption is that all the variation related to the speaker, instead of the channel differences, can be captured in a small subspace. This type of system right now provides state-of-the-art performance in the field of speaker recognition [36].

9.5.1 Forensic Speaker Recognition

Forensic speaker recognition is an application for speaker recognition that has regained interest over the last decade. The classical scenario is that of determining whether an unknown voice sample is from the same speaker as a voice sample for which the speaker identity is known. In recent years, a number of prototypes have been developed addressing the needs of the forensic community. One prototype is Vocalinc, which builds on experience from NIST speaker-recognition evaluations. Figure 9.2 is a screen shot of the current Vocalinc graphical user interface. In the case of forensic analysis, a tool like Vocalinc is expected to be used by the forensic practitioner in an interactive fashion following what is known as the “human-in-the-loop” type of operation.

The user interface allows for the selection of the speaker-recognition algorithms to be used in the analysis. The algorithms included in the tool are the GMM, SVM, JFA, IPDF, and i-vector. The inner product discriminant function (IPDF) is a generalization

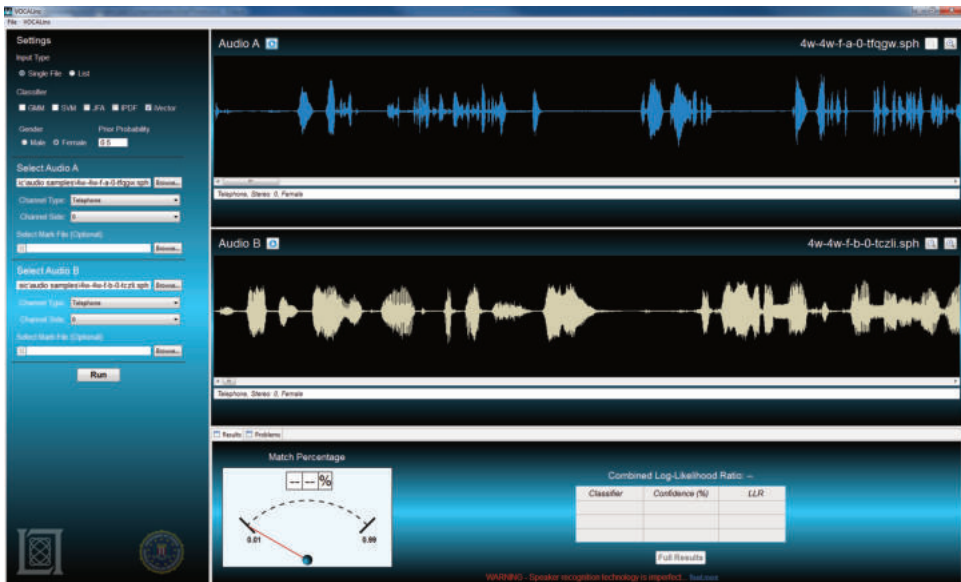


Figure 9.2 Screen shot of Vocalinc user interface.

of a hybrid system including a GMM and an SVM [37]. All of these algorithms can be used individually or combined. The user can decide to operate the tool in 1-to-1 mode, the process outlined earlier in which there is a known and a questioned speech sample, or in list mode, in which the user runs two lists of audio files against each other. List mode allows for an n -by- m set of comparisons.

The tool allows for audio files to be used and for the user to indicate metadata related to gender, channel type, channel side (in the case of stereo files), and section marks. The algorithms use these metadata to select which models should be used during analysis. In the case of section marks, the marks indicate the sections of the speech signal to analyze. If marks are used, then it is usually the case that these sections are obtained from the analysis by listening to the signal and marking the relevant areas where the speaker of interest could be present.

The interface displays the audio signals to the user and the result of the comparison. It shows the probability of match for the two speech utterances. There is also a table that shows the individual results for each of the classifiers that were selected by the user. In the case of list or matrix operation, the results for the list operation are available through the results tab in the results section.

The prototype will be extended in a number of areas. As new algorithms are developed, they will be incorporated into the system. Eliminating the need for user metadata input and allowing the system to automatically determine information like gender and channel

can result in improved performance. In the future, the system will incorporate intrinsic information about the speaker, such as emotion, health status, and stress. Understanding how intrinsic information about the speaker can affect the performance of speaker-recognition systems is currently not well understood, and it is expected to be a focus of research in decades to come.

9.6 Machine Translation

Machine translation, also known as automatic machine translation, is the process of converting audio or text from one language to audio or text for another language. Over the years, two main types of systems have been proposed to address the translation problem. First, rule-based systems have been proposed that focus on applying word-based translation followed by using linguistic rules to reorder the words or tokens [38]. The second type of system is based on statistical techniques, in which training data are available for both languages of interest, and the observations are mapped from one language to the other using basic statistics about occurrences of words and phrases.

Rule-based systems usually rely on three basic components. A dictionary component maps words from the input language to the output language. The second and third components of these systems are typically a set of linguistic rules for the source and target languages. These linguistic rules usually include rules about the sentence structure and grammar in each language [39].

Statistical machine translation has been the dominant approach for machine translation over the last two decades. Although statistical systems were proposed in the 1950s, work conducted at IBM [40] in the late 1980s started the modern shift toward statistical techniques. At the heart of the modern statistical approach is the concept of parallel corpora, that is, corpora in both the source and target language where the target language data represent the human translation of the available data in the source corpus. Examples of models proposed include word- and phrase-based models. In the case of word-based models, the typical steps include a conversion from each word (or token) in the source language to a single word (or token) in the target language. A second step in this type of system rearranges the words in the target language such that the probability of the word sequence is maximized.

In the case of phrase-based systems, the approach is similar to that of word-based systems, but it uses phrases as the unit of interest. Here, the sentences in the source language are usually parsed into phrases, defined simply as sequences of tokens, and these phrases are then converted to phrases in the target language. The set of phrases obtained in the target language is then reordered using a similar approach to that used for word-based systems. An excellent survey on statistical machine translation approaches has been written by Lopez [41].

Assessing the performance of translation systems is not easy and provides a difficult challenge to researchers in this area. Human evaluation of the generated output tends to provide an inconsistent metric of how good a translation is. Automatic measurements are needed in the machine translation arena to provide a consistent, automatic, and unbiased estimate of system performance. A well-known and widely accepted metric is based on the method known as the Bilingual Language Evaluation Understudy (BLEU) [42]. The BLEU measure is based on the concept of precision. This measure uses word and word-sequence matches between the hypothesized translation and high-quality reference human translations.

Over the last decade, research has focused on human-factor aspects of machine translation [43]. Although machine translation performance is errorful, there may be opportunities to make use of the current state of systems. In the past, evaluation metrics have focused on simply reducing translation errors. However, it is important to measure the effectiveness of a system. An imperfect or inaccurate system can still be useful to users for certain applications. Recent work has involved developing new metrics and measures of effectiveness that can help us understand how current systems can be used.

9.7 Summary

In this chapter, we have discussed speech processing systems focusing on active areas of research. In particular, we discussed applications in the areas of automatic speech recognition, language recognition, speaker recognition, and machine translation. We have described state-of-the-art algorithms in each of these areas and discussed applications of interest where appropriate. Speech processing systems are still an active area of research, and we believe that additional improvements in many areas can be made and new applications developed in the years to come.

References

1. L.R. Rabiner and R.W. Schafer, *Theory and Applications of Digital Speech Processing*. Upper Saddle River, NJ: Prentice Hall, 2011.
2. J.R. Deller Jr., J.H.L. Hansen, and J.G. Proakis, *Discrete-Time Processing of Speech Signals*. New York: IEEE Press, 2000.
3. D.G. Childers, D.P. Skinner, and R.C. Kemerait, “The Cepstrum: A Guide to Processing,” *Proceedings of the IEEE*, vol. 65, no. 10, pp. 1428–1443, 1977. DOI: 10.1109/PROC.1977.10747.
4. S.B. Davis and P. Mermelstein, “Comparison of Parametric Representations for Monosyllabic Word Recognition in Continuously Spoken Sentences,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 28, no. 4, pp. 357–366, 1980. DOI: 10.1109/TASSP.1980.1163420.

5. L. Rabiner, “A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition,” *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989. DOI: 10.1109/5.18626.
6. A.B. Poritz, “Hidden Markov Models: A Guided Tour,” in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 1988.
7. L.E. Baum and J.A. Eagon, “An Inequality with Applications to Statistical Estimation for Probabilistic Functions of Markov Processes and to a Model for Ecology,” *Bulletin of the American Mathematical Society*, vol. 73, no. 3, pp. 360–363, 1967. DOI: 10.1090/S0002-9904-1967-11751-8.
8. L.E. Baum, “An Inequality and Associated Maximization Technique in Statistical Estimation for Probabilistic Functions of Markov Processes,” in *Inequalities III*, O. Shisha, ed., New York: Academic Press, 1972.
9. A.P. Dempster, N.M. Laird, and D.B. Rubin, “Maximum Likelihood from Incomplete Data via the EM Algorithm,” *Journal of the Royal Statistical Society, Series B (Methodological)*, vol. 39, no. 1, pp. 1–38, 1977.
10. L.R. Bahl, P.F. Brown, P.V. de Souza, and R.L. Mercer, “Maximum Mutual Information Estimation of Hidden Markov Model Parameters for Speech Recognition,” in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 1986.
11. J. Bilmes, “A Gentle Tutorial of the EM Algorithm and Its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models,” International Computer Science Institute, Tech. Rep. TR-97-021, 1997.
12. A.J. Viterbi, “Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm,” *IEEE Transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, 1967. DOI: 10.1109/TIT.1967.1054010.
13. L.R. Rabiner and R.W. Schafer, “Introduction to Digital Speech Processing,” *Foundations and Trends in Signal Processing*, vol. 1, no. 1-2, pp. 1–194, 2007. DOI: 10.1561/20000000001.
14. L. Rabiner and B.-H. Juang, *Fundamentals of Speech Recognition*. Upper Saddle River, NJ: Prentice Hall, 1993.
15. G.E. Dahl, D. Yu, L. Deng, and A. Acero, “Context-Dependent Pre-Trained Deep Neural Networks for Large-Vocabulary Speech Recognition,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 20, no. 1, pp. 30–42, 2012. DOI: 10.1109/TASL.2011.2134090.
16. H. Gish, M.-H. Siu, A. Chan, and W. Belfield, “Unsupervised Training of an HMM-Based Speech Recognizer for Topic Classification,” in *Annual Conference of the International Speech Communication Association (INTERSPEECH)*, 2009.

17. T.J. Hazen, "Topic Identification," in *Spoken Language Understanding: Systems for Extracting Semantic Information from Speech*, G. Tur and R. De Mori, eds., Hoboken, NJ: Wiley, 2011.
18. K.S. Jones, "A Statistical Interpretation of Term Specificity and Its Application in Retrieval," *Journal of Documentation*, vol. 28, no. 1, pp. 11–21, 1972. doi: 10.1108/eb026526.
19. S. Deerwester, S.T. Dumais, G.W. Furnas, T.K. Landauer, and R. Harshman, "Indexing by Latent Semantic Analysis," *Journal of the American Society for Information Science*, vol. 41, no. 6, pp. 391–407, 1990. doi: 10.1002/(SICI)1097-4571(199009)41:6<391::AID-ASI1>3.0.CO;2-9.
20. T. Hazen, M.-H. Siu, H. Gish, S. Lowe, and A. Chan, "Topic Modeling for Spoken Documents Using Only Phonetic Information," in *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, 2011.
21. M. Zissman, "Comparison of Four Approaches to Automatic Language Identification of Telephone Speech," *IEEE Transactions on Speech and Audio Processing*, vol. 4, no. 1, p. 31, 1996. doi: 10.1109/TSA.1996.481450.
22. P.A. Torres-Carrasquillo, E. Singer, M.A. Kohler, R.J. Greene, D.A. Reynolds, and J.R. Deller Jr., "Approaches to Language Identification Using Gaussian Mixture Models and Shifted Delta Cepstral Features," in *Annual Conference of the International Speech Communication Association (INTERSPEECH)*, 2002.
23. N. Dehak, P.A. Torres-Carrasquillo, D.A. Reynolds, and R. Dehak, "Language Recognition via I-Vectors and Dimensionality Reduction," in *Annual Conference of the International Speech Communication Association (INTERSPEECH)*, 2011.
24. S. Kadambe and J. Hieronymus, "Language Identification with Phonological and Lexical Models," in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 5, 1995.
25. J. Hieronymus and S. Kadambe, "Spoken Language Identification Using Large Vocabulary Speech Recognition," in *International Conference on Spoken Language Processing (ICSLP)*, vol. 3, 1996.
26. W. Campbell, F. Richardson, and D. Reynolds, "Language Recognition with Word Lattices and Support Vector Machines," in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 4, 2007.
27. A. Martin and C. Greenberg, "The 2009 NIST Language Recognition Evaluation," in *Odyssey: The Speaker and Language Recognition Workshop*, 2010.
28. P. Torres-Carrasquillo, E. Singer, T. Gleason, A. McCree, D. Reynolds, F. Richardson, and D. Sturim, "The MITLL NIST LRE 2009 Language Recognition System," in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2010.

29. E. Singer, P.A. Torres-Carrasquillo, D.A. Reynolds, A. McCree, F. Richardson, N. Dehak, and D. Sturim, “The MITLL NIST LRE 2011 Language Recognition System,” in *Odyssey: The Speaker and Language Recognition Workshop*, 2012.
30. D.A. Reynolds, “Speaker Identification and Verification Using Gaussian Mixture Speaker Models,” *Speech Communication*, vol. 17, no. 1-2, pp. 91–108, 1995. DOI: 10.1016/0167-6393(95)00009-D.
31. D.A. Reynolds, T.F. Quatieri, and R.B. Dunn, “Speaker Verification Using Adapted Gaussian Mixture Models,” *Digital Signal Processing*, vol. 10, no. 1-3, pp. 19–41, 2000. DOI: 10.1006/dspr.1999.0361.
32. V. Wan and W. Campbell, “Support Vector Machines for Speaker Verification and Identification,” in *IEEE Signal Processing Society Workshop on Neural Networks for Signal Processing*, vol. 2, 2000.
33. D. Reynolds, W. Andrews, J. Campbell, J. Navratil, B. Peskin, A. Adami, Q. Jin, D. Klusacek, J. Abramson, R. Mihaescu, J. Godfrey, D. Jones, and B. Xiang, “The SuperSID Project: Exploiting High-Level Information for High-Accuracy Speaker Recognition,” in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 4, 2003.
34. W.D. Andrews, M.A. Kohler, J.P. Campbell, J.J. Godfrey, and J. Hernandez-Cordero, “Gender-Dependent Phonetic Refraction for Speaker Recognition,” in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 1, 2002.
35. N. Dehak, P. Kenny, R. Dehak, P. Dumouchel, and P. Ouellet, “Front-End Factor Analysis for Speaker Verification,” *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 19, no. 4, pp. 788–798, 2011. DOI: 10.1109/TASL.2010.2064307.
36. L. Burget, O. Plchot, S. Cumani, O. Glembek, P. Matejka, and N. Briimmer, “Discriminatively Trained Probabilistic Linear Discriminant Analysis for Speaker Verification,” in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2011.
37. W.M. Campbell, Z.N. Karam, and D.E. Sturim, “Speaker Comparison with Inner Product Discriminant Functions,” in *Advances in Neural Information Processing Systems (NIPS)*, 2009.
38. W.J. Hutchins, “Machine Translation: A Brief History,” in *Concise History of the Language Sciences: From the Sumerians to the Cognitivists*, E.F.K. Koerner and R.E. Asher, eds., Oxford: Elsevier, 1995.
39. B.J. Dorr, P.W. Jordan, and J.W. Benoit, “A Survey of Current Paradigms in Machine Translation,” *Advances in Computers*, vol. 49, pp. 1–68, 1999. DOI: 10.1016/S0065-2458(08)60282-X.

40. P.F. Brown, J. Cocke, S.A.D. Pietra, V.J.D. Pietra, F. Jelinek, J.D. Lafferty, R.L. Mercer, and P.S. Roossin, “A Statistical Approach to Machine Translation,” *Computational Linguistics*, vol. 16, no. 2, pp. 79–85, 1990.
41. A. Lopez, “Statistical Machine Translation,” *ACM Computing Surveys*, vol. 40, no. 3, pp. 1–49, 2008. doi: 10.1145/1380584.1380586.
42. K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, “BLEU: A Method for Automatic Evaluation of Machine Translation,” in *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2002.
43. D. Jones, W. Shen, and M. Herzog, “Machine Translation for Government Applications,” *Lincoln Laboratory Journal*, vol. 18, no. 1, pp. 41–53, 2009.

Optimized Airborne Collision Avoidance

Mykel J. Kochenderfer

Developing robust aircraft collision avoidance logic that reliably prevents midair collisions without excessive alerting is challenging because of sensor error and uncertainty in the future paths of the aircraft. Over the past few years, research has focused on the use of a computational method known as dynamic programming for producing an optimized decision logic for airborne collision avoidance. This chapter describes recent research in modeling the collision avoidance problem as a partially observable Markov decision process and solving for the optimal strategy using dynamic programming. The performance of the system is evaluated using a statistical airspace model represented as a dynamic Bayesian network learned from recorded radar data.

10.1 Airborne Collision Avoidance Systems

Because the sky is so big and aircraft are so small, there were few midair collisions during the early years of aviation. By the 1950s, air travel had become commonplace, and the skies became more crowded. In 1956, a midair collision over the Grand Canyon resulted in 128 fatalities. At the time, this collision was the worst commercial air disaster in history. The collision caused a press frenzy and congressional hearings, ultimately leading to the establishment of the Federal Aviation Administration (FAA) in 1958.

The establishment of the FAA led to major improvements in both airspace design and air traffic control. The airspace was designed to keep aircraft separated. For example, depending on whether aircraft were flying west or east, they were expected to fly at different altitudes. Air traffic controllers relied on ground-based radars, keeping aircraft safely separated by calling out traffic to pilots and by vectoring aircraft.

The enhancements to airspace design and air traffic control significantly improved the safety of the airspace. However, midair collisions still occurred. A midair collision involving a commercial airliner over San Diego, California, in 1978 resulted in 144 fatalities, and another commercial airliner collision over Cerritos, California, in 1986 resulted in 82 fatalities. These two collisions, in particular, convinced Congress that an

additional layer of collision protection was needed in the form of an onboard system. This system would provide an independent safety net to protect against human error, both by air traffic controllers and pilots.

10.1.1 Traffic Alert and Collision Avoidance System

Development of an onboard capability started shortly after the midair collision over the Grand Canyon. Early concepts focused on primary radar surveillance that sends out energy pulses and measures the timing of the echo to infer distance. This approach did not work well for a variety of reasons, including the inability to accurately estimate the altitude of the intruder. The focus shifted to beacon-based systems that made use of the transponders already on board most aircraft. An aircraft would send out an interrogation over the radio data link and measure the amount of time required for the aircraft to reply. Information about altitude and intended maneuvers could also be shared across this radio data link. The initial system, called Beacon Collision Avoidance System (BCAS), was designed to operate in low-density airspace. The collision over San Diego spurred the development of the Traffic Alert and Collision Avoidance System (TCAS), which was based on BCAS but with enhancements that enabled its use in high-density airspace. The development spanned several decades. The collision over Cerritos led to Congress's mandating the use of TCAS in the United States, and now TCAS is required on all large passenger and cargo aircraft worldwide.

TCAS conducts airborne surveillance, computes advisories using a safety logic, and relays those advisories to the pilot through audio and a display. If another airborne aircraft is a potential threat, then TCAS issues a traffic advisory (TA), which gives the pilots an audio announcement "Traffic, Traffic" and highlights the intruder on a traffic display. The TA is intended to help pilots achieve visual acquisition of other aircraft and prepare for a potential avoidance maneuver. If a maneuver becomes necessary, then the system will issue a resolution advisory (RA), instructing the pilots to climb or descend to maintain a safe distance. There is an audio announcement of the required vertical maneuver, and the range of acceptable vertical rates is shown on the vertical speed indicator. On some aircraft, additional pitch guidance is provided to pilots.

TCAS may issue a variety of different vertical advisories, including:

- do not climb or descend,
- limit climb or descend to 500, 1000, or 2000 ft/min,
- level off,
- climb or descend at 1500 ft/min,
- increase climb or descend to 2500 ft/min, or
- maintain current vertical rate.

Depending on how the encounter evolves, TCAS may strengthen, weaken, or reverse the direction of the advisory. An RA provides vertical guidance only; TCAS does not

issue horizontal maneuvers such as heading changes or turns. After the encounter has been resolved, TCAS declares “Clear of Conflict.”

The logic for specifying when to alert and what advisory to issue is represented as a large collection of rules. The TCAS logic uses straight-line extrapolation to estimate the time to closest approach and the projected miss distance. If both are small, then the logic determines that an alert is necessary. If an alert is necessary, then the logic will model standard climb and descend maneuvers assuming a 5-second pilot response delay, followed by a 0.25 g acceleration. It chooses the direction that provides the greatest separation from the intruder. It then models a set of different advisory rates that are consistent with the chosen direction. TCAS chooses the lowest rate that provides a required amount of separation.

Although the general steps TCAS uses to select advisories are relatively straightforward, the details of the logic are complex. Embedded in the TCAS logic specification are many heuristic rules and parameter settings designed to compensate for sensor noise and error as well as for variability in the pilot response. There are also rules that govern when to strengthen, weaken, and reverse advisories and how to handle encounters with multiple simultaneous intruder aircraft.

10.1.2 Limitations of Existing System

TCAS has been successful in preventing midair collisions over the years, but the way in which the logic was designed limits its robustness. Fundamental to TCAS design is the use of a deterministic model. However, recorded radar data show that pilots do not always behave as assumed by the logic. Not anticipating the spectrum of responses limits TCAS’s robustness, as demonstrated by the collision of two aircraft in 2002 over Überlingen, Germany. TCAS instructed one aircraft to climb and one to descend. However, the pilot who received the TCAS climb instruction instead descended in accordance with the air traffic controller’s instructions, leading to a collision with the second aircraft, which was descending according to its TCAS instruction. If TCAS recognized the noncompliance of one of the aircraft and reversed the advisory of the compliant aircraft from a descend to a climb, the collision could have been prevented. A modification was later developed to address this specific situation, but improving the overall robustness of the logic requires a fundamental design change.

Just as the airspace has evolved since the 1950s, it will continue to evolve over the next decade. Significant change will occur with the introduction of the next-generation air traffic management system, which will be based on satellite navigation. This improved surveillance will allow aircraft to fly closer together to support traffic growth. In addition, there are user classes, such as general aviation and unmanned aircraft, that would benefit from a collision avoidance capability. Unfortunately, the current version of TCAS cannot

support the safety and operational requirements of this new airspace. With aircraft flying closer together, TCAS will alert pilots too frequently to be useful.

To meet these requirements, a major overhaul of the TCAS logic and surveillance system is needed. TCAS is currently limited to large aircraft capable of supporting its hardware and power requirements. The aircraft must also have sufficient performance to achieve the required vertical rates of climb or descent that the advisories currently demand. Although a collision avoidance system for small aircraft might help improve safety within general aviation, TCAS cannot be adapted for small aircraft without a costly redesign.

10.1.3 Unmanned Aircraft Sense and Avoid

Unmanned aircraft have great potential for scientific, law enforcement, and commercial applications, but they currently cannot fly in civil airspace without special authorization. Flexible airspace access requires the capability to sense and avoid other aircraft effectively. An automated airborne collision avoidance system that meets the strict safety requirements of civil aviation authorities would greatly expand the utility of unmanned aircraft.

TCAS cannot be used directly for sense and avoid for several reasons. The legacy system assumes that TCAS-equipped aircraft can achieve rates of at least 1500 ft/min to comply with initial climb and descend advisories and 2500 ft/min for strengthened advisories. Many unmanned aircraft platforms can only achieve around 500 ft/min to 1000 ft/min. These vertical constraints may make it beneficial to maneuver horizontally rather than vertically. TCAS, however, was not designed to provide heading change or airspeed guidance.

The TCAS logic was designed assuming a beacon-based surveillance system, but unmanned aircraft cannot rely solely on such a surveillance system. Because there is not a pilot on board the aircraft who can physically see and avoid other aircraft, unmanned aircraft surveillance systems must avoid all categories of aircraft—including those not equipped with beacon transponders. Sensor systems that can detect aircraft without transponders include radar, electro-optical, and infrared. These different sensors have different error characteristics and size, weight, and power constraints.

Given that it took many years to develop and certify TCAS, it is anticipated that it will be especially challenging to develop and certify different collision avoidance systems for all the various combinations of sensor systems and aircraft platforms. Efforts in the unmanned aircraft industry have largely focused on proprietary solutions for specific platforms and sensors, but a common system that could accommodate different sensor configurations and flight characteristics would significantly reduce the cost of development and certification.

10.1.4 Airborne Collision Avoidance System X

Research over the past several years has focused on formulating the problem of collision avoidance as a partially observable Markov decision process and using dynamic programming to optimize the collision avoidance system. Simulation studies with recorded radar data have confirmed that such an approach leads to a significant improvement to safety and operational performance. The FAA has formed a team of organizations to mature the technology into a new collision avoidance system called Airborne Collision Avoidance System X (ACAS X). The system is intended to become the next international standard for collision avoidance for both manned and unmanned aircraft.

ACAS X will bring major enhancements to both surveillance and the advisory logic. The system will move from the beacon-only surveillance of TCAS to a plug-and-play surveillance architecture that supports surveillance based on global positioning system (GPS) data and accommodates new sensor modalities. The new surveillance capabilities will also enable collision avoidance protection for new user classes, including small, general-aviation aircraft not currently equipped with TCAS. ACAS X represents a major revolution in how the advisory logic is generated and represented. Instead of using ad hoc rule-based specification, ACAS X represents much of the logic by using a numeric table that has been optimized with respect to models of the airspace. This new approach improves robustness, supports new requirements, and reduces unnecessary alerts. The process adopted by ACAS X greatly simplifies development and is anticipated to significantly lower the implementation and maintenance costs.

10.2 Collision Avoidance Problem Formulation

The collision avoidance problem can be formulated as a partially observable Markov decision process (POMDP) in a variety of different ways [1]–[4]. This section describes a formulation of the collision avoidance problem used in an early prototype of ACAS X designated for large transport and cargo aircraft.

10.2.1 Resolution Advisories

TCAS issues advisories to the pilot through an aural annunciation, such as “Climb, Climb,” and through a visual display. The visual display varies, but it is typically implemented on a vertical speed indicator, a vertical speed tape, or a pitch cue on the primary flight display. The set of advisories issued by TCAS can be interpreted as target vertical rate ranges. If the current vertical rate is outside the target vertical rate range, then the pilot should maneuver to come within the required range. If the current vertical rate is within the target range, then a corrective maneuver is not required, but the pilot should be careful not to maneuver outside the specified range.

Table 10.1 Advisory Set.

Name	Vertical Rate Range		Aural Annunciation
	Min (ft/min)	Max (ft/min)	
COC	$-\infty$	∞	Clear of Conflict (or nothing)
DNC2000	$-\infty$	2000	Monitor Vertical Speed
DND2000	-2000	∞	Monitor Vertical Speed
DNC1000	$-\infty$	1000	Monitor Vertical Speed
DND1000	-1000	∞	Monitor Vertical Speed
DNC500	$-\infty$	500	Monitor Vertical Speed
DND500	-500	∞	Monitor Vertical Speed
DNC	$-\infty$	0	Level-off, Level-off (or Monitor Vertical Speed)
DND	0	∞	Level-off, Level-off (or Monitor Vertical Speed)
MDES	$-\infty$	current	Maintain Vertical Speed, Maintain
MCL	current	∞	Maintain Vertical Speed, Maintain
DES1500	$-\infty$	-1500	Descend, Descend
CL1500	1500	∞	Climb, Climb
SDES1500	$-\infty$	-1500	Descend, Descend NOW, Descend, Descend NOW
SCL1500	1500	∞	Climb, Climb NOW, Climb, Climb NOW
SDES2500	$-\infty$	-2500	Increase Descent, Increase Descent
SCL2500	2500	∞	Increase Climb, Increase Climb

The advisories available to ACAS X are the same as those available to TCAS and are summarized in Table 10.1. In the POMDP problem formulation, the discrete actions correspond to the various advisories—except that MCL and MDES are merged into a single “maintain” action to reduce the size of the action space. Whether the maintain action is an MCL or an MDES is determined during execution on the basis of the current vertical rate.

If the current vertical rate is within the prescribed range of the advisory when it is issued, then the advisory is called *preventive*. Otherwise, the advisory is called *corrective*. In the table, DES1500, CL1500, SDES1500, SCL1500, SDES2500, and SCL2500 are always corrective. The DNC and DND can be either corrective or preventive. If DNC or DND is issued as a corrective, then it is annunciated as “Level-off, Level-off” (LOLO); otherwise, it is annunciated as “Monitor Vertical Speed” (MVS). All other MVS advisories are preventive.

The availability of these 16 discrete actions depends on the current advisory. For example, SDES2500 can only be issued after an MDES or a DES1500. The constraints on transitions between advisories are carried over from the original TCAS design so as to reduce the amount of new pilot training requirements.

The *sense* of an advisory is either up or down (or neither in the case of COC). An up advisory instructs the pilots to climb or not descend. A down sense advisory instructs the pilots to descend or not climb. Transitioning from one sense to another is called a *reversal*. Transitioning between two advisories of the same sense is either a *strengthening* or *weakening* depending on whether the new advisory instructs a faster vertical rate. Distinguishing these categories of transitions is important in modeling the pilot response.

10.2.2 Dynamic Model

The ACAS X prototype uses a relatively simple dynamic model of how aircraft behave. For several reasons, it is important to keep the model as simple as possible while capturing the important properties of aircraft behavior. Simpler models are easier for designers to validate and are less likely to be tailored too tightly to the idiosyncrasies of a particular airspace. In addition, because simpler models require fewer state variables, computing optimal advisories using dynamic programming is more tractable.

There are six state variables in the POMDP formulation:

- h , the altitude of the intruder relative to the own aircraft,
- $\dot{h}_0$, the vertical rate of the own aircraft,
- $\dot{h}_1$, the vertical rate of the intruder aircraft,
- τ , the time to potential collision,
- s_{adv} , the current advisory, and
- s_{res} , whether the pilot is responding to the advisory.

In past publications, s_{adv} and s_{res} were combined together in a single discrete variable called s_{RA} , but keeping these variables separate simplifies the explanation.

The discrete-time dynamic model assumes a time step of $\Delta t = 1$ s, which corresponds to the decision frequency of TCAS. The advisory response at the next time step s'_{res} is determined stochastically based on the current advisory s_{adv} , response s_{res} , and new advisory a according to

$$P(s'_{\text{res}} = \text{true} \mid s_{\text{adv}}, s_{\text{res}}, a) = \begin{cases} 1 & \text{if } a = \text{COC} \\ 1 & \text{if } s_{\text{res}} = \text{true and } s_{\text{adv}} = a \\ 1/(1+5) & \text{if } s_{\text{adv}} = \text{COC and } a \neq \text{COC} \\ 1/(1+5) & \text{if } s_{\text{adv}} \text{ and } a \text{ are opposite sense} \\ 1/(1+3) & \text{if } s_{\text{adv}} \text{ and } a \text{ are same sense} \end{cases} \quad (10.1)$$

Because the advisory response is determined according to a Bernoulli process, the delay until response follows a geometric distribution. If the response probability at each step in the process is $(1/(1+k))$, then the mean time until response is k . Hence,

- the pilot always responds to a COC,
- once the pilot responds, the pilot continues to respond for the duration of the advisory,
- the average response delay for initial advisories is 5 s,
- the average response delay for reversals is 5 s, and
- the average response delay for a strengthening or weakening is 3 s.

A more complex pilot response model has been explored, but it brought little benefit and added significantly to the size of the state space [5].

The own acceleration $\ddot{h}_0$ is determined stochastically from s'_{res} and a . If the pilot is not responding to an advisory, then $\ddot{h}_0$ is sampled from a zero-mean Gaussian with a standard deviation of 3 ft/s². If the pilot is responding, then $\ddot{h}_0$ is chosen to bring the vertical rate to within the range required by the advisory. The magnitude of the acceleration is drawn from a Gaussian distribution with a standard deviation of 1 ft/s². Generally, the mean of the Gaussian distribution is 8.33 ft/s², but if an advisory has been reversed or strengthened to a climb or descend, then a stronger acceleration of 10.7 ft/s² is used instead. These accelerations were chosen to match those used by TCAS.

The model assumes that the intruder simply applies random accelerations, drawn independently once per second. The intruder acceleration $\ddot{h}_1$ is simply drawn from a zero-mean Gaussian with 3 ft/s² standard deviation. More sophisticated models explored in the past have captured the influence of the intruder's own collision avoidance system, but analysis has shown relatively little benefit [6].

Given a , s'_{res} , $\ddot{h}_0$, and $\ddot{h}_1$, the state is updated as follows:

$$\begin{bmatrix} h \\ \dot{h}_0 \\ \ddot{h}_0 \\ \dot{h}_1 \\ \ddot{h}_1 \\ \tau \\ s_{\text{adv}} \\ s_{\text{res}} \end{bmatrix} \leftarrow \begin{bmatrix} h + \dot{h}_1(\Delta t) + \frac{1}{2}\ddot{h}_1(\Delta t)^2 - \dot{h}_0(\Delta t) - \frac{1}{2}\ddot{h}_0(\Delta t)^2 \\ \dot{h}_0 + \ddot{h}_0(\Delta t) \\ \ddot{h}_0 \\ \dot{h}_1 + \ddot{h}_1(\Delta t) \\ \ddot{h}_1 \\ \tau - 1 \\ a \\ s'_{\text{res}} \end{bmatrix}. \quad (10.2)$$

10.2.3 Reward Function

The reward function is intended to capture the safety and operational considerations. To easily apply dynamic programming, the reward function can only depend on the six state variables and the current action. Early in the development of the system, the reward function had costs assigned to near collision, issuing an initial advisory, strengthening, and reversing. Analysis of the performance of the system in simulation revealed that a variety of additional cost parameters were necessary to make the advisories more suitable and effective [7].

Table 10.2 Event rewards.

Reward	Separation (ft)	Closure (ft/min)	Event
-1	≤ 175		$\tau \leq 0$
-1			Maintain advisory with $\dot{h}_0 < 1500$ ft/min
-1			Prohibited advisory transitions
-1			Preventive crossing advisory
-0.1	> 650	< 2000	Corrective advisory
-3×10^{-2}	> 1000	< 4000	Corrective advisory
-1×10^{-2}	> 650	< 2000	Preventive advisory
-1×10^{-2}	> 500		Crossing advisory
-8×10^{-3}			Reversal
-5×10^{-3}			Strengthening
-1×10^{-3}			Weakening
-1.5×10^{-3}		> 3000	Non-MVS/LOLO
-2.3×10^{-3}		< 3000	Any advisory
-5×10^{-4}		> 3000	MVS/LOLO
$-4 \times 10^{-4} \times \Delta \dot{h}$			Crossing advisory when $ \dot{h}_0 > 500$ ft/min and $\dot{h}_0$ is in opposite direction of advisory
-4×10^{-4}			Maintain
-1×10^{-4}			MVS/LOLO
$-3 \times 10^{-5} \times \Delta \dot{h}$			Any advisory
-1×10^{-5}			Corrective advisory
1×10^{-9}			COC

Table 10.2 outlines the various event rewards. For a reward in the table to apply, the specified separation, closure, and event must all hold. Given a state and action, all events that apply in the table contribute to the immediate reward. All the rewards in the table are negative except for issuing COC. Some rewards depend on the vertical separation (i.e., $|h|$) and vertical closure rate (i.e., $|\dot{h}_1 - \dot{h}_0|$). Others depend on whether the advisory is a *crossing*, which is defined to occur when either (1) a down sense is issued when the intruder is below, or (2) an up sense is issued when the intruder is above. Crossing advisories can require the aircraft to cross each other in altitude and are avoided when possible by TCAS.

The table contains two rows that depend on a variable $\Delta \dot{h}$. This variable is defined to be the magnitude of the difference between the minimum required change in vertical rate to comply with the advisory. In other words, if the advisory has a required range of $[\dot{h}_{\min}, \dot{h}_{\max}]$, then $\Delta \dot{h} = \min(|\dot{h}_{\min} - \dot{h}_0|, |\dot{h}_{\max} - \dot{h}_0|)$.

As discussed later, an important part of the development process involves determining how to set these reward parameters. These parameters can be adjusted to make trades

between different safety and operational considerations. For example, the corrective advisory cost can be decreased to increase the rate of corrective advisories, leading to increased safety.

10.2.4 Dynamic Programming

Dynamic programming (Section 4.2) is used to compute the value function U^* assuming full observability. The dynamics, as described in Section 10.2.2, have the distributions over the accelerations $\ddot{h}_0$ and $\ddot{h}_1$ specified as continuous probability densities—and, therefore, the next state distribution T is a density. Hence, the Bellman equation involves integration instead of summation:

$$U^*(s) = \max_a \left(R(s, a) + \gamma \int T(s' | s, a) U^*(s') ds' \right). \quad (10.3)$$

Evaluating the integral analytically is not feasible, but any of the standard numerical integration methods can be used. The approach taken in the ACAS X prototype involves generating a set of weighted values for $\ddot{h}_0$ and $\ddot{h}_1$ using sigma-point sampling [8]. With the set of next states given s and a now finite, the Bellman equation becomes

$$U^*(s) = \max_a \left(R(s, a) + \gamma \sum_{s'} T(s' | s, a) U^*(s') \right). \quad (10.4)$$

Because the state space is continuous, the local approximation value iteration algorithm (Section 4.5.1) is used to compute U^* . In particular, ACAS X uses multilinear interpolation over a grid-based discretization of the state space. The h variable is discretized into 33 points over the range ± 4000 ft with finer discretization near 0 ft. The vertical rate variables $\dot{h}_0$ and $\dot{h}_1$ are discretized into 25 points each between $\pm 10,000$ ft/min with finer discretization near 0 ft/min. The time to potential collision variable τ is discretized at 1 s from 0 to 40 s.

The structure of the problem is such that only a single sweep of Gauss-Seidel value iteration (Section 4.2.6) is required. Because states with $\tau = k$ depend only on the states with $\tau = k - 1$, ordering the sweep of the states by increasing τ value results in the optimal value function. Although there are more than 26 million vertices, the process requires only a few minutes on a modern workstation. The state-action values $Q^*(s, a)$ produced through dynamic programming are saved in a lookup table. Recent research has explored methods for reducing the size of the table for use on airborne equipment with limited memory capacity [9].

10.3 State Estimation

The previous section explained how to compute the lookup tables under the assumption of full observability. During flight, uncertainty in the state variables needs to be accommodated in real time. The system estimates a belief distribution b and uses it to select the best action:

$$\pi^*(b) = \arg \max_a \int b(s) Q^*(s, a) ds, \quad (10.5)$$

where $Q^*(s, a)$ comes from using multilinear interpolation on the lookup table computed offline. This method for approximating the solution to a POMDP was introduced in Section 6.4.1 as the QMDP technique. To improve the efficiency of computing Equation (10.5), instead of representing the belief b as a probability density function, the system uses a set of samples $s^{(1)}, \dots, s^{(n)}$ with associated weights $w^{(1)}, \dots, w^{(n)}$. Hence, Equation (10.5) becomes

$$\pi^*(b) = \arg \max_a \sum_i w^{(i)} Q^*(s^{(i)}, a). \quad (10.6)$$

The belief state b can be factored as follows:

$$b(s) = b(h, \dot{h}_0, \dot{h}_1, \tau, s_{\text{adv}}, s_{\text{res}}) = b(h, \dot{h}_0, \dot{h}_1) b(\tau) b(s_{\text{adv}}, s_{\text{res}}), \quad (10.7)$$

where the three components are disambiguated in the text here by their arguments. This section explains how to estimate these three component distributions and how to incorporate additional operational considerations in real time through the introduction of online costs.

10.3.1 Sensor Error

Aircraft estimate their altitude using an altimeter that measures atmospheric pressure. These estimates include some error resulting from variability in pressure gradients and calibration. When an aircraft sends its altitude information to another aircraft across a radio data link, it is quantized to either 25 or 100 ft depending on the transponder.

TCAS uses a simple alpha-beta filter to estimate vertical rate when the altitude is quantized at 25 ft [10], but it uses a much more complex, nonlinear filter when the altitude is quantized at 100 ft [11]. The TCAS filters only provide single estimates of the altitude and vertical rates.

Early in the development of ACAS X, it was found that performance could be improved by explicitly taking into account the uncertainty of the estimates of h , $\dot{h}_0$, and $\dot{h}_1$ [12]. Further work resulted in a flexible filter based on the Kalman filter (Section 6.2.2) modified to better accommodate quantization error [13]. The filter outputs a set of weighted state samples. A similar filter was also developed to accommodate noise in angular position measurements [14].

10.3.2 Pilot Response

The distribution over whether the pilot is responding to the previously issued advisory is updated over time according to Bayes' rule [5]. The update is a function of the distribution over own vertical rate $\dot{h}_0$, the advisory issued, and the pilot response model specified in Equation (10.1).

10.3.3 Time to Potential Collision

The time to potential collision τ cannot be known exactly because it depends on the future trajectories of the aircraft. There are many different models of how the lateral positions of aircraft evolve over time [15], but one of the simplest models assumes white-noise acceleration vectors. Using this model, ACAS X estimates the distribution over the time until another aircraft comes within some fixed lateral distance.

There are different ways to estimate the time to potential collision. For example, one could use Monte Carlo sampling. Given the initial positions and velocities of the aircraft, the accelerations can be sampled and the paths simulated forward in time. With a sufficient number of sampled trajectories, a histogram over the time to collision can be built. One disadvantage of a sampling approach is that it can be too computationally demanding to be done in real time. In addition, certifying a safety-critical system that relies on random-number generation can be difficult.

The approach taken in ACAS X is to compute the distributions offline and store them in a lookup table. The distributions can be computed efficiently by using an iterative process. Let $D_k(s)$ represent the probability that the other aircraft comes within a fixed lateral distance in k seconds. Computing $D_0(s)$ is straightforward because it is simply 1 if s is a state in which the other aircraft is within the fixed lateral distance and 0 otherwise. The probability $D_k(s)$ can be computed from D_{k-1} as follows:

$$D_k(s) = \sum_{s'} T(s, s') D_{k-1}(s'), \quad (10.8)$$

where $T(s, s')$ is the probability of transitioning from one horizontal state s to another horizontal state s' . The equation above can be applied repeatedly up to some desired horizon. For the early prototype of ACAS X, this horizon is 40 s, chosen to provide

adequate alert time prior to potential collision. The probability that $\tau \geq 40$ given state s is simply $1 - \sum_{k=0}^{39} D_k(s)$.

By taking advantage of symmetry, it is possible to represent the horizontal state space using three variables (see [16]). To store the distribution in a table, the horizontal state space must be discretized. Because there are only three variables, the discretization can be made relatively fine. To determine the distribution for an arbitrary horizontal state not corresponding to a discretization vertex, ACAS X uses multilinear interpolation.

The horizontal state, of course, cannot be known exactly, but a distribution can be inferred from sensor measurements. Specialized state-estimation algorithms based on the unscented Kalman filter [8] have been developed for different surveillance systems. These algorithms output weighted horizontal state samples, and their associated distributions are combined together.

In cases when the aircraft are close to each other horizontally, it can be beneficial to base $b(\tau)$ on h , $\dot{h}_0$, and $\dot{h}_1$. The vertical distribution tables can be computed using the same process specified in Equation (10.8). The vertical dynamic model also assumes white-noise acceleration and can be represented using three state variables (i.e., h , $\dot{h}_0$, and $\dot{h}_1$).

10.4 Real-Time Execution

The real-time execution of ACAS X consists primarily of estimating the belief state and computing the associated state-action values by interpolating entries in the lookup table. However, there are certain situations in which the state-action values are modified online (i.e., during the execution of the logic). This section outlines these online costs and discusses the handling of multiple threats and traffic alerts.

10.4.1 Online Costs

As discussed in Section 10.2.2, the lookup table is a function of only six variables. However, several other variables (e.g., altitude above ground) need to be taken into account during execution that are not part of the offline optimization. Of course, these other variables could be added to the optimization, but it would be at the expense of larger table sizes.

Because early prototypes of ACAS X were constrained by table size, research explored alternatives to increasing the number of state variables in the offline optimization. Experiments showed that simply adding costs to the action values online can be effective. One of the first online costs explored in the development of ACAS X was for altitude inhibits. Legacy TCAS has rules that inhibit advisories on the basis of altitude to prevent disruption during landing. It was desired that these rules would be preserved in ACAS

X, and so if these rules trigger an advisory inhibit during flight, then infinite cost is added online to the appropriate actions.

Another example of an online cost is related to the coordination mechanism adopted from TCAS. When an aircraft issues an advisory against an intruder, it sends the intruder the sense of its advisory over the radio datalink. The intruder adjusts its advisory, if any, to be compatible (i.e., in opposite direction). In cases of simultaneous sense selection, ties are broken based on unique transponder identification numbers. If table size is not a concern, then the intruder sense could be incorporated into the offline optimization [17]. The offline optimization could account for the fact that the intruder will likely maneuver on the basis of the sense information and also account for the fact that the system cannot select incompatible senses in the future. Although an online cost approach is suboptimal in general, experiments have shown that this method is effective in practice [17].

The set of online costs incorporated in a prototype of ACAS X include

- *Altitude inhibit* prevents issuing advisories below certain altitudes.
- *Advisory switch or restart* penalizes advisory changes or restarts within a certain amount of time.
- *Initialization* prevents advisories from being issued during the first few seconds of the system's starting.
- *Multiple reversals* prevents multiple reversals (unless required by coordination).
- *Bad transitions* prevents advisory sequences not allowed by TCAS.
- *No-response vertical chase* penalizes continuing an advisory in a vertical chase scenario when the pilot is not responding.
- *Compatibility* prevents issuing advisories that are not compatible with the advisories issued by other aircraft.
- *Crossing* forces an advisory when an intruder issues an advisory that requires passing through the altitude of the own aircraft.

Some of these costs are effectively infinite (e.g., altitude inhibit), but others are relatively small (e.g., advisory switch and restart). The rules governing these online costs are implemented in code and can be arbitrarily complex without impacting memory requirements.

10.4.2 Multiple Threats

The MDP in Section 10.2 assumes a single intruder. It would be relatively straightforward to add additional state variables for each additional intruder, but the table size would grow exponentially with the number of intruders. Legacy TCAS determines the best advisory for each intruder in isolation and then relies on a relatively complex set of rules to combine these individual advisories to produce a single advisory to provide to the pilot.

ACAS X, in contrast with TCAS, fuses the state-action costs associated with different intruders [18]. Experimental results have shown that this method is effective and can scale to a large number of intruders [17]. In certain situations, the system will issue a multi-threat level off (MTLO) advisory, which does not belong in the action set available to the MDP. An MTLO can be issued if two different intruders indicate they are following different senses. Because an MTLO is neither an up sense or down sense, it can remain compatible with both intruders. MTLOs are useful in “sandwich” encounters in which an aircraft must fly between two other aircraft.

Unlike TCAS, ACAS X has the capability to provide different protection modes against different aircraft. Such a capability is especially useful in situations such as closely spaced parallel operations [19]. The system may want to adopt a less conservative alerting behavior against another aircraft that is known to be on a simultaneous approach but still provide the standard alerting behavior against all other aircraft. Early prototypes of ACAS X that implement this functionality use different lookup tables and state-estimation parameters for different protection modes and then fuse the state-action values together.

10.4.3 Traffic Alerts

The majority of the development effort on ACAS X has focused on the generation of RAs, but TAs play an important role in visual acquisition and preparing the pilots to respond to an RA. Several different approaches can be taken to produce optimized TAs with respect to the following objectives:

- Allow for appropriate lead time. The ideal time between a TA and RA is around 10 to 15 s. A lead time of less than 6 s is called a surprise RA and is unlikely to be sufficient.
- Avoid nuisance alerts. TAs without RAs should be avoided. Because of variability in pilot response, it is not possible to perfectly predict whether an RA will be issued in the future. Therefore, some TAs without RAs must be tolerated. Limiting nuisance TAs must be balanced with limiting surprise RAs.
- Avoid split alerts. The system should try to prevent multiple TA segments during a single encounter.

The approach taken in ACAS X does not require enlarging or modifying the current lookup table. TAs are generated based on the value of the “no-alert” action. The no-alert value is primarily influenced by the cost of collision, providing a measure of threat. The more likely a collision is, the lower the value of no alert. Comparing this value against a single threshold can cause chatter (i.e., excessive switching of the TA between on and off) when the value is near the threshold. To help prevent chatter in the original design of TCAS, TAs were required to stay on for at least 8 s. This requirement is carried over

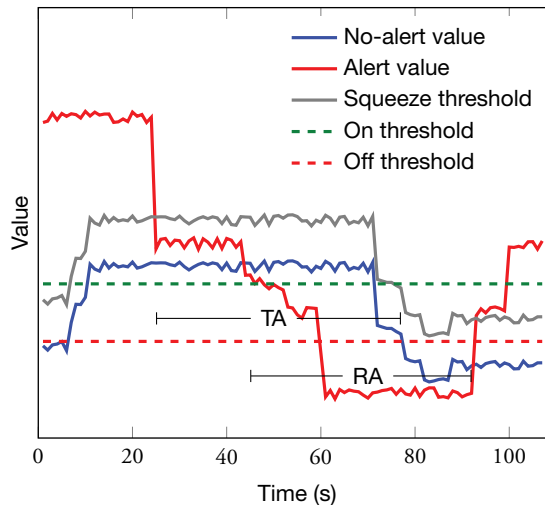


Figure 10.1 Notional TA behavior using on, off, and squeeze thresholds.

to ACAS X. To further prevent chatter and improve operational performance, ACAS X uses the following three thresholds:

- The *on threshold* is a constant value threshold. When the no-alert value descends below the on threshold, it is determined costly enough to allow a TA to be issued against the intruder.
- The *squeeze threshold* is a constant offset from the no-alert value. To help prevent excessive lead times or unnecessary TAs, this threshold allows the system to suppress TAs when no advisory values are close to the no-alert cost. If an advisory is less expensive than the no-alert cost offset by the squeeze threshold and the on threshold is also met, then a TA is issued against the intruder.
- The *off threshold* is a constant value threshold similar to the on threshold. If a TA has remained active for at least 8 s and the no-alert cost drops below the off threshold, then the TA will be discontinued.

These thresholds were chosen by running simulations of the system on radar data [20].

Figure 10.1 is a notional plot comparing the cost of the no-alert value (blue) and the highest-value advisory (alert value) (red) over the course of a single encounter. First, the no-alert cost rises above the on threshold, but the highest-value advisory is above the squeeze threshold, so a TA is not issued. Later, when the highest-value advisory descends below the squeeze threshold, a TA is issued. When the value of the highest-value advisory exceeds the value of no alert, an RA is issued [20].

10.5 Evaluation

ACAS X must accommodate many operational goals and constraints while meeting the established safety requirements. It is important that the system provide effective collision protection without unnecessarily disrupting pilots and the air traffic control system. In addition to producing as few alerts as possible, it must issue advisories that resolve encounters in a manner deemed suitable and acceptable by pilots and the operational community. This section discusses the process of safety and operational performance analysis, tuning of the logic, and flight tests of an ACAS X prototype.

10.5.1 Safety Analysis

Collision risk is estimated using an encounter model, which can be used to generate encounters that are representative of the airspace. Sampling a large collection of situations from an encounter model and running them in simulation both with and without a collision avoidance system provides an estimate of the differential in collision risk. Estimated risk depends strongly on the distribution of the encounters represented by the model. Hence, it is important that the encounter geometries and aircraft behavior represented by the model be as representative of the actual airspace as possible; otherwise, the risk associated with a collision avoidance system could be significantly over- or underestimated. To ensure a representative model, a large collection of recorded surveillance data is typically used to extract state probabilities over various encounter variables.

The encounter model used in the assessment of ACAS X is based on a U.S. radar data stream from 130 short- and long-range radars with 4.5 and 12 s update rates, amounting to approximately 10 GB of data per day. The raw reports provide range, azimuth, altitude, and transponder code. Those reports are converted to latitude and longitude coordinates, and tracks are fused from multiple sensors [21]. A database of tracks ranging from December 2007 to August 2008 was used to identify encounters that meet certain distance and time criteria, chosen to be less restrictive than the alerting criteria used by TCAS.

Associated with each of the 393,077 encounters identified are a set of static features, such as the horizontal miss distance and the initial vertical rates of the aircraft, along with a set of dynamic features, such as the turn rate and airspeed acceleration [22]. Bayesian network structure learning (Section 2.4) resulted in the topology of the initial and transition Bayesian networks shown in Figure 10.2. The distribution parameters were learned using the process outlined in Section 2.3.2.

Sampling from the model will produce encounters that are representative of the airspace. The collision avoidance system can then be run in simulation on these encounters to estimate collision risk. Safety studies typically involve simulating the aircraft as

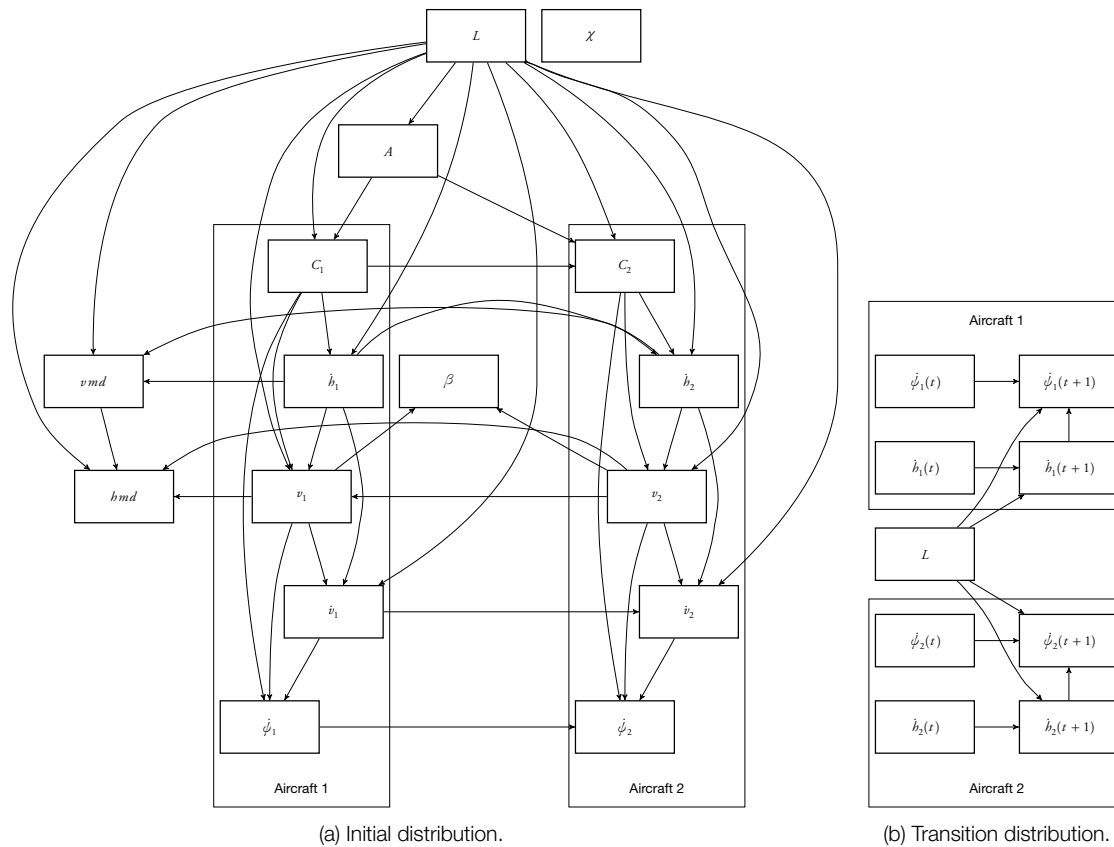


Figure 10.2 Airspace encounter model structure.

point masses and then estimating the probability of near midair collision (NMAC), defined to be when two aircraft come within 500 ft horizontally and 100 ft vertically. One of the most important metrics in safety analyses is *risk ratio*, the probability of NMAC with the collision avoidance system divided by the probability of NMAC without the collision avoidance system. If the rate of actual midair collision is to be estimated, then one must simulate aircraft wire frames and estimate distributions over aircraft types [23].

Directly sampling from the model and computing the average number of NMACs will provide an unbiased estimate of the probability that an encounter results in an NMAC. However, because of the rarity of NMAC events in the airspace, direct sampling from the encounter distribution will result in the generation of relatively few NMACs. Simulating encounters that are unlikely to result in an NMAC is inefficient. Instead, it is better to produce encounters that have mostly low vertical and horizontal separation at the time of closest approach and then weight the encounters appropriately [22]. Such an approach is known as *importance sampling* and has been widely studied in the literature as a variance-reduction technique in estimation [24]. The cross entropy method introduced in Chapter 4 has been applied to finding a suitable importance sampling distribution [25]. Even with importance sampling, generally on the order of hundreds of thousands of encounters are required to arrive at a risk ratio estimate with a relatively narrow confidence interval.

Assuming standard models for altimetry bias, active surveillance, and pilot response, the risk ratio for the current ACAS X prototype is less than 40% that of TCAS. Although the overall risk ratio for ACAS X is encouraging, ongoing work involves identifying and categorizing areas where ACAS X can be further improved. One area of study involves analyzing the safety of the system in European airspace where the encounter distribution is known to be different because of different air traffic control procedures. Besides analyzing failure cases in encounters models designed to be representative of an actual airspace, the development team studied logic performance on *stress testing* models. These stress testing models probe the limits of the system on exhaustive variations of certain classes of encounters [26]. Analysis of ACAS X has also been done using probabilistic model checking [27] and a hybrid systems theorem prover [28].

10.5.2 Operational Suitability and Acceptability

The operational performance of ACAS X is evaluated in simulation using real TCAS encounters collected under the FAA TCAS monitoring program by the TCAS Resolution Advisory Monitoring System (TRAMS). The data comprise more than 100,000 encounters that occurred during normal operations in 21 high-density terminal areas [29]. These encounters reflect all airspace classes, altitudes, domestic and foreign air carrier and business jet operations, enroute and terminal air traffic separation and pro-

cedures, airport arrival and departure routes, and a variety of intruder aircraft types and encounter geometries.

In addition to the TRAMS data, procedure-specific mini-models are also used to comprehensively assess current and future procedures of interest across a wide range of encounter dynamics [7]. These mini-models include procedures such as 500 ft and 1000 ft vertical separation encounters, closely spaced parallel approaches, and 3 nmi enroute separation procedures. As future air traffic procedures mature, additional models will be created to assess the safety and operational compatibility of ACAS X.

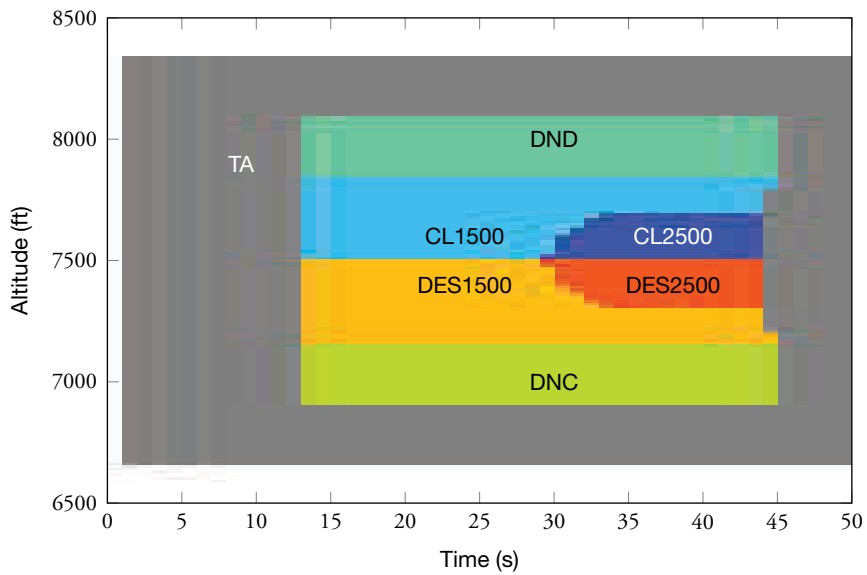
One of the key operational suitability metrics is the overall alert rate. The overall alert rate, as estimated from the TRAMS dataset, is 30% lower than that of TCAS. The reason that the alert rate is significantly lower than for TCAS can be seen in Figure 10.3. In these plots, both aircraft are level and are approaching head on. ACAS X has a much smaller alerting region than TCAS for this geometry. Except for the strengthening transition to increase climb or descend, all the alerts occur later than TCAS.

Ensuring that pilots will trust the ACAS X alerts is an important goal of logic tuning. During initial TCAS development, pilots identified that reversal and altitude crossing alerts warrant extra consideration because of their impact on flight crews. ACAS X performance was specifically evaluated in these areas to ensure equal or better performance than TCAS. Reversal performance was assessed with the TRAMS data, and numerous encounters were manually examined to ensure that they were acceptable. Overall, the current ACAS X prototype reduces reversals by 22%.

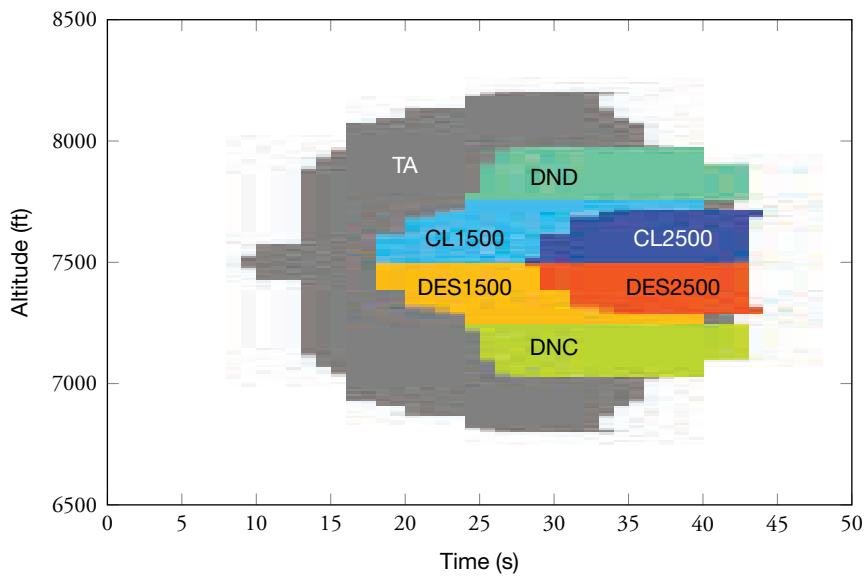
Due to the size of the TRAMS dataset, it is impossible to manually inspect every encounter; however, hundreds were examined over the nine logic iterations. Assessment of individual encounters is an important step in verifying that the logic tuning is effective in encouraging desired behavior. Figure 10.4 shows an example level-off encounter. The horizontal geometry (not shown) is a 90-degree crossing, common to many encounters.

The aircraft with a collision avoidance system initially descends, and the threat climbs and then levels off. TCAS issues an initial crossing descend alert and then reverses to a climb. After the climb alert, TCAS issues a “weakening” level-off, which is intended to minimize the altitude change when sufficient vertical separation has been achieved. Finally, the clear-of-conflict notification occurs well after the closest horizontal approach.

ACAS X, in contrast, waits a little longer than TCAS and issues a level-off. The clear of conflict then occurs shortly after the closest approach. In this example, ACAS X resolves the encounter without a crossing or reversal alert. The single level-off alert did not cause a deviation from the pilots’ intentions of leveling off, resulting in an acceptable resolution, while still providing safe vertical guidance.

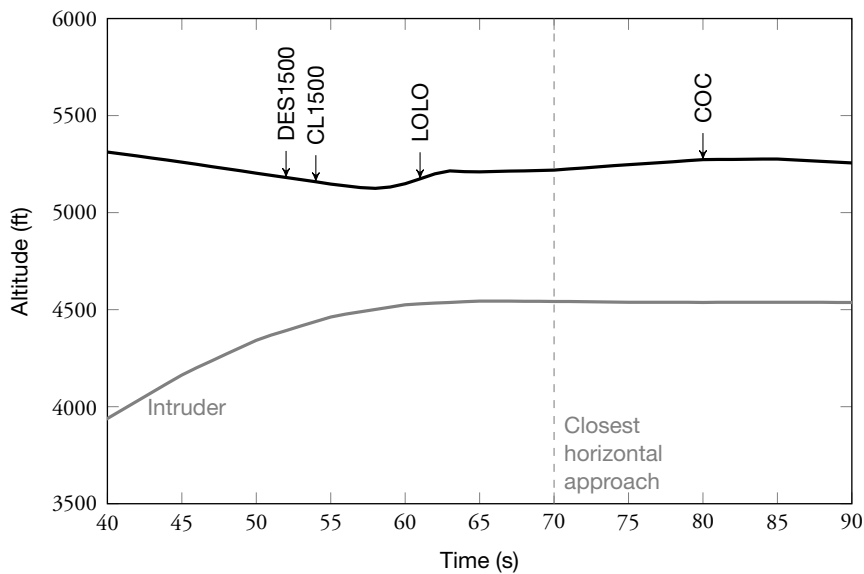


(a) TCAS.

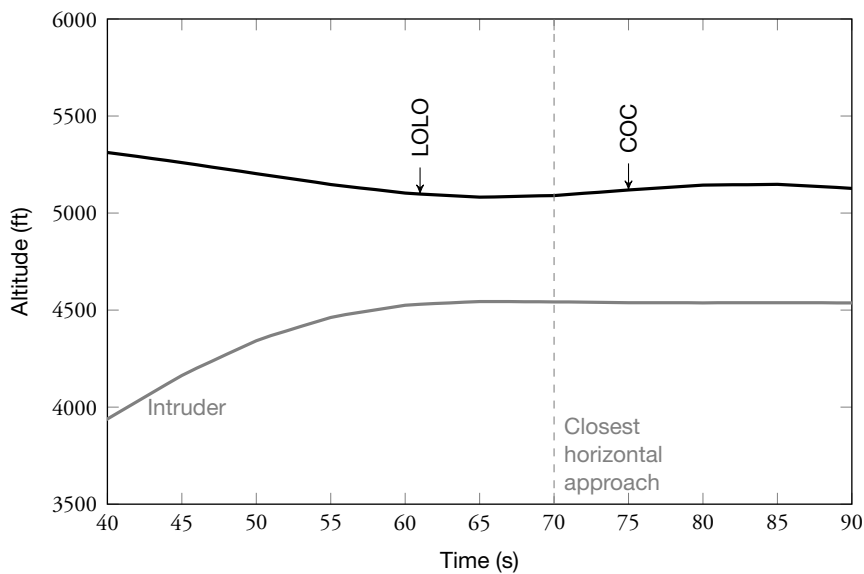


(b) ACAS X.

Figure 10.3 Logic plots.



(a) TCAS.



(b) ACAS X.

Figure 10.4 Example level-off encounter.

10.5.3 Parameter Tuning

A designer can modify many *design parameters* in ACAS X to achieve the goals of the system. Examples of design parameters include the offline cost of alert, the online cost of restarting an advisory, and the white-noise acceleration parameter used in the MDP. These design parameters can be adjusted to trade performance on different metrics. For example, the cost of alert could be increased to reduce the number of alerts but at the expense of decreased safety.

These design parameters should be distinguished from the *system parameters*. System parameters are the parameters that govern the behavior of the system but are not necessarily adjusted directly by the designer. In ACAS X, many of the design parameters are also system parameters, but the system parameters also include all the millions of values stored in the lookup table. Hence, there are many more system parameters than design parameters. The large number of system parameters allows the alerting behavior (as illustrated in Figure 10.3) to be more finely adjusted to achieve better performance.

One of the advantages of the design approach taken in ACAS X comes from the fact that the number of design parameters is small compared to the number of system parameters. In general, the complexity of the design process can grow exponentially with the number of design parameters. Each evaluation of a design point requires millions of simulations using a variety of different models. Even with a high-performance compute cluster with 64 nodes, evaluating a single design point requires an hour. In addition, given the results of a single evaluation, it can be challenging to predict which design point to try next. Hence, during the early development of ACAS X, there was tremendous interest in automating the process of tuning the design parameters.

To automate the process, there needs to be some scalar utility function u defined over the design space. If there are two design points θ_1 and θ_2 and $u(\theta_1) > u(\theta_2)$, then θ_1 is preferred to θ_2 . For ACAS X, u is based on the results of millions of different simulations generated by a variety of models. Because u is a scalar function, the various metrics must be weighted together appropriately. Although future work will explore the use of formal utility elicitation techniques to determine the weights (Section 3.1.4), the weights in the most recent tuning process were chosen by the consensus of an ad hoc committee of experts.

The optimization process is outlined in Figure 10.5. The first step involves *screening* the design parameters. Searching the design space is made easier by screening out the parameters that have little impact on the performance of the system. The elementary effects of the parameters were estimated by varying the parameters individually from their nominal value. The parameters that did not have a significant effect on the metrics were not included in the design search [19].

Once the important design parameters are identified, *surrogate model optimization* searches for the point in the design space that maximizes utility. Surrogate model

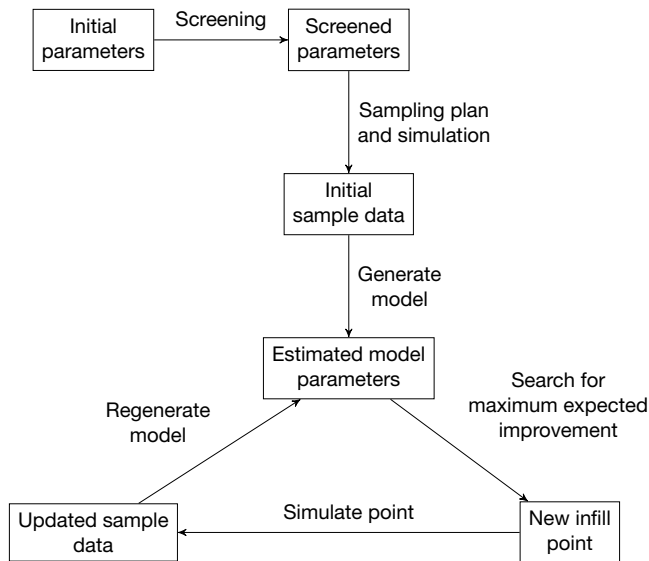


Figure 10.5 Overview of surrogate model optimization process.

optimization involves using past evaluations of design points to build a surrogate model of the utility function. Each step in the optimization process involves searching for the point in the design space that provides the maximum expected improvement according to the surrogate model. That point is then evaluated in simulation, and the model is updated using Bayes' rule. This process is repeated until a selection of high-performing design points are found. A similar surrogate optimization process has been used in a variety of other applications, including airfoil design [30].

10.5.4 Flight Test

ACAS X development takes advantage of modeling and simulation, but flight testing is still critical. Flight tests are required to validate simulation results and update models. They also expose the system to real environment challenges and are important in collecting pilot acceptability feedback.

The first flight test of ACAS X was conducted in August 2013 at the FAA William J. Hughes Technical Center in Atlantic City, New Jersey. MIT Lincoln Laboratory provided the algorithm specifications and lookup tables. Johns Hopkins University Applied Physics Laboratory implemented the algorithms on modified legacy hardware provided by Honeywell, Inc. The ACAS X box was flown on a Convair 580 using TCAS

beacon surveillance. Encounters were scripted with a Beechcraft King Air equipped with a Mode S transponder and another Convair 580 with TCAS.

The prototype unit operated successfully for more than 21 hours on 11 flights without any hardware or software failures. There were no unit resets necessary in flight. Because of memory constraints with the legacy hardware, the lookup tables were compressed using algorithms that did not significantly impact the timing of real-time lookups [9]. To ensure processing completed within each surveillance cycle, the logic processed only the four closest intruders. Research is currently underway to investigate ways of accommodating more intruders each processing cycle.

The logic was tested on a wide variety of different encounter scenarios, including those discussed in Section 10.5.2. In total, 127 test cards were flown. These encounters involved both traffic advisories and the various categories of resolutions advisories provided by TCAS. The encounters exercised the coordination mechanism with legacy TCAS. Encounters were flown both with and without a pilot response to the advisories.

ACAS X performed as desired in most of the test scenarios. However, there were a few areas in which the performance could be improved. Some undesirable alerts were issued in certain level-off encounters with 500 ft separation. Future rounds of optimization will aim to remove as many of these alerts as possible while still providing the safety required in blunder scenarios. Alerts were also observed during non-blunder parallel approaches. Improvements to surveillance anticipated in the final ACAS X system will help remove these alerts. The encounters also revealed known issues with the older version of the logic used for the test flight, such as desired reversals not being issued under certain circumstances. Some differences between the flight test and simulations were observed, resulting in modifications to the next iteration of analysis and optimization.

10.6 Summary

This chapter showed how the problem of aircraft collision avoidance can be modeled as a partially observable Markov decision process and solved using dynamic programming. Modeling and simulation reveal that such an approach can lead to a system that is less disruptive than TCAS while improving on the level of safety TCAS provides. This research has led to the establishment of the ACAS X, which is aimed to become the next international standard for collision avoidance. As with TCAS, the regulatory effort required for both U.S. and international acceptance is intensive. Following the flight test, the standardization process commenced with the federal advisory committee, the Radio Technical Commission for Aeronautics (RTCA). The system will play an important role in the next generation of commercial aviation, as well as support the safe introduction of unmanned aircraft in civil airspace.

References

1. S. Temizer, M.J. Kochenderfer, L.P. Kaelbling, T. Lozano-Pérez, and J.K. Kuchar, “Collision Avoidance for Unmanned Aircraft Using Markov Decision Processes,” in *AIAA Guidance, Navigation, and Control Conference (GNC)*, 2010.
2. M.J. Kochenderfer and J.P. Chryssanthacopoulos, “A Decision-Theoretic Approach to Developing Robust Collision Avoidance Logic,” in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2010.
3. T.B. Wolf and M.J. Kochenderfer, “Aircraft Collision Avoidance Using Monte Carlo Real-Time Belief Space Search,” *Journal of Intelligent and Robotic Systems*, vol. 64, no. 2, pp. 277–298, 2011. DOI: 10.1007/s10846-010-9532-6.
4. H. Bai, D. Hsu, M.J. Kochenderfer, and W.S. Lee, “Unmanned Aircraft Collision Avoidance Using Continuous-State POMDPs,” in *Robotics: Science and Systems*, 2011.
5. J.P. Chryssanthacopoulos and M.J. Kochenderfer, “Collision Avoidance System Optimization with Probabilistic Pilot Response Models,” in *American Control Conference (ACC)*, 2011.
6. M.J. Kochenderfer and J.P. Chryssanthacopoulos, “Robust Airborne Collision Avoidance Through Dynamic Programming,” Massachusetts Institute of Technology, Lincoln Laboratory, Project Report ATC-371, 2011.
7. J.E. Holland, M.J. Kochenderfer, and W.A. Olson, “Optimizing the Next Generation Collision Avoidance System for Safe, Suitable, and Acceptable Operational Performance,” *Air Traffic Control Quarterly*, vol. 21, no. 3, pp. 275–297, 2013.
8. S. Julier and J. Uhlmann, “Unscented Filtering and Nonlinear Estimation,” *Proceedings of the IEEE*, vol. 92, no. 3, pp. 401–422, 2004. DOI: 10.1109/JPROC.2003.823141.
9. M.J. Kochenderfer and N. Monath, “Compression of Optimal Value Functions for Markov Decision Processes,” in *Data Compression Conference*, 2013.
10. RTCA, *Minimum Operational Performance Standards for Traffic Alert and Collision Avoidance System II (TCAS II)*, DO-185B, 2008.
11. J.W. Andrews, “An Improved Technique for Altitude Tracking of Aircraft,” Massachusetts Institute of Technology, Lincoln Laboratory, Project Report ATC-105, 1981.
12. J.P. Chryssanthacopoulos and M.J. Kochenderfer, “Accounting for State Uncertainty in Collision Avoidance,” *AIAA Journal on Guidance, Control, and Dynamics*, vol. 34, no. 4, pp. 951–960, 2011. DOI: 10.2514/1.53172.

13. D.M. Asmar, M.J. Kochenderfer, and J.P. Chryssanthacopoulos, “Vertical State Estimation for Aircraft Collision Avoidance with Quantized Measurements,” *AIAA Journal on Guidance, Control, and Dynamics*, vol. 36, no. 6, pp. 1797–1802, 2013. DOI: 10.2514/1.58938.
14. A. Panken and M.J. Kochenderfer, “Error Model Estimation for Airborne Beacon-Based Surveillance,” *IET Radar, Sonar and Navigation*, vol. 8, no. 6, pp. 667–675, 2014. DOI: 10.1049/iet-rsn.2013.0266.
15. K.V. Ramachandra, *Kalman Filtering Techniques for Radar Tracking*. New York: Marcel Dekker, 2000.
16. M.J. Kochenderfer and J.P. Chryssanthacopoulos, “Partially-Controlled Markov Decision Processes for Collision Avoidance Systems,” in *International Conference on Agents and Artificial Intelligence (ICAART)*, 2011.
17. D.M. Asmar, “Airborne Collision Avoidance in Mixed Equipage Environments,” M.S. thesis, Massachusetts Institute of Technology, 2011.
18. J.P. Chryssanthacopoulos and M.J. Kochenderfer, “Decomposition Methods for Optimized Collision Avoidance with Multiple Threats,” *AIAA Journal on Guidance, Control, and Dynamics*, vol. 35, no. 2, pp. 398–405, 2012. DOI: 10.2514/1.54805.
19. K. Smith, M.J. Kochenderfer, W. Olson, and A. Vela, “Collision Avoidance System Optimization for Closely Spaced Parallel Operations Through Surrogate Modeling,” in *AIAA Guidance, Navigation, and Control Conference (GNC)*, 2013.
20. B. Puntin and M.J. Kochenderfer, “Traffic Alert Optimization for Airborne Collision Avoidance Systems,” in *AIAA Guidance, Navigation, and Control Conference (GNC)*, 2013.
21. M.J. Kochenderfer, L.P. Espindle, J.K. Kuchar, and J.D. Griffith, “Correlated Encounter Model for Cooperative Aircraft in the National Airspace System,” Massachusetts Institute of Technology, Lincoln Laboratory, Project Report ATC-344, 2008.
22. M.J. Kochenderfer, M.W.M. Edwards, L.P. Espindle, J.K. Kuchar, and J.D. Griffith, “Airspace Encounter Models for Estimating Collision Risk,” *AIAA Journal on Guidance, Control, and Dynamics*, vol. 33, no. 2, pp. 487–499, 2010. DOI: 10.2514/1.44867.
23. M.J. Kochenderfer, J.D. Griffith, and J.E. Olszta, “On Estimating Mid-Air Collision Risk,” in *AIAA Aviation Technology, Integration, and Operations Conference (ATTO)*, 2010.
24. R. Srinivasan, *Importance Sampling: Applications in Communications and Detection*. Berlin: Springer, 2002.

25. Y. Kim and M.J. Kochenderfer, “Improving Aircraft Collision Risk Estimation Using the Cross-Entropy Method,” in *ALAA Modeling and Simulation Technologies Conference*, 2015.
26. B.J. Chludzinski, “Evaluation of TCASII Version 7.1 Using the FAA Fast-Time Encounter Generator Model,” Massachusetts Institute of Technology, Lincoln Laboratory, Project Report ATC-346, 2009.
27. C. von Essen and D. Giannakopoulou, “Analyzing the Next Generation Airborne Collision Avoidance System,” in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2014.
28. J.-B. Jeannin, K. Ghorbal, Y. Kouskoulas, R. Gardner, A. Schmidt, E. Zawadzki, and A. Platzer, “A Formally Verified Hybrid System for the Next-Generation Airborne Collision Avoidance System,” in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2015.
29. W.A. Olson and J.E. Olszta, “TCAS Operational Performance Assessment in the U.S. National Airspace,” in *Digital Avionics Systems Conference (DASC)*, 2010.
30. A.I.J. Forrester, A.J. Keane, and N.W. Bressloff, “Design and Analysis of Noisy Computer Experiments,” *ALAA Journal*, vol. 44, no. 10, pp. 2331–2339, 2006. DOI: 10.2514/1.20068.

11

Multiagent Planning for Persistent Surveillance

N. Kemal Üre, Girish Chowdhary, Jonathan P. How, and John Vian

An important application of unmanned aircraft is persistent surveillance over areas of interest. For example, a team of aircraft can be used to monitor a forest for biological activities, a disaster flooded area for water levels, or a battle theater for movement. In such scenarios, surveillance must be persistent over an extended duration, and team members may be geographically distributed. It is important that the planning algorithms account for communication constraints and the health of the aircraft. This chapter presents a planning framework for multiagent cooperation and demonstrates the robustness of the approach in simulation and quadrotor flight tests.

11.1 Mission Description

To assess our multiagent planning algorithms, we engaged in a simulated mission to search for target vehicles while continuously tracking those that have been detected [1], [2]. The mission area is divided into three regions: base, communication relay area, and surveillance area. Aircraft start at the base area and travel to other regions for tasking and communication duties. As fuel depletes or failures occur, the aircraft return to base for refueling or repair. The communication area is a transition region between the base and surveillance areas. An agent is required to act as a relay link for communication to and from the base. The target vehicles to be tracked are located in the surveillance area.

Persistent surveillance presents several challenges:

- *Communication relay constraints.* In many applications of autonomous systems, it is necessary to maintain a communications link between the agents performing the mission and a base. This link may be used by human operators or ground-based autonomous planning systems to send commands to the agents, or to collect and analyze real-time sensor data from the agents. For example, in a search-and-rescue mission with camera-equipped aircraft, a human operator may need to observe the real-time video feeds from each aircraft to determine probable locations of the

party to be rescued. In many cases, the surveillance area is out of range of the base, and so a communication relay must be established.

- *Fuel constraints.* Each vehicle has limited fuel capacity and can therefore only operate for a limited amount of time in the communication or surveillance areas. If an aircraft runs out of fuel in either of these areas, then it cannot be recovered. The battery changing and charging station is located in the base area to refuel the aircraft. The rate at which fuel is depleted is stochastic.
- *System health characteristics.* Sensors are required for surveillance, and actuators are required for mobility. At any point during the mission, a sensor or an actuator may unexpectedly fail. When a sensor failure occurs, the agent becomes useless in the surveillance area. However, the agent may still serve as a communication relay. Upon returning to base, the sensor can be repaired. When an actuator is damaged, the agent becomes useless for performing any of the tasks in the mission, but the actuator can be repaired at the base.

The approach described in this chapter is a form of model-based reinforcement learning (as introduced in Section 5.2). The planner is initialized with a guess for the model of the system. The planner uses this model up to some horizon to generate a policy. This policy is then executed in the actual environment for some number of steps, and the observations are fed to a model learning algorithm. The process repeats with the planner generating a new policy using the updated model. Because the planning horizon is limited, the planning algorithm needs to be fast enough to update the current policy based on the most recent model. Interacting with the real environment is usually expensive, and so the algorithm must be data-efficient. This chapter presents algorithms that address the planning needs for persistent surveillance missions.

11.2 Centralized Problem Formulation

As introduced in Section 7.3.3, an MMDP is a multiagent Markov decision process. This section presents an MMDP formulation of the persistent surveillance problem [2], [3]. The formulation requires specifying the state space, action space, state transition model, and reward function.

11.2.1 State Space

The state space $\mathcal{S}$ can be decomposed into the product of the state space of the individual agents. If $\mathcal{S}_i$ is the state space of agent i , then $\mathcal{S} = \prod_i \mathcal{S}_i$. The state of each agent is determined by three discrete variables describing the location, fuel remaining, and health status. The location of agent i is denoted y_i , with

$$y_i \in Y = \{Y_B, Y_C, Y_S\}, \quad (11.1)$$

where Y_B is the base, Y_C is the communication area, and Y_S is the surveillance area.

The fuel state of agent i is denoted f_i , with

$$f_i \in F = \{0, \Delta f, 2\Delta f, \dots, F_{\max} - \Delta f, F_{\max}\} \quad (11.2)$$

where Δf is an appropriate discrete fuel quantity.

The health status of agent i is denoted h_i , with

$$h_i \in H = \{H_{\text{nom}}, H_{\text{sns}}, H_{\text{act}}\}, \quad (11.3)$$

where H_{nom} , H_{sns} , and H_{act} represent nominal health, a failed sensor, and a damaged actuator, respectively.

If there are n agents, then the total size of the state space is given by $|\mathcal{S}| = (|Y| \times |F| \times |H|)^n$. As discussed later, it is difficult to solve for an optimal strategy with more than a few agents because of the explosion in the size of the state space.

11.2.2 Action Space

The actions available to agent i depend on that agent's current location y_i and its remaining fuel f_i :

$$a_i \in \begin{cases} \{A_B, A_R, A_S\} & \text{if } y_i = Y_C \\ \{A_B, A_R\} & \text{if } y_i = Y_S \\ \{A_R, A_S\} & \text{if } y_i = Y_B \\ \{A_R\} & \text{if } f_i = 0 \end{cases}, \quad (11.4)$$

where A_B is move toward base, A_R is remain in location, and A_S is move toward surveillance area. The size of the action space is given by $|\mathcal{A}| = (|a_i|)^n = 3^n$.

11.2.3 State Transition Model

Given the fuel level, location, and action by agent i at time t , the location of that agent at time $t + 1$ is given by

$$y_i(t+1) = \begin{cases} y_i(t) & \text{if } f_i(t) = 0 \text{ or } a_i(t) = A_R \\ Y_B & \text{if } y_i(t) = Y_C \text{ and } a_i(t) = A_B \\ Y_S & \text{if } y_i(t) = Y_C \text{ and } a_i(t) = A_S \\ Y_C & \text{if } y_i(t) = Y_S \text{ and } a_i(t) = A_B \\ Y_C & \text{if } y_i(t) = Y_B \text{ and } a_i(t) = A_S \end{cases}. \quad (11.5)$$

The dynamics for the fuel state f_i are stochastic, with parameter P_{fuel} representing the probability of burning fuel at the nominal rate $\dot{F}_{\text{burn}}$. With probability $1 - P_{\text{fuel}}$, fuel is consumed at twice the nominal rate. If the agent is at the base, the fuel level increases at a rate of $\dot{F}_{\text{refuel}}$ until reaching $F_{\max}$.

The health state of each agent is also a stochastic model. If there is no fuel, then the health remains the same. If $y_i(k) = Y_B$, then $h_i(k+1) = H_{\text{nom}}$. If agent i is not at the base and its health is nominal at step k , then $h_i(k+1)$ is determined stochastically according to

$$h_i(k+1) = \begin{cases} H_{\text{nom}} & \text{with probability } (1 - P_{\text{sns}})(1 - P_{\text{act}}) \\ H_{\text{sns}} & \text{with probability } P_{\text{sns}}(1 - P_{\text{act}}) \\ H_{\text{act}} & \text{with probability } P_{\text{act}} \end{cases}, \quad (11.6)$$

where the probability of a sensor failure is P_{sns} and the probability of actuator failure is P_{act} .

11.2.4 Reward Function

The reward function encourages there to be some minimal number of aircraft in the surveillance region with a communication relay. If n_d aircraft are desired in the surveillance region but there are only n_s , then there is an immediate cost of $C_{\text{gap}} \times \max((n_d - n_s), 0)$, where C_{gap} is a small penalty for each missing aircraft. If the communication relay is broken, then a large penalty C_{fail} is assessed.

11.3 Decentralized Approximate Formulations

The MMDP formulation in the previous section completely captures all inter-agent interaction and can be used to arrive at the optimal policy through dynamic programming. However, this approach does not scale well with the number of agents. Even with three agents, the solution can be slow to compute [2]. This section describes two approximate problem formulations that can scale to problems where there are many agents. Both approaches involve computing a decentralized policy from a single-agent perspective given approximate information about the states and actions of other agents.

11.3.1 Factored Decomposition

The centralized problem decomposes the state space $\mathcal{S}$ into $\times_i \mathcal{S}_i$, where $\mathcal{S}_i = (Y \times F \times H)$. We can approximate the formulation from the perspective of agent i by defining $\mathcal{S}_j$ (for all $j \neq i$) to be the set of location-direction pairs associated with agent j . There are six location-direction pairs that are useful in this problem:

$$(Y_B, A_S), (Y_C, A_B), (Y_C, A_R), (Y_C, A_S), (Y_S, A_B), (Y_S, A_R). \quad (11.7)$$

The pair (Y_B, A_R) is not included because it is generally not useful to remain at the base. The size of the state space still grows exponentially with the size of the team, but the number of location-direction pairs is smaller than $|Y| \times |F| \times |H|$.

The state transition model for $\mathcal{S}_i$ from the perspective of agent i is identical to that of the centralized formulation. For $\mathcal{S}_j$, with $j \neq i$, the state of agent j from the perspective of agent i is approximated as follows:

$$s_{ij}(k+1) = \begin{cases} (Y_C, A_R) & \text{if } s_{ij}(k) = (Y_B, A_S) \text{ with probability } 0.5 \\ (Y_C, A_S) & \text{if } s_{ij}(k) = (Y_B, A_S) \text{ with probability } 0.5 \\ (Y_B, A_S) & \text{if } s_{ij}(k) = (Y_C, A_B) \\ (Y_C, A_R) & \text{if } s_{ij}(k) = (Y_C, A_R) \\ (Y_S, A_R) & \text{if } s_{ij}(k) = (Y_C, A_S) \\ (Y_C, A_B) & \text{if } s_{ij}(k) = (Y_S, A_B) \text{ with probability } 0.5 \\ (Y_C, A_R) & \text{if } s_{ij}(k) = (Y_S, A_B) \text{ with probability } 0.5 \\ (Y_S, A_R) & \text{if } s_{ij}(k) = (Y_S, A_R) \end{cases}. \quad (11.8)$$

11.3.2 Group Aggregate Decomposition

An alternative problem formulation approximates the behavior of all the teammates collectively with a single, reduced model [4]. Instead of tracking the locations and directions of each individual teammate, a more compact approximation involves tracking three features:

- whether there is at least one teammate predicted to be in the communication area,
- the predicted number of agents in the surveillance area, and
- the predicted number of healthy agents in the surveillance area.

The size of the state space from the perspective of a single agent is then only $|Y \times F \times H| \times 2n^2$, which grows only quadratically with the size of the team. To predict the positions of the teammates at the next time step, a single agent policy $\pi^{(n=1)}$ is used. In the experiments in this chapter, $\pi^{(n=1)}$ is set to be the solution of the single-agent centralized formulation without the communication relay requirement.

Although imitating the reward function from the centralized formulation is straightforward, specifying the transition probabilities for the three aggregate features is less clear. One approach is to create a state-transition table based on running the mission with five agents under a factored formulation [2]. When there are more than five agents, bicubic interpolation can be used [4].

11.3.3 Planning

Both decomposition methods presented above are from the perspective of a single agent. Because their corresponding MDPs have relatively low computational complexity, they can be solved via traditional dynamic programming algorithms, such as value iteration. The input to both the value function and policy is the decentralized state. From the

perspective of agent i , the decentralized is given by $\tilde{s}_i = \text{DecState}(\mathbf{s}, \mathbf{a}, i)$, where $\mathbf{s}$ is the centralized state and $\mathbf{a}$ is the collection of actions by all the agents.

Given $\tilde{s}_i$, the action taken by agent i is $a_i = \pi^*(\tilde{s}_i)$. It is important to note that the construction of the state $\tilde{s}_i$ depends on the prediction of the actions of the rest of the teammates using $\pi^{(n=1)}$. However, some performance degradation is expected because the predicted actions of teammates may be different from their actual actions. To account for the state-action coupling between agents, a planning scheme can be adopted by which each agent chooses its action in sequence. The resulting action is passed to the next agent in the sequence who can use it to compute its decision rather than attempting to predict it. The final agent calculates its action based on the actual actions of the rest of the team. This scheme is summarized in Algorithm 11.1. In this algorithm, the function v specifies the fixed ordering, where the i th agent to select its action is agent $v(i)$.

Algorithm 11.1 Ordered search

```

1: function ORDEREDSEARCH( $v, \pi^{(n=1)}, \mathbf{s}$ )
2:   for  $i \leftarrow 1$  to  $n$ 
3:      $a_{v(i)} \leftarrow \pi^{(n=1)}(\mathbf{s})$ 
4:   for  $i \leftarrow 1$  to  $n$ 
5:      $\tilde{s}_{v(i)} \leftarrow \text{DecState}(\mathbf{s}, \mathbf{a}, v(i))$ 
6:      $a_{v(i)} \leftarrow \pi^*(\tilde{s}_{v(i)})$ 
7:   return  $\mathbf{a}$ 

```

The choice of ordering v can influence the quality of the resulting policy. Hence, it may be desirable to try several different orderings at each step. These orderings can be sampled from the $n!$ permutations of length n . The best ordering to use can be determined from the expected utility. Algorithm 11.2 shows an example of a scheme that uses m different orderings $v_{1:m}$ and a generative model that samples the next state $\mathbf{s}' \sim T(\mathbf{s}, \mathbf{a})$ and reward $r \sim R(\mathbf{s}, \mathbf{a})$. Algorithm 11.3 shows a generalization of the sampled ordered search process up to depth d . The experiments in this chapter use $d = 2$.

11.4 Model Learning

The earlier sections presented an approach to multiagent planning under the assumption that the dynamics are known. In reality, the parameters of the dynamic cannot be known prior to the mission and must be learned online. The agents must estimate the state-correlated sensor failure probabilities from observed state transitions. Estimating and storing these transition probabilities are not feasible due to the size of the planning

Algorithm 11.2 Sampled ordered search

```

1: function SAMPLEDORDEREDSEARCH( $v_{1:m}, \pi^{(n=1)}, \mathbf{s}$ )
2:    $(\mathbf{a}^*, q^*) \leftarrow (\text{NIL}, -\infty)$ 
3:   for  $i \leftarrow 1$  to  $m$ 
4:      $\mathbf{a} \leftarrow \text{ORDEREDSEARCH}(v_i, \pi^{(n=1)}, \mathbf{s})$ 
5:      $\mathbf{s}' \sim T(\mathbf{s}, \mathbf{a})$ 
6:      $r \sim R(\mathbf{s}, \mathbf{a})$ 
7:      $\tilde{\mathbf{s}}_1 \leftarrow \text{DECSTATE}(\mathbf{s}', \mathbf{a}, 1)$ 
8:     if  $r + U^*(\tilde{\mathbf{s}}_1) > q^*$ 
9:        $(\mathbf{a}^*, q^*) \leftarrow (\mathbf{a}, r + U^*(\tilde{\mathbf{s}}_1))$ 
10:  return  $(\mathbf{a}^*, q^*)$ 

```

Algorithm 11.3 Forward sampled ordered search

```

1: function FORWARDSAMPLEDORDEREDSEARCH( $v_{1:m}, \pi^{(n=1)}, \mathbf{s}, d$ )
2:   if  $d = 0$ 
3:     return  $(\text{NIL}, 0)$ 
4:    $(\mathbf{a}^*, q^*) \leftarrow (\text{NIL}, -\infty)$ 
5:   for  $i \leftarrow 1$  to  $m$ 
6:      $\mathbf{a} \leftarrow \text{ORDEREDSEARCH}(v_i, \pi^{(n=1)}, \mathbf{s})$ 
7:      $\mathbf{s}' \sim T(\mathbf{s}, \mathbf{a})$ 
8:      $r \sim R(\mathbf{s}, \mathbf{a})$ 
9:      $(\mathbf{a}', q') \leftarrow \text{FORWARDSAMPLEDORDEREDSEARCH}(v_i, \pi^{(n=1)}, \mathbf{s}', d - 1)$ 
10:    if  $r + q' > q^*$ 
11:       $(\mathbf{a}^*, q^*) \leftarrow (\mathbf{a}, r + q')$ 
12:  return  $(\mathbf{a}^*, q^*)$ 

```

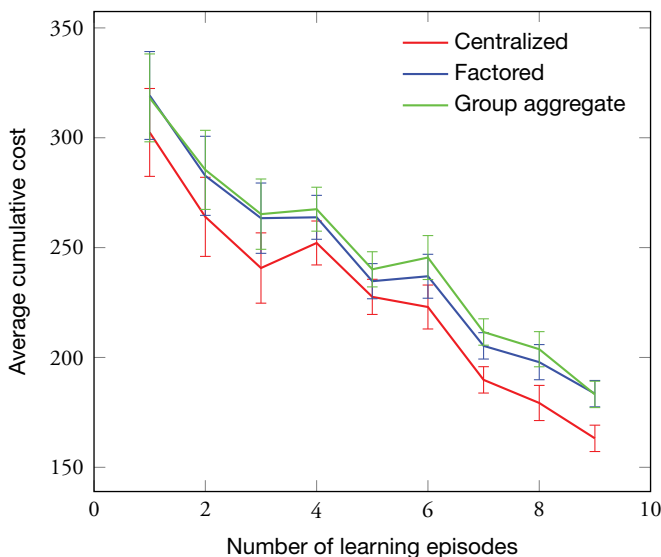


Figure 11.1 Planning and learning performance of different approaches. Each point in the plot of the specific planner shows the average cost accumulated in between each learning update.

space, and so approximation is required. For this work, we employed an incremental feature dependency discovery algorithm. This algorithm adjusts the flexibility of the approximation automatically based on observed transitions, freeing the designer from hand coding a fixed approximation structure. The details of the algorithm can be found in a paper by Geramifard et al. [5]. Applications to learning state-correlated uncertainties can be found in the work by Ure et al. [6].

Prior work involved estimating the probability of sensor failure under the assumption that this probability is uniform throughout the state space [7]. However, it was found that it is important to take into account the correlation between states and the probability of sensor failure. For instance, an aircraft in the surveillance area may perform more aggressive maneuvers and may operate in a more hostile and uncertain environment while tracking its target. Therefore, it may have a higher probability of sensor failure in the surveillance area compared with the probability of sensor failure in the communication or base areas. Similarly, an aircraft low on fuel has a tighter power budget, which may lead to a higher probability of sensor failure or sensor disablement.

To assess the effectiveness of integrating model learning with the various planning algorithms, we ran a set of 30 simulations. Each simulation involved nine iterations of learning and planning. The average cumulative cost after each iteration is plotted in Figure 11.1. The error bars denote the standard deviation in the cost. The learning

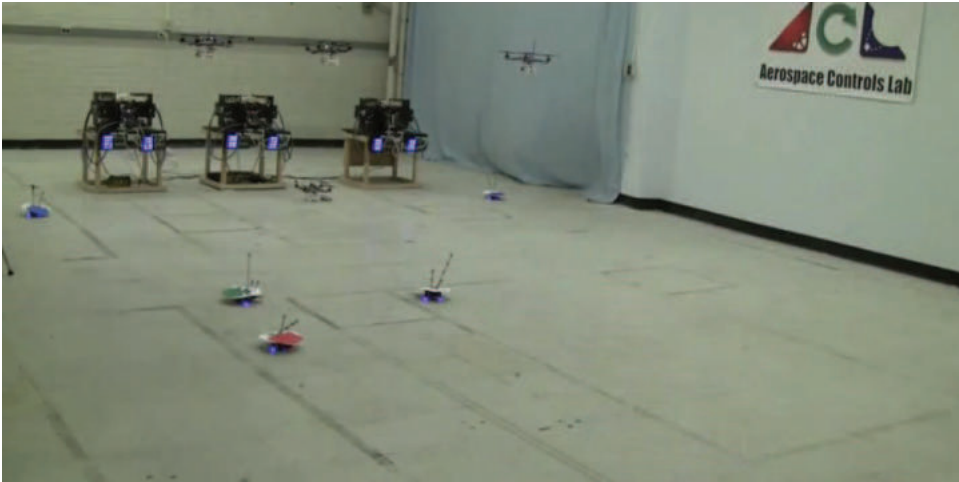


Figure 11.2 The experiment setup shows the quadrotors and ground vehicles. In the background are the three recharge stations used to enable persistent flight operations.

algorithm results in better performance as experience is accumulated. Factored decomposed planning provides performance within 5% to 7% of the centralized planning performance. The group aggregate performance is around 2% to 3% of the factored decomposition performance despite being much more computationally efficient.

11.5 Flight Test

The persistent surveillance missions were executed at the MIT Aerospace Control Laboratory's RAVEN test environment [8], [9]. The RAVEN test area is equipped with a Vicon motion capture system, which provides accurate position and velocity information. The flight vehicles have attitude stabilization loops that use the onboard gyros and accelerometers to estimate attitude. Three quadrotors are the agents performing the mission [2]. The vehicles are shown in Figure 11.2. The quadrotors are battery powered. When fully charged, the batteries are capable of powering the aerial vehicles for 8 to 10 minutes. Therefore, three automated recharge stations are implemented in the experiment area to enable multiple-hour missions.

Performance depends on both the performance of the planner and learning of the uncertainty. In particular, the planner performance is expected to improve as uncertainty is reduced through the use of incremental feature dependency discovery. The uncertain parameter in this case is the probability of sensor failure, and for purposes of these flight tests, it is assumed to be constant (0.05) throughout the state space, which results in a simple state-independent uncertainty model.

Table 11.1 Estimated probability of sensor failure.

Location	High Fuel	Low Fuel
Base	0.0	0.0
Communication	0.067	0.132
Surveillance	0.123	0.351

The results in Figure 11.3 consist of three subplots. The top plot shows the sum of the accumulated cost incurred by the cooperative planner for each agent in the mission. Lower cost is better, and the slope of the lines in this plot represent the rate at which cost is incurred. The middle plot is a filtered piecewise derivative of the top plot and provides a notion of how fast cost is incurred. The bottom plot shows a single agent's estimate of the probability of experiencing a sensor failure. It can be seen that as the estimate of the uncertainty improves, cost is incurred at a lower rate. These flight results demonstrate the desired interaction between the planning and learning algorithms and the system's ability to learn from experience and improve the overall performance.

The autonomous flight test lasted for three hours, during which about 120 total autonomous battery swaps were performed by the three recharge stations. The learning framework was pessimistically initialized with 30% sensor failure across all states. Table 11.1 shows the parameters of the learned state-correlated sensor-failure model. After every learning update, the policy was recomputed. Figure 11.4 displays the performance of the updated policy after each learning cycle in terms of average cumulative cost.

Due to the pessimistic initialization of the failure model, the initial policy has a high cost because it calls the quadrotors back to the base frequently for repair. As the learning process arrives at better estimates of the parameters, the planning algorithm becomes more confident in the ability of the aircraft to operate without failures and assigns them more efficiently between the base and tasking areas. Figure 11.5 shows how the learning and planning process results in fewer battery swaps in the second half of the mission.

11.6 Summary

This chapter presented an experimental demonstration of a multiagent planning framework for persistent missions. The planning algorithms were formulated as solutions to Markov decision processes that approximate the team dynamics. The decomposition approximation significantly reduces the required computation with only a small sacrifice in performance. Learning was used to update the model parameters through observed state transitions. The overall approach was validated on a persistent search and track testbed with unknown sensor failure dynamics.

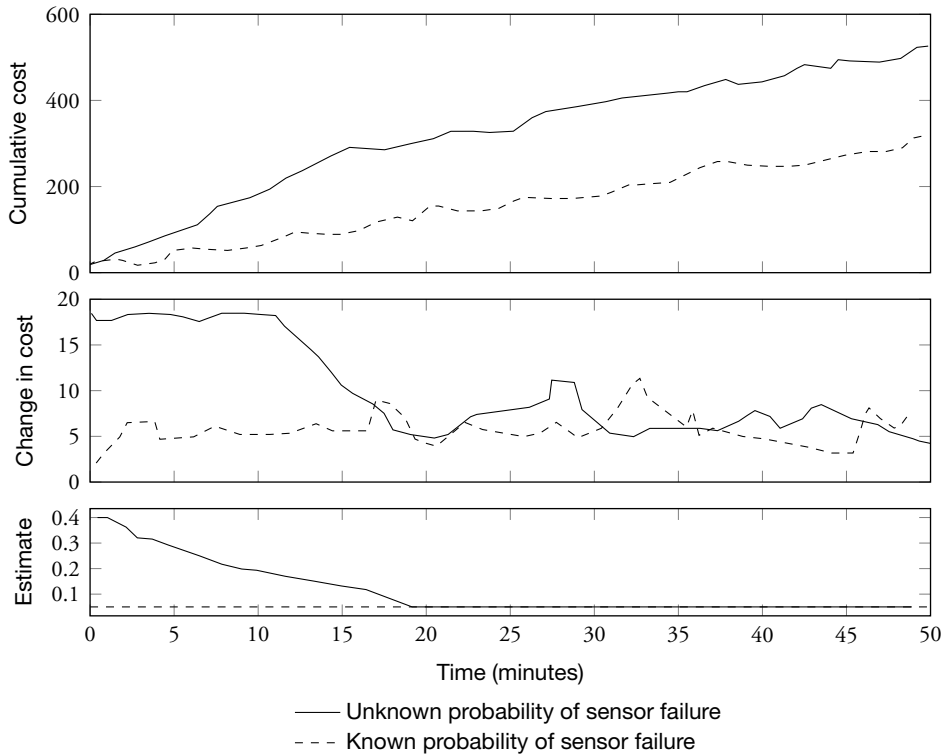


Figure 11.3 Flight test results of the persistent surveillance mission with planning using factored decomposition. The top plot shows accumulated cost over the course of the mission. In the middle, a filtered piecewise derivative of the top subplot provides a notion of how fast costs are being incurred. Finally, the bottom plot shows an estimate of the probability of experiencing a sensor failure, which is updated over time by the learning algorithm. Solutions to the cooperative planner formulation are generated online as the uncertainty in the model parameter decreases, thereby improving subsequent plans.

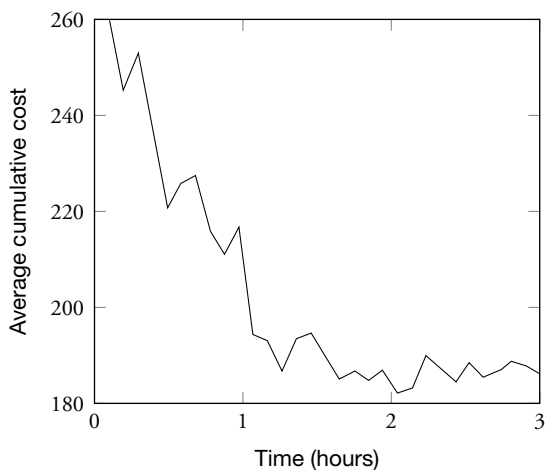


Figure 11.4 Flight-test results for decentralized factored planner and incremental feature dependency discovery. The plot shows the decrease of average cumulative cost as a result of the learning process.

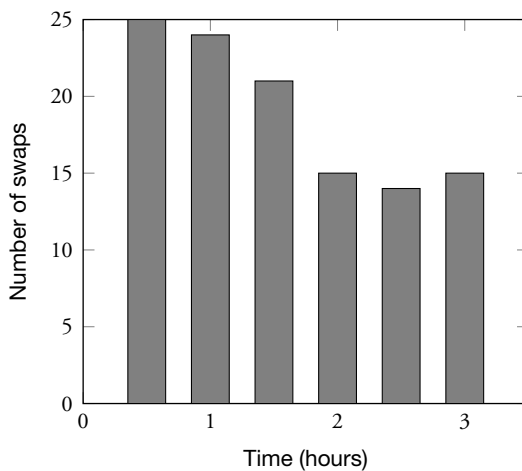


Figure 11.5 Number of battery swaps as a function of time. The decrease in swaps indicates more efficient coordination between the base and surveillance area.

References

1. B. Bethke, J.P. How, and J. Vian, “Multi-UAV Persistent Surveillance with Communication Constraints and Health Management,” in *AIAA Guidance, Navigation, and Control Conference (GNC)*, 2009.
2. J.D. Redding, T. Toksoz, N.K. Ure, A. Geramifard, J.P. How, M. Vavrina, and J. Vian, “Persistent Distributed Multi-Agent Missions with Automated Battery Management,” in *AIAA Guidance, Navigation, and Control Conference (GNC)*, 2011.
3. J.D. Redding, “Approximate Multi-Agent Planning in Dynamic and Uncertain Environments,” Ph.D. dissertation, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, Cambridge, MA, 2012.
4. J.D. Redding, N.K. Ure, J.P. How, M. Vavrina, and J. Vian, “Scalable, MDP-Based Planning for Multiple, Cooperating Agents,” in *American Control Conference (ACC)*, 2012.
5. A. Geramifard, F. Doshi, J. Redding, N. Roy, and J. How, “Online Discovery of Feature Dependencies,” in *International Conference on Machine Learning (ICML)*, 2011.
6. K. Ure, A. Geramifard, G. Chowdhary, and J.P. How, “Adaptive Planning for Markov Decision Processes with Uncertain Transition Models via Incremental Feature Dependency Discovery,” in *European Conference on Machine Learning (ECML)*, 2012.
7. B.M. Bethke, “Kernel-Based Approximate Dynamic Programming Using Bellman Residual Elimination,” Ph.D. dissertation, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, 2010.
8. M. Valenti, B. Bethke, G. Fiore, J.P. How, and E. Feron, “Indoor Multi-Vehicle Flight Testbed for Fault Detection, Isolation, and Recovery,” in *AIAA Guidance, Navigation, and Control Conference (GNC)*, 2006.
9. J.P. How, B. Bethke, A. Frank, D. Dale, and J. Vian, “Real-Time Indoor Autonomous Vehicle Test Environment,” *IEEE Control Systems Magazine*, vol. 28, no. 2, pp. 51–64, 2008. doi: 10.1109/MCS.2007.914691.

12

Integrating Automation with Humans

Hayley J. Davison Reynolds

The focus of this book has been on computational methods for decision making. The successful application of these methods to the building of decision support systems requires special consideration of the humans using the systems. This chapter discusses the challenges of human-systems integration and provides strategies for effective implementation.

12.1 Human Capabilities and Coping

Often the human-systems integration is an afterthought, if not completely trimmed in the budget. Designers depend on the fact that humans are generally flexible and adaptable to the brittle nature of technology. This section explores some of the perceptual and cognitive capabilities of humans and coping strategies.

Table 12.1 is a consolidated list of specific design considerations with respect to the information presented in this section. These are not meant to be inclusive or in any way replace the value that a human factors professional would have on the design. They simply point out best practices and commonly ignored issues that tend to pose problems for the system during implementation, which can provide a starting point for algorithm designers and development specialists.

12.1.1 Perceptual and Cognitive Capabilities

Three of the core capabilities to process information are attention, cognition, and memory [2], [3]. Information is perceived by the human, and the attention capability allows a portion of this perceived information to be processed. The portion of the information that is attended to is then cognitively processed by the human, using memory as an aid to make sense of the information. Each capability is described in more detail below.

Table 12.1 Design considerations and suggestions: Human capabilities and coping.

Work Analysis

- Perform a cognitive work analysis, with particular focus on the decisions that the system is supposed to be aiding to identify the context within which the system must function [1]. This analysis includes identifying the information provided, cognitive processing requirements, decisions made, and how the decisions are made actionable. This step aids the designer in identifying what aspects of the human's processing capabilities are already spoken for.
-

Attention, Memory, and Cognitive Processing

- Ensure the human operator is not required to pay attention to too many items at once.
 - Consider allocating information presentation appropriately across different modalities (auditory or visual) and processing codes (spatial or verbal) to ensure information is attended to as it is needed.
 - Limit the amount of information an operator is required to remember on a short-term basis.
 - Provide structured training to ensure that any lengthy and complex information about the system is conveyed to the operator (and stored appropriately in long-term memory).
 - Make use of appropriate previously learned information, procedures, and contextual norms and practice to take advantage of top-down processing.
 - In unfamiliar situations, provide appropriate cues to aid the operator in bottom-up processing.
-

Decision Making

- Identify whether there is enough information and time to support classical decision making in the operational environment. (Hint: The chances are good that there is not.)
 - Use the cognitive work/task analysis information to better understand the operator mental models of the system aspects of interest to the designer.
 - Write a short, functional concept of operations for the decision support system that identifies how you, as the designer, expect the user to understand and use the system. Take into account user mental models of your system and others.
 - Consider how your expectations of how the operator's make decisions with your system might be affected by common decision-making heuristics (e.g., availability, representativeness, anchoring).
 - Recognize current procedures and if or how they need to change with the implementation of your system.
-

Attention is a capability that allows the human to process only a limited set of information at a time [2]. Three metaphors in the psychological literature have been used to describe attention: a filter, a spotlight, and pools. What information is focused on can change depending on the salience of the information or how the human actively directs attention [4]. The flashing light on a control panel draws the operator's attention to information with unusual readings. The ability of the person to listen to a quiet story being told in a crowded room at a party also illustrates this filter capability.

It has also been demonstrated in the psychological literature that there are multiple modalities that have a certain amount of attention capacity [5]–[7]. Humans can effectively distribute attention between different modalities (e.g., auditory and visual), processing codes (e.g., spatial and verbal), and processing stages (e.g., perception, cognition, and response). For example, it is easier to attend to math homework while listening to music with lyrics than it is to attend to reading a novel while doing the same. Demands on verbal resources required to read a novel and comprehend lyrics can result in either having to reread sections of the novel or learning to ignore the music. Likewise, it is easier for a driver to carry on a conversation while driving a road that is familiar (requiring few perception and cognition resources but mostly response resources) than it is carrying on the conversation in an unfamiliar city or road segment, which can overload the perception and cognition attention resources.

Memory is another core capability that has limitations but also extraordinary abilities if used in the right way. Studies of short-term memory have arrived at a number of 7 ± 2 items for a typical human memory capacity of unlinked individual items [8]. However, experts in memory challenges have the ability to memorize hundreds of unrelated items in a particular order [9]. Experts who memorize items with functional relationships with one another can remember significantly more information than even the experts in memory challenges [10]. This capability to expand capacity with expertise is critical for complex-system operators such as pilots and air traffic controllers, who must remember hundreds of pages of procedures and standards to perform their daily activities in a time-critical environment. Once information has successfully passed from short-term to long-term memory through practice or some particular significance attached to the information, items can be remembered throughout a person's life. However, even long-term memory can decay [2]; thus, periodic refresher training of critical information becomes important.

A majority of the information processing strength occurs within the cognitive processing capability. A significant amount of time and space could be dedicated to this capability alone, but in this chapter, the focus will be on the bottom-up versus top-down processing that occurs within cognitive processing. Babies and people dropped into foreign situations use almost exclusively bottom-up processing. In bottom-up processing, attention is drawn to the salient features of the environment, and sense-making on these salient features occurs [11]. “What is going on here?” is the prevalent question of

bottom-up processing. As children and adults develop expertise in an environment, they have previous experiences residing in the memory to aid in this sense-making process. Top-down processing allows the human to begin searching for information to confirm or disaffirm hypotheses about what is going on in the environment [12], [13]. The prevalent question in top-down processing is, “Is this what I think it is?” Presented with a set of three lights—one red, one yellow, and one green—stacked on the top of one another, most adults in the United States can make the assumption, “Oh, this is a traffic light. I am now on a road that requires me to pay attention to what color is lit on the traffic light. Red means stop. Green means go. Yellow means caution, the light is about to turn red.” In more complex environments, the top-down processing occurs when certain states of a system are associated with certain temporal patterns of information on a display or control console. For example, with information from two radar sweeps in their trained sectors, experienced air traffic controllers can detect whether there are any potential separation violations.

12.1.2 Naturalistic Decision Making

Once information has been processed and understood, humans then have decisions to make about their response to the environment. The field of decision making has reflected two modeling paths—classical decision theory (a normative model of how decisions “should” be made) and naturalistic decision making (descriptive models of how decisions are “actually” made by humans) [14]. Classical decision making relies on quantitatively evaluating the utility of possible decision choices (see Chapter 3 and reference [15]). While this is a rational and unbiased means of making a decision, the reality is that (1) the human may not have access to the values to populate this equation, and (2) there may not be time to acquire the information or cognitive resources to perform this process.

Alternatively, descriptive models of decision making have identified ways that decisions are actually made by humans. *Satisficing* is a common method used in which humans do not attempt to conceive all of the potential choices but then make a decision on a choice that is “good enough” [16]. In deciding to take an alternate flight route around weather, a pilot will not outline every possible flight plan and then evaluate them all one by one. He or she will likely find a route with minimal deviations from the original flight route that is not impacted by the weather, even though this choice may not end up being the “optimal” flight route.

Multiple heuristics have been identified by Kahneman, Slovic, and Tversky that simplify the decision-making process [17]. The “availability” heuristic suggests that humans will recall most often the choice that was considered most recently or most frequently. The “representativeness” heuristic suggests humans will expect a choice to be as likely as other choices that have similar properties. The “anchoring” heuristic

suggests that an initial numerical estimate will provide the initial “anchor” and bias any subsequent estimations in the direction of the initial estimate. These heuristics (and others [2]) reduce the load on working memory and cognitive processing in both daily and more complex situations. However, these heuristics have the disadvantage of producing potentially “suboptimal” but efficient and “good enough” results for the humans.

The scientists who produced descriptive models of human decision making were followed by others who began extensively studying the decision making of experts in situ. These experts worked in the fields of firefighting, aviation, medicine, and warfighting. Their typical decision-making situation was characterized by time pressure, high stakes, extensive training, inadequate information, dynamic conditions, and team coordination [18]. The most revealing statement from one of these experts about his decision process was, “I don’t make decisions. I don’t remember when I’ve ever made a decision” [19]. In other words, there is one choice that is obvious to the expert, revealed as successful through hundreds or thousands of similar experiences. This pattern matching of the situation cognitively linked to previous analogous experience to quickly and accurately produce a satisfactory course of action is termed recognition-primed decision making.

Recognition-primed decision making is built on experts’ mental models of the systems with which they have extensive experience. Mental models are representations of an actual system that allow the human to take information from the environment and predict how the situation will evolve into the future [20], [21]. Firefighters have mental models of the evolution of the spread of fire in different types of buildings. Weather forecasters have mental models of convective (thunderstorm) development and decay. Pilots have mental models of the aircraft and its response to different control inputs. These mental models must be simple enough to allow the expert to run mental simulations with different inputs quickly; however, they must be elaborate enough to adequately describe the complexity of their particular system. These models must also be able to be adapted to analogous situations. For example, a pilot who has experience in a Boeing aircraft may be able to use his or her mental model to predict how an Airbus aircraft would react to a control input. (In some cases, this mental model would be more successful in correct prediction than others.) With experience, an expert learns which pieces of the mental model are critical to correct prediction and which can be dropped to reduce cognitive workload.

Another way that expert users simplify the situation to ensure that the cognitive processing is manageable is by using the structure of the context to constrain the possible outcomes of the situation. Structure is defined as the shared knowledge about the functioning of a system that inherently limits the evolution of the state of a system [22], [23]. This structure can be an inherent property of the system—for example, the maximum ascent profile of a Boeing 737 aircraft. Or this structure can be artificially imposed, such as the arrival and departure procedures into and out of the New York

metro airports. By knowing this structure, a pilot knows that an aircraft cannot reach 30,000 feet in one minute. An air traffic controller knows that if the departure procedures out of New York's LaGuardia Airport are issued (and conformed to), then this aircraft will not violate separation restrictions with any arrivals or departures from Newark or John F. Kennedy International Airports. Knowledge of the structure of a system plays a valuable role in both predicting the evolution of a system and eliminating impossible outcomes that clog valuable cognitive resources.

12.2 Considering the Human in Design

The previous section described a number of the capabilities and limitations that humans exhibit. The consequences of these capabilities and limitations to system design will be discussed in this section. Table 12.2 provides a set of design considerations with respect to trust, information uncertainty, and decision making over long timescales.

12.2.1 Trust and Value of Decision Logic Transparency

The first issue that must be overcome when deploying a decision support system to the field is that the users must be able to trust the information and recommendations the system is providing [24], [25]. The definition of trust used here is selectively adapted from Muir [25]. Trust is the expectation held by a member of a system that there will be reliability and technically competent performance from another member of the system, which is related to, but not necessarily isomorphic with, objective measures of these qualities. "Reliability" refers to consistent and predictable performance. "Technically competent" refers to the capable execution of functions allocated to that member within certain specified boundaries and constraints. The fact that trust is not directly representative of these qualities indicates that there is a difference based on the human's perception of these attributes, which may be biased. Bias may be due to the innate cognitive processing limitations discussed in the section above, lack of adequate information to accurately represent technical competence and reliability, or "willful miscalibration of trust." Willful miscalibration of trust can result when a human either over-relies on a system because of conscious misgivings about his or her own capabilities or under-reliance on the system due to general mistrust in automation or fear of being replaced [24]. Other forms of trust miscalibration are discussed later in this section.

The two aspects required to develop appropriate trust in a decision support system are

1. understanding of the capabilities (and limits of these capabilities) of the system, and
2. knowledge about the reliability of the information and recommendations provided by the system.

Table 12.2 Design considerations and suggestions: Considering the human in design.

Trust

- Convey thresholds of system limitations in the design of the system (e.g., in the display) if possible.
 - Design system to convey a mental model of the system to the operator that is consistent with the function of the system.
 - Provide data about reliability and data about confidence of the system if the operator is able to use this information during the decision process.
 - Provide opportunities for frequent feedback on the system's decision support performance accuracy to build trust in the system.
 - Ensure that the human operator's role in the system is an appropriate human role that will not cause the operator to fall prey to unnecessary stress or fatigue.
-

Certainty and Uncertainty

- Attempt to provide reliable information.
 - Provide stable information and recommendations that will not frustrate the user by oscillating wildly during a decision-making process.
 - Display the rationale behind the decision support recommendation when it is possible for the operator to process this information.
 - When the system's recommendation is uncertain, provide supplementary information to aid the operator in determining a course of action.
-

Decisions over Long Timescales

- Characterize the error of the decision support recommendations over the time frame of decision making for particular actions available to the operator.
 - Understand the threshold of uncertainty acceptable to the operator for each action.
 - Determine the minimum lag time to implement each action.
 - Provide information and decision support recommendations to the operator during the acceptable decision space.
 - Support the operator questions of "Which action is best?," "Should I wait?," and "When do I revisit?" with information on how to act, when to act, and when the information has changed such that revisitation of the course of action is required.
-



Figure 12.1 Boeing's green arc indicates the horizontal position at which altitude capture occurs using a simple linear extrapolation algorithm.

Some decision support systems are simple, consisting of a succinct algorithm. For these systems, reliability of the output is straightforward and easily confirmed, so long as the algorithm is implemented correctly. Even more important, the human can internalize a highly comprehensive mental model of the system. The benefit to simple decision logic is that the user is able to internalize the logic fully, leading to accurate predictions and clarity on the limitations of the system capabilities. For example, on the Boeing 767 horizontal situation indicator, a “green arc” provides a simple indication of the horizontal position at which the aircraft is expected to capture a selected altitude using linear extrapolation (Figure 12.1). The logic does not take into account the intent of the pilot, which may be programmed into the flight management system and may deviate from the extrapolated position. This limitation of the green arc is quickly internalized and compensated by the pilot. During one simulated approach, the pilot can try out the green arc function by changing aircraft pitch and viewing how the arc changes location with the input. The hundreds of responses on a single approach can be sufficient for a pilot to understand the function of the green arc and its constraints through feedback.

However, most decision support systems in dynamic environments requiring significant amounts of expertise are incredibly complex. Determining reliability of these systems can, in itself, be difficult because of the nature of the system and potential unexpected emergent behaviors. Unfortunately, due to human limitations in memory and cognitive processing discussed in the first section, not even the expert users can have a complete understanding of the intricacies of the system. In these systems, a good mental model is required to provide functional detail to the level required. What

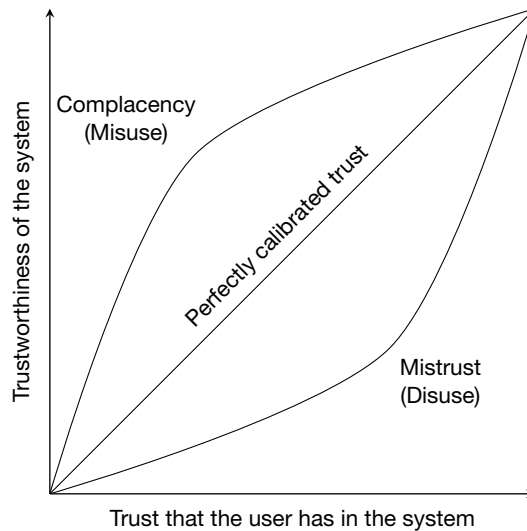


Figure 12.2 Miscalibrations of trust.

level of detail this mental model requires is often an iterative exercise among the users, trainers, and designers of the decision support system. Too little training on a system may produce a low-detail mental model that does not adequately allow the user to predict its behavior, leading to mistrust in the system. However, lengthy and extensive training on the system may lose the user's attention or exceed the user's capability of comprehension during the training process, and the primary function of the decision support system may be lost in a sea of less important detail.

The decision support system can exhibit a certain objective reliability and technical competence. However, as described in the definition, there is also a perceptual element to trust that may result in the possibility of miscalibrations of trust in the system that do not reflect the objective reality of the system's attributes. These miscalibrations include mistrust, complacency, abuse, and willful miscalibration [24], [25].

Miscalibrations can occur when a user has no (or few) preconceptions about the decision support system and sufficient confidence in his or her own capabilities as an operator. Let us consider a parameter of "trustworthiness" as a combination of objective technical competence and reliability of the system, as depicted in Figure 12.2. Plotting trust as a function of trustworthiness, a perfectly calibrated user trust in the system would be a diagonal line that increases the level of trust as the trustworthiness increases. As the user's true trust diverges from this diagonal line, the bias in either direction indicates a particular form of miscalibration.

If the user's trust is biased below the diagonal line, then there is a certain level of "mistrust" (or "disuse") in the system. Mistrust is defined as the "neglect or underutilization of automation" [24]. Biasing the user toward underutilization of the decision support system can occur when the system produces too many false alarms.

If the user's trust is biased above the diagonal line, then there is a tendency for the user to become "complacent" (or "misuse" the decision support). Complacency is defined as the "overreliance on automation, which can result in the failure of monitoring or decision biases" [24]. Bias, unfortunately, can occur when the decision support system has a proven track record of good performance. When a decision support system has consistently provided accurate recommendations to the user, the user can seek to decrease cognitive workload and fail to adequately weigh the recommendation provided by the system before acting on it. Complacency can also occur when the system does not provide sufficient feedback to the user to detect when the system is not providing good recommendations [26].

"Abuse" of the decision support system can take several forms. The often discussed form is when designers overautomate a function such that the role left for the human user is primarily a supervisory control or monitoring role. When the automation is successful, a boring role is left for the users, who fall prey to stress and fatigue caused by maintaining high levels of vigilance [27]. Another form of automation abuse occurs when users begin to use the system for unintended uses. While often innovative and useful, these unintended uses stretch the limits of technical competence of the system, whose usage in these domains was unanticipated by designers. An interesting example of unintended usage of a system is the M-PESA system, which is a mobile phone-based money transfer service intended to allow easy replenishment of pay-as-you-go mobile phone usage. However, because of the perceived instability of the banking systems in African nations such as Kenya, the M-PESA system has become a sort of ad hoc banking system in which users are more confident storing money than in the native banking institutions. Challenges soon followed addressing client information protection not required for the intended usage but expected from more formal banking transaction capabilities [26].

12.2.2 Designing for Different Levels of Certainty

In designing a decision support system, a balance is required to provide information or recommendations to support the decisions that need to be made while accounting for the capabilities and constraints of human processing. The design process will likely need to be iterative. To properly calibrate user trust in the decision support system, the designer would need to determine the appropriate balance of level and type of information for the domain. In this section, some considerations are provided for different levels of decision support certainty.

When the decision support system has a clear answer, it should provide:

1. *Information or recommendation in the terms of the decision.* When designing a decision support algorithm, designers often consider information in the form of the problem that needs to be solved, not necessarily in the terms that the user requires to interpret this information. Once information and/or a recommendation is generated by the decision support, care should be taken by the designer to translate this into contextually integratable terminology and form. For example, in the design of aviation weather decision support, care can be taken to address the weather forecasting problem. However, it is not sufficient to provide current weather status and even forecasts of weather status to air traffic controllers. To truly make this weather information adhere to the definition of “decision support,” the weather information needs to be translated into recommendations for operational decision making. Air traffic controllers are less interested in answering the question, “What is the weather like now? And in the future?” than in answering the question, “Will I need to close this departure route because of weather? And if so, when?”
2. *Timely information or recommendation.* Likewise, when the system provides information or recommendation, it should be provided at the appropriate time. Some decisions need to be made hours before implementation, whereas others are time-critical, and the user only has seconds to respond appropriately.
3. *Reliable information or recommendation.* As discussed in the section above, reliability of the system is critical to the user’s ability to develop trust in the system. Reliability under multiple circumstances should be evaluated before system implementation.
4. *Frequent opportunities to view information or recommendation trustworthiness.* For designers to appropriately calibrate trust in a decision support system, information on the trustworthiness of a system must be regularly conveyed to the user. The designers must provide not only the reliability and technical competence of the information or recommendation in a variety of circumstances, but also the system performance feedback once the recommendation has or has not been implemented.
5. *The rationale behind the decision support recommendation.* Some decision support systems help decision making on longer timescales, allowing an opportunity to provide additional information about the decision support information or recommendation. Although there is a balance between information overload and lack of information, some studies have shown that providing information on the rationale behind a decision support recommendation allows the user to better understand the decision support and improves trust in the system.

When designing a decision support system that utilizes uncertain information and the information has a confidence associated with it, one must consider additional factors. It is helpful to provide an indication of the confidence of the information presented, as well as additional information that can help the user make sense of the information that

the decision support tool finds uncertain. Let us consider one approach to uncertain information with the example of the Route Availability Planning Tool (RAPT), an air traffic management decision support tool for aviation weather.

RAPT, shown in Figure 12.3, is a tool that aids air traffic managers in determining whether airport departure routes are blocked by weather. The system also helps them identify alternative departure routes that are not blocked. RAPT provides decision support guidance to the air traffic managers by assigning a status color—“red” (blocked), “yellow” (impacted), “dark green” (insignificant weather encountered), or “green” (clear)—to each route for departure times in five-minute intervals up to 30 minutes into the future. The status is determined by combining deterministic weather forecasts from the Corridor Integrated Weather System (CIWS) with a route-blockage algorithm that incorporates a model for departure airspace usage. The route-blockage model calculates the severity of the weather impact on the first 45 minutes of flight time of the departure route. The sources of uncertainty in this tool stem not only from the predictability of the weather but also from the ability to determine whether pilots will fly through the different types of weather present. When considering the level of weather blockage to assign to each color recommendation, issues of trust in the algorithms arise. During the RAPT prototyping effort (which has been in iteration since 2003), adjustments to the algorithm have been made to better adhere to the air traffic managers’ model of the type of guidance that RAPT should be providing. Adjustments have also been made to provide guidance that has a quantitative effect on the ability of the tool to improve departure throughput in convective weather (thunderstorm) situations [28].

For the New York prototype of RAPT, the decision was made to show “green” and “red” status when the information was fairly certain and to flag departure route status as “yellow” when there was uncertainty present. This decision resulted in a relatively large number of “yellow” recommendations in convective weather situations. The designers combated this issue by training users to seek the situation when the recommendation was “yellow.” Information that the air traffic managers seek in “yellow” situations includes more information on volatility, extent, severity, geographical distribution, and location of weather impacts [29]. An example of this information is in the pop-up window shown in the figure, which displays information about the past blockage color statuses as well as the storm height on that departure route. The height of the storm is important because thunderstorms have a natural life-death cycle, of which the “echo tops” height is one indicator. Besides this information, the air traffic managers can consult the CIWS weather map above the RAPT timeline, which provides an indication of the geographical location of the storms as well as the storm “type” (e.g., a predictable front or less predictable “popcorn” convection). Using the additional information beyond the core color status recommendations provided by RAPT, air traffic managers can utilize their own judgment to determine whether and when to open or close the departure routes to make the most of the available capacity.

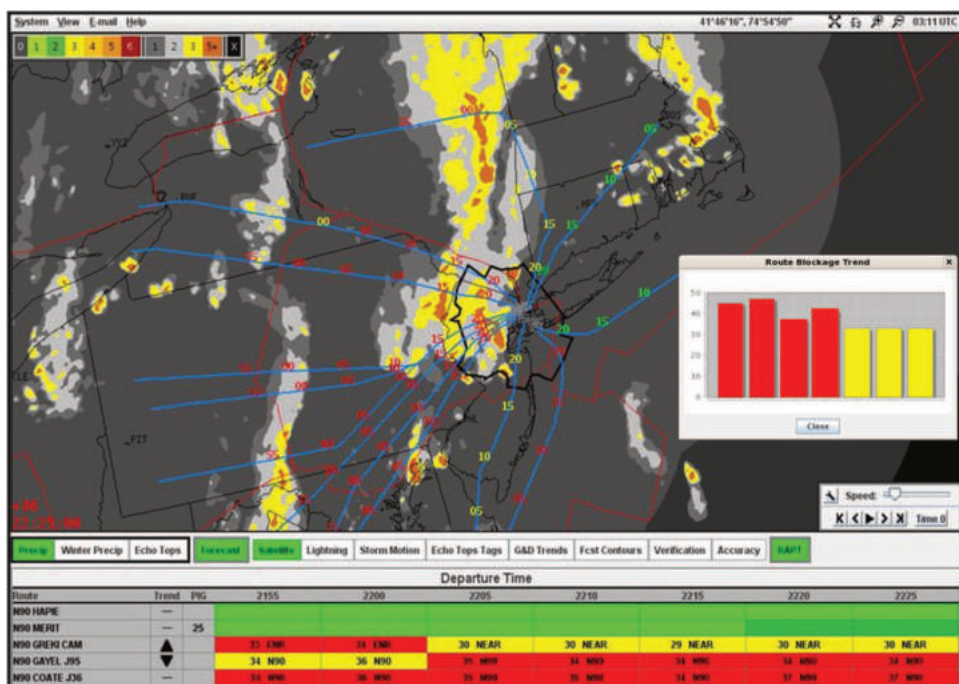


Figure 12.3 Route Availability Planning Tool (RAPT) used in air traffic management during convective weather.

Issues can arise when the decision support system provides misleading information or poor recommendations. Between interpreting uncertain information and the unanticipated complexities of real-life situations, no decision support system can be correct all the time. At this point, it is critical that the other member of the system team—the human user—maintains the functioning of the overall system. Throughout the design process, it is imperative for decision support system designers to consider the role of the human in the system. To ensure that the human is engaged and capable to override or supplement the decision support system when the decision support goes awry, the role of the human user must be an active one in the good times as well as the bad. The primary means for designers to have an input on the human user's interaction with the decision support and the environment as a whole is through the training provided on the decision support system. Training is usually provided once at the installation of the decision support system or as the user becomes qualified for the position. Occasionally, opportunities for recurrent training can be used to refresh the users' memories about the subtleties of the decision support tool.

Training issues that should be addressed include:

- Specifying the role of the decision support system in the decision-making process of the user, outlining scenario-based examples using domain-specific data and terminology.
- Specifying the constraints within which the system is trustworthy and should be relied on.
- Specifying the constraints within which the system is not trustworthy.
- Providing indications of what the human user should do in anticipation of cases in which the system is not trustworthy.

The training should clearly outline the division of responsibility for the functions that the decision support system and its user provide to the system. These are best communicated during real-time field usage, when new tools and methods can be best integrated into naturalistic decision-making processes. Alternatively, unusual situations may be best communicated in simulated settings. At a minimum, historical examples using real data and terminology should be used in a tabletop scenario to communicate the human and automation roles.

Training is also necessary to communicate the capabilities of the decision support system. Capabilities not only include “buttonology” of the decision support tool but also how the information or recommendation from the decision support system can affect operational decision making. Again, this information should be communicated through scenarios heavily based in the operational context to improve transfer of the training to the operational decision making. Scenarios aid the user in quickly developing an accurate mental model of the key aspects of the system from which to predict its behavior, thus quickly building trust in the system.

Likewise, training should also cover the limitations of the decision support system capabilities. These capabilities should be outlined to the users, and indications of the implications of these limitations for operational decision making should be provided. In the RAPT example from earlier, one of the limitations of weather forecasting is that the forecasts are inherently less reliable for “popcorn” convection than for large, predictable fronts. In the training, air traffic managers should be encouraged to use robust traffic management planning strategies (e.g., assigning routes that are viable, but possibly not optimal, in different weather situations) to combat the unpredictability of the popcorn convection type weather. If the user knows that it is popcorn convection, then it is prudent to reduce the demand traveling through the affected area below the maximum fair-weather capacity to account for possible pop-up reductions in capacity.

12.2.3 Supporting Decisions over Long Timescales

In some decisions that require support, the decision maker has hours or even days to gather information and then act. Uncertainty of the information provided can change dramatically over the decision space. A construct for understanding the relationship between information uncertainty and decision making over long timescales is presented in this section.

Which action or strategy is best given the information we have? Should I make a decision now, or would it be best to wait? When should I revisit past decisions and update my strategy? These three questions are critical to many decision makers with long-timescale problems. To address these three critical questions in decision making, procedures, and decision support, there are two key aspects to consider: forecast information quality and progressive decision making. The forecast information quality defines the realistic limits of planning and the likelihood that the world will evolve as the decision maker expects. This forecast should be expressed using an uncertainty metric that informs practical operational decision making. Because many complex system entities have a stochastic element, information changes with some regular frequency, thus requiring users to progressively revisit decisions made based on a forecast.

Choosing the variable to forecast is an important element to the problem. The variable should be directly correlated to a decision that is available to the user. To use forecast information effectively, the decision maker needs to understand the behavior of the forecast and its quality over the forecast horizon. In particular, the decision maker should have information about the magnitude and sign of the forecast error, bias, and volatility. Figure 12.4 shows a notional forecast characteristic curve and the interaction among forecast uncertainty, forecast horizon, acceptable risk, and implementation horizon for a strategy to address forecast impacts at time t_{impact} .

The time at which the impact is forecasted is t_{impact} . The latest time at which an action or strategy can be implemented for effect at t_{impact} is $t_{\text{implement}}$. For example,

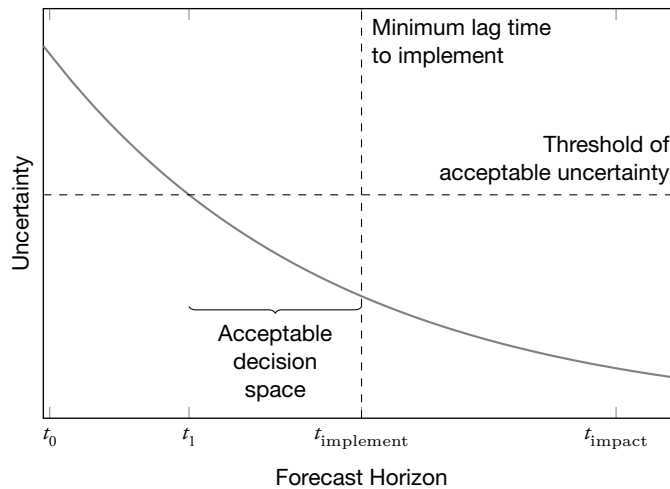


Figure 12.4 Notional uncertainty change over the forecast horizon.

occasional disease outbreaks within the population can be declared by public health departments at a particular time (t_{impact}). These events are monitored because public health can take action to prevent or reduce the impact of these events by distributing vaccines or issuing public health alerts to local hospitals. Some actions, such as a public health alert email, can have an effect within hours, so its $t_{\text{implement}}$ is close to t_{impact} . However, some actions, such as vaccine creation and distribution, take several months to have an effect, so its $t_{\text{implement}}$ is farther away from t_{impact} . The earliest time at which a decision maker would feel comfortable acting on the information is termed t_1 . This time can be highly variable from person to person and from decision to decision for a single person. For example, a well-seasoned public health official with 30 years of experience may feel comfortable with a higher threshold of acceptable uncertainty than a newly graduated epidemiologist. A public health official may also have a higher threshold for acceptable uncertainty for a decision to issue a public health alert than he or she would have for a decision to recommend closure of a school.

A decision maker in complex systems often has multiple potential strategies available for addressing an event. In trying to determine the best strategy (“Which is best?”), the first aspect to consider is the error bound (with respect to truth) across the forecast horizon. If the forecast error would not affect whether the decision maker chose choice A or choice B (i.e., the uncertainty is similar across the forecast horizon), then the forecast is good enough to make an assessment regardless of other characteristics of its behavior. However, if there is significant risk that forecast error results in the wrong decision choice (i.e., uncertainty is greater for one decision at some time on the forecast horizon),

then further characterization of the forecast behavior is required. The assessment of risk is dependent on the cost of recovery from a poor decision.

If the likelihood of a good decision is unacceptable due to the magnitude of forecast error, then the next consideration is the forecast improvement over time (“Should I wait?”). In Figure 12.4, at time t_0 , the uncertainty of the forecast is too high to provide a good indication of which decision to make. However, the slope of the uncertainty curve suggests that forecast uncertainty is decreasing rapidly. Because the latest time to implement $t_{\text{implement}}$ is still fairly far off, it is advisable to postpone the decision. By time t_1 , the uncertainty has fallen to an acceptable level, and the decision maker has the forecast information and accuracy needed to make the decision. The decision maker could continue to wait a little longer, just to be “extra certain,” so long as time $t_{\text{implement}}$ has not been reached.

Note that the threshold of acceptable uncertainty could change in an operation depending on the context of the situation. For instance, a medical report of a single student experiencing neck pain might not normally cause a university doctor to alert public health. However, if the doctor is aware of a meningitis outbreak on campus, then his threshold of uncertainty might be lowered, and he might order some tests for the illness and give a “heads up” to public health. Similarly, depending on the decision, the minimum lag time to implement will vary for different actions and strategies or for different scopes of an action (e.g., a public health alert within a city versus the entire nation). Finally, when there is an extended time over which an acceptable decision may be made, the slope of the uncertainty can provide the benefit of waiting—if there is any. For areas of the curve in which uncertainty drops quickly, there can be great benefit in waiting to ensure that the decision is made with a forecast with a better chance of being correct. If the decision space lies over an area of the curve in which the slope is nearly flat, then there is no benefit (and potentially significant costs) to waiting to make this decision.

Another consideration in addressing all three questions (“Which strategy? Do I wait? When do I revisit?”) is the volatility of the forecast. Forecasts with low volatility give the user confidence that the next forecast will be similar to the last forecast, thus providing stability of information for decision-making purposes. If forecasts are volatile as a result of the underlying unpredictability of the processes being forecast (e.g., highly dynamic and rapidly evolving convective weather), then the decision maker needs to know that the processes are inherently unpredictable and short planning horizons and frequent adjustments are necessary. However, routine or excessive forecast volatility (e.g., as a result of poorly conceived or overly precise forecast models) erodes user confidence and provides little value to decision makers.

In certain circumstances, critical variables in an environment may be impossible to forecast with acceptable accuracy. The initial decision may not always be the best

decision as time passes and the situation evolves. For this reason, progressive decision making is required.

Progressive decision making is defined as the periodic revisitation of information and adaptation of strategic decisions to updated information as it becomes available. This type of decision making involves (1) making robust strategic decisions that anticipate the need for downstream adaptation by ensuring multiple tactical options are available later in the operational time horizon, and (2) revisiting the decisions made as information updates. Procedures and decision support for progressive decision making must account for future strategic and tactical adjustment (i.e., the decisions available at different points over the forecast horizon) and the ability to recognize operationally significant changes in the forecasts from time step to time step. A similar concept termed “integrated planning” in the military decision-making literature emphasizes the importance of revisiting a “concept” with “design” to mitigate potential uncertainty in the early stages of military planning [30]. Davison Reynolds et al. discuss how a progressive decision-making framework was applied to air traffic management [31].

12.3 A Systems View of Implementation

Implementing an effective decision support system is an iterative process (Figure 12.5). Ideally, the decision support design should begin with field observations and analyses of operational data to understand the operation and the role of users. The designer’s understanding of the operation, users, and constraints forms the operational model. From this operational model, the need for decision support may emerge, and requirements are defined. The decision support design and development then occurs, resulting in a procedure, a human-machine interface (HMI), and training. Implementation of the decision support into the environment then impacts the operations, and the effect of the decision support can then be measured through operational data analyses and field observations. This section outlines each phase of the design process. Design considerations for effective systems implementation are provided in Table 12.3.

12.3.1 Interface, Training, and Procedures

When one is designing a decision support tool, there is often a core idea for the system that has been translated into, at least, a functional decision support requirement. This requirement is based on an operational model of the entirety of the system, including the humans, the operations, the procedures, and the constraints. Once a need for decision support has been determined, the requirement is then translated into a decision support system design. Taking a broad interpretation of a decision support system, the decision support provided could include a technical algorithm supported by an HMI visualization to the human, a new procedure, new or modified training, or some combination of

Table 12.3 Design considerations and suggestions: A systems view of implementation.

Human-Machine Interface

- Display information or recommendations in terms of the decision and action to be made.
 - Use the display to help convey an accurate (if simplified) mental model of the decision support logic to foster operator trust and appropriate interaction with the system.
 - Provide information in a timely fashion to aid the decisions being made.
 - Ensure that the text fonts and colors are readable to the operator [32].
 - Avoid clutter and information not critical to the decision being made by the operator.
 - Test the understandability and usability of the HMI within the operational environment to account for unforeseen circumstances (e.g., noise levels, bright sunshine, dark of night).
-

Training

- Specify the role of the decision support system in the decision-making process of the operator, outlining scenario-based examples using domain-specific data and terminology.
 - If possible, be specific about the benefit the decision support designers wish to achieve and how the operators can help to reach that goal.
 - Convey the capabilities and limitations of the system specifically and tangibly for the operator.
 - Provide indications of what the operator should do when the system is operating outside its limits.
-

Procedures

- Consider how the current procedures simplify the cognitive load on the operator.
 - Design procedures to maintain or optimize the operator's cognitive load if possible.
 - To ensure that the operator is using the system as it was designed, design procedures to map the information presented by the HMI to the course of action desired. By clarifying procedures in the design and training, the human will respond more predictably.
-

Measurement of System Effectiveness

- Identify the key metric by which success of the decision support system can be defined.
 - Develop a hypothesis about how the decision support will affect this metric.
 - Use operational data after beta system implementation to assess whether the benefits were achieved.
 - Use observation data to identify other reasons why the system is not achieving the desired benefit.
-

Organizational Influences

- Identify the goals of the operational domain.
 - Identify the organizational incentives of the operators: what is rewarded and what is reprimanded.
 - Consider how the incentives in the domain affect the use of the decision support system and the acceptability of and trust in the recommendations provided.
-

these three items. The effective implementation of the means of decision support (tool, procedure, training) should then affect the key variables of the operational system, resulting in a measurable benefit to operations.

12.3.2 Measuring Decision Support Effectiveness

The key to the design process is to ensure that the decision support system is actually achieving operational benefits and having an impact. Thus, it is the designer's responsibility to go back to the field and assess the impact that the decision support tool is having. This measurement process must be both quantitative and qualitative in nature. On the quantitative side, operational data that result from the improved decision making should be gathered. For example, if the decision support tool being designed is an airborne traffic collision avoidance system, then operational data on airborne collisions, near-misses, and separation in general should be gathered. The operational model that fed the design of the system included a hypothesis: "If pilots had a system to aid in knowing when a collision was imminent and gave a suggestion about how to avoid it, then there should be fewer accidents, near-misses, and separation violations in the air." This hypothesis forms the input in the "Expected operational problems" in Figure 12.5. Implementing the decision support system and subsequently measuring the operational data allow the designer to test this hypothesis. The operational data should eventually reveal whether this hypothesis made by the designers is correct (i.e., "Are the problems identified the real problems?" in Figure 12.5). If the operational data reveal that the decision support implemented does, in fact, reduce accidents, near-misses, and separation violations to the degree desired, then the designers can consider themselves successful. However, it is rare that the initial implementation of decision support achieves the benefits it aimed to achieve.

One reason that the decision support was not successful is that the designer's hypothesis was wrong and the problem identified was not the actual problem. Gathering operational data can help correct this issue. For example, in the case of the RAPT decision support tool, designers initially thought that the operational problem was helping the traffic managers know when to close departure routes during a thunderstorm. Extensive analysis and thought went into considering training for traffic managers on how to interpret patterns of the RAPT timeline and what that meant for departure route closures. Unfortunately, even after in-depth training sessions and operational evaluation, the tool led to few improvements in delay statistics in poor weather conditions.

The designers returned to the analysis and reconsidered the delay statistic measurement. While doing this, they discovered that a majority of the unnecessary delay was not caused by traffic managers closing the routes at the wrong time but opening the routes too late after a storm had already passed through! The designers were slightly appalled by this finding. The traffic managers were missing the easy decision, which

was opening the departure routes when the routes were clearly GREEN on the RAPT display. However, once the operational data revealed what the real problem was, the designers could be surgical about what decision support was provided. They adjusted the RAPT display to include a “post-impact GREEN” (PIG) timer that began counting as soon as a route that had been RED turned all-GREEN. This HMI adjustment allowed the traffic managers to see if or when they were missing an opportunity to open the route.

The designers also accompanied this HMI adjustment with a training modification that included providing the traffic managers with the statistics that they were not opening the routes as quickly as they could, providing motivation to use the PIG timer. The next year, the benefits of using the RAPT tool were measured double over the previous year, achieving the desired operational impact.

Another reason that the decision support was not successful could be that the designer’s hypothesis was correct but the decision support did not achieve the goal. Better understanding this requires the use of qualitative measurement of the operational environment. Substantial information on the usefulness of the decision support can also be gathered from the users during use (or lack thereof) of the decision support. In the operational environment, users are able to point out directly why the decision support helps or does not. Placement of the decision support, font size, and lighting issues all become starkly clear in the operational environment. If these basic needs are satisfied, then users may point to subtler issues, such as how the information provided is not exactly what they need to make the decision or that while the decision support is good, this user is not the decision maker at all. While performing field studies in the evaluation of RAPT, the designers originally provided RAPT to the traffic managers in the Traffic Management Unit. During the evaluation, it was discovered that the traffic managers could use the RAPT tool and decide when a route should be opened, but the individual sectors with the air traffic controllers who were responsible for separating the aircraft did not have access to the RAPT tool and therefore would refuse the opening of the route. The RAPT monitors were inserted into the individual sector areas, and the influence of RAPT began impacting decisions.

Each of these examples describes the iterative measurement feedback loops in Figure 12.5. The quantitative loop on top is the contribution that the operational data analysis makes. This data analysis is usually in the form of a benefits analysis, which allows the designer to monetize the benefit that the decision support has on the operation. Often the most difficult part of the measurement process is identifying the metric on which to base operationally measurable benefit. The second loop is the qualitative loop. Qualitative feedback can be gathered through surveys, formal interviews, or preferably in situ field observations of the users in the operation. Feedback is useful when users provide indications of why the decision support is not working as is, as discussed above. Qualitative feedback also provides a chance for the users to suggest to the designers

decision support ideas of their own, which can be incredibly insightful when it originates from a “superuser” who has a broad perspective on the operation as a system. As each evaluation of the decision support systems gathers more data, both quantitative and qualitative, there is an opportunity to better understand the system into which the decision support system is to be implemented. The operational model is updated with this new information, which is critical to improving the design in future iterations.

12.3.3 Organization Influences on System Effectiveness

The feedback loop described above works optimally in a system that is driven primarily by benefits-centered improvements. The FAA is an example of such a system; at least on paper, research and acquisition programs are promoted that will provide an established efficiency or safety benefit to the airspace system.

However, the elements of the system are a product of the organization in which they are members. Users can be driven by motivations other than optimizing the overall system function. These other motivations may be personal motivations, but often they are organizational incentives. For example, the typical traffic manager who works for the FAA is motivated to adjust the demand on the routes through his or her facility to optimize the capacity available at any given time. If the demand is too high, then the sector air traffic controllers will be overwhelmed and stop the flow altogether (and perhaps get angry at the traffic manager in the process). If the demand is reduced too low, well below capacity, then the facility could receive pushback the next day from the Air Traffic Control System Command Center or other facilities. The incentive for the traffic manager is to effectively balance demand and capacity within his or her own facility. This incentive sense superficially; however, one must consider that the aviation system is not composed of “islands” of air traffic but rather is a system in which each facility’s demand and capacity affects other facilities. The incentives of traffic managers are based solely on the demand and capacity of their own facilities, not the effect of decisions on other facilities. Thus, when seeking to optimize air traffic throughout the airspace, the consideration of incentives becomes an important constraint in the effective implementation of a solution. Considering the organizational impacts on user behavior in an operation can translate into a highly developed operational model for the designers, and this effort can result in better decision support.

12.4 Summary

In this chapter, several approaches were discussed to aid the designer in considering the human capabilities and limitations in decision support. A systems approach was provided as a framework to scaffold an effective iterative design method, which incorporates training and system measurement in addition to the design work. The concept of trust

was addressed as a key component to both algorithm design and acceptability. Finally, a point was made that the importance of understanding the decision-making problem in a naturalistic setting, with all of its constraints and uncertainties, can completely change the design approach and considerations from problem to problem.

In summary, the best advice is to make a concerted effort to truly understand the problem, context, and humans involved before designing the decision support. This understanding will make the difference between a system that provides true decision support and one that is either an impediment to the operation or not used at all.

References

1. B. Crandall, G.A. Klein, and R.R. Hoffman, *Working Minds: A Practitioner's Guide to Cognitive Task Analysis*. Cambridge, MA: MIT Press, 2006.
2. C.D. Wickens, J.G. Hollands, and R. Parasuraman, *Engineering Psychology and Human Performance*, 4th ed. Boston: HarperCollins, 2013.
3. C.D. Wickens, S.E. Gordon, and Y. Liu, *An Introduction to Human Factors Engineering*. New York: Longman, 1998.
4. P.L. Wachtel, "Conceptions of Broad and Narrow Attention," *Psychological Bulletin*, vol. 68, no. 6, pp. 417–429, 1967. DOI: 10.1037/h0025186.
5. D. Navon and D. Gopher, "On the Economy of the Human-Processing Systems," *Psychological Review*, vol. 86, no. 3, pp. 214–255, 1979. DOI: 10.1037/0033-295X.86.3.214.
6. C.D. Wickens, "The Structure of Attentional Resources," in *Attention and Performance VIII*, R. Nickerson, ed., New York: Erlbaum, 1980.
7. ———, "Processing Resources in Attention," in *Varieties of Attention*, R. Parasuraman and R. Davies, eds., Orlando, FL: Academic Press, 1984.
8. G.A. Miller, "The Magical Number Seven Plus or Minus Two: Some Limits on Our Capacity for Processing Information," *Psychological Review*, vol. 63, no. 2, pp. 81–97, 1956. DOI: 10.1037/h0043158.
9. W.G. Chase and A. Ericsson, "Skilled Memory," in *Cognitive Skills and Their Acquisition*, S.A. Anderson, ed., New York: Erlbaum, 1981.
10. W.G. Chase and H.A. Simon, "The Mind's Eye in Chess," in *Visual Information Processing*, W.G. Chase, ed., New York: Academic Press, 1973.
11. J.R. Anderson, *Cognitive Psychology*, 4th ed. London: W. H. Freeman, 1995.
12. D.E. Broadbent, "Language and Ergonomics," *Applied Ergonomics*, vol. 8, no. 1, pp. 15–18, 1977. DOI: 10.1016/0003-6870(77)90111-9.
13. P.H. Lindsay and D.A. Norman, *Human Information Processing*. New York: Academic Press, 1972.

14. I.L. Janis and L. Mann, *Decision Making: A Psychological Analysis of Conflict, Choice and Commitment*. New York: Free Press, 1977.
15. L.B. Beach and R. Lipshitz, “Why Classical Decision Theory Is an Inappropriate Standard for Evaluating and Aiding Most Human Decision Making,” in *Decision Making in Action: Models and Methods*, G. Klein, J. Orasanu, R. Calderwood, and C. Zsombok, eds., Norwood, NJ: Ablex, 1993.
16. H. Simon, *Models of Man: Social and Rational*. New York: Wiley, 1957.
17. D. Kahneman, P. Slovic, and A. Tversky, *Judgment Under Uncertainty: Heuristics and Biases*. New York: Cambridge University Press, 1982.
18. J. Orasanu and T. Connolly, “The Reinvention of Decision Making,” in *Decision Making in Action: Models and Methods*, G. Klein, J. Orasanu, R. Calderwood, and C.E. Zsombok, eds., Norwood, NJ: Ablex, 1993.
19. G. Klein, *Sources of Power*. Cambridge, MA: MIT Press, 1999.
20. N. Moray, “A Lattice Theory Approach to the Structure of Mental Models,” *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, vol. 327, no. 1241, pp. 577–583, 1990.
21. D. Gentner and A.L. Stevens, eds., *Mental Models*. Hillsdale, NJ: Erlbaum, 1983.
22. H.J. Davison and R.J. Hansman, “Use of Structure as a Basis for Abstraction in ATC,” in *International Symposium on Aviation Psychology*, 2003.
23. J.M. Histon, “The Impact of Structure on Cognitive Complexity in Air Traffic Control,” M.S. thesis, Massachusetts Institute of Technology, 2002.
24. R. Parasuraman and V. Riley, “Humans and Automation: Use, Misuse, Disuse, Abuse,” *Human Factors*, vol. 39, no. 2, pp. 230–253, 1997. doi: 10.1518/001872097778543886.
25. B.M. Muir, “Trust Between Humans and Machines, and the Design of Decision Aids,” *International Journal of Man-Machine Studies*, vol. 27, no. 5-6, pp. 527–539, 1987. doi: 10.1016/S0020-7373(87)80013-5.
26. R. Parasuraman and D.H. Manzey, “Complacency and Bias in Human Use of Automation: An Attentional Integration,” *Human Factors*, vol. 52, no. 3, pp. 381–410, 2010. doi: 10.1177/0018720810376055.
27. J.S. Warm, R. Parasuraman, and G. Matthews, “Vigilance Requires Hard Mental Work and Is Stressful,” *Human Factors*, vol. 50, no. 3, pp. 433–441, 2008. doi: 10.1518/001872008X312152.
28. H.J. Davison Reynolds, R. DeLaura, and M. Robinson, “Field & (Data) Stream: A Method for Functional Evolution of the Air Traffic Management Route Availability Planning Tool (RAPT),” in *Annual Human Factors and Ergonomics Society Conference*, 2010.

29. N. Underhill and R. DeLaura, “Estimation of New York Departure Fix Capacities in Fair and Convective Weather,” in *Aviation, Range, and Aerospace Meteorology Special Symposium on Weather-Air Traffic Management Integration*, 2012.
30. W. Grigsby, S. Gorman, J. Marr, J. McLamb, M. Stewart, and P. Schifferle, “Integrated Planning: The Operations Process, Design and the Military Decision Making Process,” *Military Review*, no. 1, pp. 28–35, 2011.
31. H. Davison Reynolds, R. DeLaura, J. Venuti, and M. Wolfson, “Uncertainty and Decision Making in Air Traffic Management,” in *AIAA Aviation Technology, Integration, and Operations Conference (ATIO)*, 2013.
32. Department of Defense, “Design Criteria Standard,” Standard MIL-STD-1472F, 2012.

Index

- Abstraction, 122
- ACAS X, *see* Airborne Collision Avoidance System X
- Acoustic model, 217
- Additive decomposition, 85
- Affine transformation, 56
- Agent, 2
- Air traffic control, 2, 229, 248, 270, 278, 281, 289
- Airborne Collision Avoidance System X, 233, 241, 253
- Aircraft
 - general aviation, 233
 - unmanned, 3, 232, 233
- Airspace model, 229
- Alpha vector, 132
- Approximate dynamic programming, 89, 105, 128, 167
- Attention, 267
- Attribute-based search, 175
- Average reward, 75
- Axioms of probability, 12
- Backpropagation, 122
- Bandit problem, 107, 123
- Bandwidth, 46
- Basis function, 92, 120
- Bayes' rule, 13, 29, 45, 47, 147, 241, 252
- Bayes-adaptive Markov decision process, 114, 124
- Bayesian network, 17, 19, 21, 31, 54
 - dynamic, 25, 229
 - hybrid, 21
 - inference, 25
 - parameter learning, 40, 246
 - representation, 17
 - structure learning, 46, 246
- Bayesian policy gradient, 124
- Bayesian score, 47, 52, 54
- BDe, 51, 54
- BDeu, 51, 54
- Behavioral cloning, 4
- Belief, 12
- Belief propagation, 32, 54
- Belief state, 109, 126, 239, 240
- Belief state update, 126
- Belief-state Markov decision process, 126
- Bernoulli process, 236
- Best response, 68
- Biometric search, 175
- Branch and bound, 96, 143
- Cepstral feature, 214
- Certainty effect, 62
- Chain, 20
- Chain rule, 18, 27, 31
- Chance node, 62
- Chatter, 244
- Classification, 4, 26
- Classifier, 186
- Collision avoidance, 57, 59, 229
- Collision risk, 42, 246
- Communication, 3, 11, 172, 255
- Completeness, 56
- Completion policy, 163
- Compression, 239
- Conditional edge, 62
- Conditional independence, 26, 75

- Conditional plan, 133
- Conditional probability, 12
- Conditionally, 19
- Constraints, 232
- Continuity, 56
- Cooperation, 255
- Coordination graph, 158
- Corrective, 234
- Cross entropy, 101, 105, 248
- Crossing, 237
- Crossover, 104
- Crying baby, 125, 129, 133–135
- Cumulative distribution function, 13

- D-separation, 20
- Dec-MDP, *see* Decentralized Markov decision process
- Dec-POMDP, *see* Decentralized partially observable Markov decision process
- Dec-POMDP-Com, *see* Decentralized partially observable Markov decision process with communication
- Decentralized Markov decision process, 156
- Decentralized partially observable Markov decision process, 149
- Decentralized partially observable Markov decision process with communication, 169
- Decision diagrams, 85
- Decision network, 62, 71
 - dynamic, 113, 125, 126
- Decision node, 62
- Decision problem
 - sequential, 73, 104
 - single-shot, 63
- Decision support system, 3
- Decision theory, 270
- Decision tree, 85
- Design parameter, 249, 252
- Deterministic, 231
- Diagnosis, 11

- Diagnostic test, 65
- Direct policy search, 99
- Directed acyclic graph pattern, *see* Partially directed graph
- Directed graph search, 48
- Discount factor, 74
- Discretization, 16, 87, 90, 239
- Distribution
 - beta, 43, 45, 108
 - binomial, 40
 - class-conditional, 27
 - compound Dirichlet-multinomial, 194
 - conditional linear Gaussian, 22
 - Dirichlet, 45, 47, 113, 184, 189
 - Gaussian, 13, 14, 30, 41, 130, 141, 236
 - geometric, 236
 - joint, 16
 - linear Gaussian, 21, 22, 24, 87, 128, 130
 - multivariate Gaussian, 23
 - piecewise-uniform, 16
 - Polya, 194
 - prior, 27
 - quasi-normal, 180
 - symmetric Dirichlet, 45
 - uniform, 13, 108
- Dominant strategy, 68
- Dyna, 111, 123
- Dynamic programming, 75, 105, 110, 160, 167, 170, 229, 234, 237, 259
 - real-time, 105

- Eligibility trace, 117, 124
- Elite sample, 101
- EM, *see* Expectation maximization
- Equilibrium
 - dominant strategy, 68
 - Nash, 68, 70, 72, 168
- Essential graph, *see* Partially directed graph
- Estimation, 230, 231, 240, 241, 244, 246

- kernel density, 45
- maximum likelihood, 40, 111, 123, 190
- recursive Bayesian, 128
- Evaluative feedback, 107
- Evidence variable, 26
- Expectation maximization, 216
- Explaining away, 20
- Exploitation, 107
- Exploration, 107
 - ϵ -greedy, 109
 - directed, 109
 - interval, 109
 - softmax, 109
- Extrapolation, 231
- Factor graph, 54
- Factored Markov decision process, 85
- Factorial, 43
- Fast informed bound, 136, 143, 147
- Features, 92
- Feedback, 289
- Fibonacci sequence, 75
- Filter
 - alpha-beta, 240
 - Kalman, 30, 130, 147, 240
 - nonlinear, 240
 - particle, 130, 147
 - unscented Kalman, 242
- Filtering, 29
- Finite horizon, 74
- Flight test, 253
- Fork, 20
- Forward search, 95, 142
- Framing effect, 62
- Functional edge, 63
- Fusion, 246
- Game theory, 67, 71
 - behavioral, 69, 72
- Gamma function, 43
- Gaussian mixture model, 16, 213, 215, 216, 221–223, 225
- Generalization, 118, 124
- Generalized belief state, 155, 160, 161
- Generative model, 97, 98, 100, 130, 179
- Genetic algorithm, 50, 102, 105
- Genetic local search, *see* Memetic algorithm
- Genetic programming, 104, 105
- Gittins allocation index, 110
- Global approximation, 92, 120
- Global positioning system, 233
- GMM, *see* Gaussian mixture model
- GPS, *see* Global positioning system
- Gradient ascent, *see* Local search
- Grid world, 79, 104, 151
- Heuristic, 167, 231, 270
- Heuristic Search Value Iteration, 147
- Hidden layer, 122
- Hidden Markov model, 24, 29, 75, 213–215, 217, 218, 220
- Hidden variable, 26
- Hill climbing, *see* Local search
- Histogram, 241
- History, 144
- History tree, 145
- HMI, *see* Human-machine interface
- HMM, *see* Hidden Markov model
- HSV color space, *see* Hue-saturation-value color space
- Hue-saturation-value color space, 180
- Human error, 230
- Human-machine interface, 284
- Human-systems integration, 267
- Hypergraph, 158
- I-vector, 222, 224, 225
- ICM, *see* Iterated conditional modes
- Imperfect state information, 105
- Incremental feature dependency discovery, 262, 263

- Independence, 56
- Independent, 19
- Inference, 25, 27
- Infinite horizon, 74
- Influence diagram, *see* Decision network
- Information-gathering action, 136
- Informational edge, 62
- Input layer, 122
- Interpolation, 259
 - bilinear, 90
 - linear, 90
 - multilinear, 90, 239, 240, 242
 - simplex-based, 91
- Interview, 289
- Inverted fork, *see* V-structure
- Irrationality, 61, 71
- Iterated conditional modes, 198
- Iterative policy evaluation, 76
- JESP, *see* Joint equilibrium search for policies
- Joint equilibrium search for policies, 168, 171
- Joint policy, 150
- Junction tree algorithm, 32, 54
- K2, 49, 54
- Kalman gain, 130
- Kernel function, 46, 89, 192
- Kronecker delta function, 114, 128
- Language identification, 215, 220–222
- Language model, 217, 218, 220, 221
- LAO*, 105
- Latent variable, 176, 181, 189
- Law of total probability, 12, 27, 29, 31, 47
- Learning rate, 116
- Linear dynamical system, 24, 30
- Linear programming, 135, 161
 - mixed integer, 172
- Linear quadratic Gaussian, 105
- Linear quadratic regulator, 89
- Linear regression, 92
- Local approximation, 119
- Local graph operations, 52
- Local information, 149, 154
- Local optima, 5, 50
- Local search, 49, 100, 105
- Local states, 157
- Locally fully observable, 157
- Log-factorial, 206
- Log-likelihood, 41, 199
- Logic model, 22
- Logit level- k model, 69
- Lookahead, 141
- Lookup table, 240–242, 244
- Loopy belief propagation, 40
- Lottery, 55, 56, 58, 67
- MAA*, *see* Multiagent A*
- Machine translation, 220, 226, 227
- Marginalization, 31, 196
- Markov assumption, 73
- Markov blanket, 20, 38
- Markov chain, 23
- Markov chain Monte Carlo, 38
- Markov decision process, 73, 104, 108, 149, 162, 259
 - stationary, 73
- Markov equivalence class, 51
- Markov equivalent, 51, 54
- Markov random field, 54
- Match score, 199
- Maximum expected utility principle, 57, 61, 71
- MBDP, *see* Memory-bounded dynamic programming
- MDP, *see* Markov decision process
- Medical diagnosis, 63
- Memetic algorithm, 104
- Memetic algorithms, 51
- Memory, 269

- Memory-bounded dynamic programming, 167, 171
- Mental model, 271, 274
- Mental simulations, 271
- Message passing algorithm, 54
- Mixed strategy, 67
- MMDP, *see* Multiagent Markov decision process
- Modified policy iteration, 77
- Monte Carlo tree search, 98, 105, 124, 144, 147
- Most likely explanation, 29
- Multiagent A*, 162, 170
- Multiagent belief state, *see* Generalized belief state
- Multiagent Markov decision process, 159, 256, 258
- Multiagent planning, 255
- Multimodal, 14
- Mutation, 104
- Myopic communication, 171
- Naive Bayes model, 26
- Naturalistic decision making, 270
- ND-POMDP, *see* Network distributed partially observable Markov decision process
- Near midair collision, 246
- Nearest neighbor, 90
- Nelder-Mead optimization, 195
- Network distributed partially observable Markov decision process, 158, 170
- Neuron, 120
- NEXP-complete, 155, 158, 159, 169
- NMAC, *see* near midair collision
- Nonparametric learning, 45
- Normalization constant, 28
- Normalized utility function, 57
- NP, 33
- NP-complete, 33, 69, 158
- NP-hard, 32–34, 51, 54
- Observation, 2
- Observation independent, 157
- Observation model, 126, 150
- Observe-act cycle, 1
- Online cost, 242, 243
- Operational considerations, 3, 237
- Optimization, 5
- Output layer, 122
- Overautomation, 275
- Overreliance, 275
- P, 33
- P-complete, 155
- Parallel approach, 254
- Parameter learning
 - Bayesian, 42
 - maximum likelihood, 40
- Partial policy, 163
- Partially directed graph, 52
- Partially observable Markov decision process, 114, 125, 149, 162, 229, 234, 240
- Partially Observable Monte Carlo Planning, 147
- Partially observable stochastic game, 150
- Particle deprivation, 132
- Path planning, 80
- Perceptron, 120
- Perseus, 147
- Persistent surveillance, 3, 255
- Planning, 5, 80
 - closed-loop, 80
 - open-loop, 80
- Point estimate, 240
- Policy, 75, 126, 256
 - decentralized, 149
- Policy evaluation, 76, 77
- Policy improvement, 77
- Policy iteration, 77, 166
 - structured, 85, 104

- Policy loss, 79
- Policy tree, 133, 152
- Polytree, 54
- POMDP, *see* Partially observable Markov decision process
- POSG, *see* Partially observable stochastic game
- Posterior, 30
- Prediction, 29
- Preference elicitation, 58
- Preventive, 234
- Prioritized sweeping, 112, 123
- Priority queue, 112
- Prisoner's dilemma, 67, 68
- Probability density function, 13
- Probit model, 22
- Programming, 4
- Progressive widening, 105
- Prospect theory, 71
- Pseudocount, 44, 184
- PSPACE-complete, 136, 155
- Pure strategy, 67
- Q-learning, 116, 124
 - linear approximation, 119
 - perceptron, 120
- QMDP, 136, 142, 143, 147, 240
- Quadratic reward, 87
- Quadrotor, 255, 263, 264
- Quantization, 240
- Query variable, 26
- RA, *see* Resolution advisory
- Radar, 2, 14, 21, 26, 229–232, 246, 270
- Randomized restart, 50
- RAPT, *see* Route Availability Planning Tool
- Rational agent, 57
- Rational preference, 56
- Recursive Bayesian estimation, 30
- Reinforcement learning, 5, 105, 107, 123
 - Bayesian, 113
 - Bayesian model-based, 124
 - Bayesian model-free, 124
 - model-based, 111, 124, 256
 - model-free, 115
 - multiagent, 124
- Resolution advisory, 230, 234, 244
- Reversal, 234
- Reward function, 73, 150, 237, 258
- Reward independent, 157
- Risk averse, 58
- Risk neutral, 58
- Risk ratio, 246, 248
- Risk seeking, 58
- Rollout policy, 99, 142
- Route Availability Planning Tool, 278
- Safety, 3, 229, 237, 241, 246
- Sampling, 246
 - direct, 35, 39
 - Gibbs, 38, 39
 - importance, 248
 - likelihood-weighted, 36, 39
 - Monte Carlo, 241
 - Thompson, 115, 124
- Sarsa, 117, 124
- Satellite, 11, 17
- Satisfiable, 33
- Screening, 252
- Self-organizing map, 119
- Sense, 234
- Sense and avoid, 232
- Sensor error, 3, 4, 125, 229, 240
- Sigmoid, 22
- Simulated annealing, 50, 195
- Smoothing, 29
- Soft biometrics, 176
- Sparse sampling, 97, 124
- Speaker identification, *see* Speaker recognition
- Speaker recognition, 215, 223–225
- Speech processing, 214, 215

- Speech recognition, 214–220, 224
- Standard normal cumulative distribution function, 14
- State aggregation, 85
- State space, 236, 241, 256, 258
- State transition model, 23, 73, 97, 257, 259
- Stationary, 23, 30
- Stochastic optimization, 105
- Strategy profile, 68
- Strengthening, 234
- Stress testing, 248
- Structure learning, 46
- Successive Approximations of the Reachable Space under Optimal Policies, 147
- Sum-squared error, 92
- Supervised learning, 4
- Support, 14
- Support vector machine, 221, 223, 225
- Surrogate model optimization, 252
- Surveillance system, 2, 232
- Survey, 289
- SVM, *see* Support vector machine
- System parameter, 252
- TA, *see* Traffic advisory
- Tabu search, 50
- TCAS, *see* Traffic Alert and Collision Avoidance System
- Temporal difference error, 116
- Temporal model, 23, 29
- Topic identification, 219
- Topological sort, 35
- Traffic advisory, 230, 244
- Traffic Alert and Collision Avoidance System, 2, 3, 230, 231, 243
- Training examples, 4
- Transition independent, 157
- Transition model, 150, 157
- Transitivity, 12, 56
- Traveler's dilemma, 70
- Trust, 272, 277
- Uncomputable, 136
- Undecidable, 155
- Underutilization, 275
- Unimodal, 14
- Universal comparability, 12
- Unmanned aircraft, 255
- Upper confidence bound for trees, 98
- Utility, 56
 - additively decomposed, 60
 - multiple variable, 59
- Utility elicitation, 58, 252
- Utility function, 252
- Utility node, 62
- Utility of money, 58
- Utility theory, 55, 71
- V-structure, 20, 51, 52
- Value iteration, 77, 135
 - asynchronous, 80
 - Gauss-Seidel, 80, 239
 - linear regression, 92
 - local approximation, 89, 239
 - point-based, 137, 142
 - structured, 85, 104
- Value of information, 64, 71
- Variable elimination, 31
- Video surveillance, 175
- Von Neumann-Morgenstern axioms, 56
- Weakening, 234