

Markov Decision Processes in Artificial Intelligence

Markov Decision Processes in Artificial Intelligence

MDPs, Beyond MDPs and Applications

Edited by
Olivier Sigaud
Olivier Buffet



First published 2008 in France by Hermes Science/Lavoisier in two volumes entitled: *Processus décisionnels de Markov en intelligence artificielle* © LAVOISIER 2008
First published 2010 in Great Britain and the United States by ISTE Ltd and John Wiley & Sons, Inc.

Apart from any fair dealing for the purposes of research or private study, or criticism or review, as permitted under the Copyright, Designs and Patents Act 1988, this publication may only be reproduced, stored or transmitted, in any form or by any means, with the prior permission in writing of the publishers, or in the case of reprographic reproduction in accordance with the terms and licenses issued by the CLA. Enquiries concerning reproduction outside these terms should be sent to the publishers at the undermentioned address:

ISTE Ltd
27-37 St George's Road
London SW19 4EU
UK
www.iste.co.uk

John Wiley & Sons, Inc.
111 River Street
Hoboken, NJ 07030
USA
www.wiley.com

© ISTE Ltd 2010

The rights of Olivier Sigaud and Olivier Buffet to be identified as the authors of this work have been asserted by them in accordance with the Copyright, Designs and Patents Act 1988.

Library of Congress Cataloging-in-Publication Data

Markov decision processes in artificial intelligence : MDPs, beyond MDPs and applications / edited by Olivier Sigaud, Olivier Buffet.

p. cm.

Includes bibliographical references and index.

ISBN 978-1-84821-167-4

1. Artificial intelligence--Mathematics. 2. Artificial intelligence--Statistical methods. 3. Markov processes. 4. Statistical decision. I. Sigaud, Olivier. II. Buffet, Olivier.

Q335.M374 2010

006.301'509233--dc22

2009048651

British Library Cataloguing-in-Publication Data
A CIP record for this book is available from the British Library
ISBN 978-1-84821-167-4

Printed and bound in Great Britain by CPI Antony Rowe, Chippenham and Eastbourne



Cert no. SGS-COC-2953
www.fsc.org
© 1996 Forest Stewardship Council

Table of Contents

Preface	xvii
List of Authors	xix
PART 1. MDPs: MODELS AND METHODS	1
Chapter 1. Markov Decision Processes	3
Frédéric GARCIA and Emmanuel RACHELSON	
1.1. Introduction	3
1.2. Markov decision problems	4
1.2.1. Markov decision processes	4
1.2.2. Action policies	7
1.2.3. Performance criterion	8
1.3. Value functions	9
1.3.1. The finite criterion	10
1.3.2. The γ -discounted criterion	10
1.3.3. The total reward criterion	11
1.3.4. The average reward criterion	11
1.4. Markov policies	12
1.4.1. Equivalence of history-dependent and Markov policies	12
1.4.2. Markov policies and valued Markov chains	13
1.5. Characterization of optimal policies	14
1.5.1. The finite criterion	14
1.5.1.1. Optimality equations	14
1.5.1.2. Evaluation of a deterministic Markov policy	15
1.5.2. The discounted criterion	16
1.5.2.1. Evaluation of a stationary Markov policy	16
1.5.2.2. Optimality equations	17
1.5.3. The total reward criterion	22

1.5.4. The average reward criterion	24
1.5.4.1. Evaluation of a stationary Markov policy	24
1.5.4.2. Optimality equations	27
1.6. Optimization algorithms for MDPs	28
1.6.1. The finite criterion	28
1.6.2. The discounted criterion	28
1.6.2.1. Linear programming	28
1.6.2.2. The value iteration algorithm	29
1.6.2.3. The policy iteration algorithm	30
1.6.3. The total reward criterion	34
1.6.3.1. Positive MDPs	34
1.6.3.2. Negative MDPs	34
1.6.4. The average criterion	35
1.6.4.1. Relative value iteration algorithm	35
1.6.4.2. Modified policy iteration algorithm	36
1.7. Conclusion and outlook	37
1.8. Bibliography	37
Chapter 2. Reinforcement Learning	39
Olivier SIGAUD and Frédérick GARCIA	
2.1. Introduction	39
2.1.1. Historical overview	39
2.2. Reinforcement learning: a global view	40
2.2.1. Reinforcement learning as approximate dynamic programming	41
2.2.2. Temporal, non-supervised and trial-and-error based learning	42
2.2.3. Exploration versus exploitation	42
2.2.4. General preliminaries on estimation methods	44
2.3. Monte Carlo methods	45
2.4. From Monte Carlo to temporal difference methods	45
2.5. Temporal difference methods	46
2.5.1. The TD(0) algorithm	47
2.5.2. The SARSA algorithm	48
2.5.3. The Q-learning algorithm	49
2.5.4. The TD(λ), SARSA(λ) and Q(λ) algorithms	51
2.5.5. Eligibility traces and TD(λ)	52
2.5.6. From TD(λ) to SARSA(λ)	55
2.5.7. Q(λ)	56
2.5.8. The R-learning algorithm	58
2.6. Model-based methods: learning a model	59
2.6.1. Dyna architectures	60
2.6.2. The E^3 algorithm	61
2.6.3. The R_{\max} algorithm	62

2.7. Conclusion	63
2.8. Bibliography	63
Chapter 3. Approximate Dynamic Programming	67
Rémi MUNOS	
3.1. Introduction	68
3.2. Approximate value iteration (AVI)	70
3.2.1. Sample-based implementation and supervised learning	71
3.2.2. Analysis of the AVI algorithm	73
3.2.3. Numerical illustration	74
3.3. Approximate policy iteration (API)	77
3.3.1. Analysis in L^∞ -norm of the API algorithm	77
3.3.2. Approximate policy evaluation	79
3.3.3. Linear approximation and least-squares methods	80
3.3.3.1. TD(λ)	81
3.3.3.2. Least-squares methods	82
3.3.3.3. Linear approximation of the state-action value function	85
3.4. Direct minimization of the Bellman residual	87
3.5. Towards an analysis of dynamic programming in L^p -norm	88
3.5.1. Intuition of an L^p analysis in dynamic programming	89
3.5.2. PAC bounds for RL algorithms	91
3.6. Conclusions	93
3.7. Bibliography	93
Chapter 4. Factored Markov Decision Processes	99
Thomas DEGRIS and Olivier SIGAUD	
4.1. Introduction	99
4.2. Modeling a problem with an FMDP	100
4.2.1. Representing the state space	100
4.2.2. The <i>Coffee Robot</i> example	100
4.2.3. Decomposition and function-specific independence	101
4.2.3.1. Transition function decomposition	102
4.2.3.2. Dynamic Bayesian networks in FMDPs	103
4.2.3.3. Factored model of the transition function in an FMDP	104
4.2.3.4. Factored model of the reward function	105
4.2.4. Context-specific independence	107
4.3. Planning with FMDPs	108
4.3.1. Structured policy iteration and structured value iteration	108
4.3.1.1. Decision trees	108
4.3.1.2. Representation of the transition function	108
4.3.1.3. Representation of the reward function	110
4.3.1.4. Representation of a policy	110
4.3.1.5. Representation of the value function	111
4.3.1.6. Algorithms	112

4.3.2. SPUDD: stochastic planning using decision diagrams	112
4.3.2.1. Representing the functions of an FMDP with ADDs	113
4.3.2.2. Algorithm	114
4.3.3. Approximate linear programming in FMDPs	115
4.3.3.1. Representations	116
4.3.3.2. Representation of the transition function	116
4.3.3.3. Representation of the reward function	117
4.3.3.4. Policy representation	118
4.3.3.5. Representation of the value function	119
4.3.3.6. Algorithms	122
4.4. Perspectives and conclusion	122
4.5. Bibliography	123
Chapter 5. Policy-Gradient Algorithms	127
Olivier BUFFET	
5.1. Reminder about the notion of gradient	128
5.1.1. Gradient of a function	128
5.1.2. Gradient descent	129
5.2. Optimizing a parameterized policy with a gradient algorithm	130
5.2.1. Application to MDPs: overview	130
5.2.1.1. First attempt to define a parameterized policy	131
5.2.1.2. Example of definition of a parameterized policy	131
5.2.2. Finite difference method	132
5.2.3. Gradient estimation of f in an MDP, case of the finite time horizon	133
5.2.3.1. Time horizon 1	133
5.2.3.2. Time horizon T	134
5.2.4. Extension to the infinite time horizon: discounted criterion, average criterion	137
5.2.4.1. Case of a regenerative process	138
5.2.4.2. Using a moving window	138
5.2.4.3. Using a discount factor	139
5.2.5. Partially observable case	139
5.2.6. Continuous action spaces	141
5.3. Actor-critic methods	143
5.3.1. A gradient estimator using the Q -values	143
5.3.2. Compatibility with an approximate value function	144
5.3.2.1. Approximating Q^θ	144
5.3.2.2. Compatibility of the approximators	145
5.3.3. An actor-critic algorithm	146
5.4. Complements	147
5.5. Conclusion	150
5.6. Bibliography	150

Chapter 6. Online Resolution Techniques	153
Laurent PÉRET and Frédéric GARCIA	
6.1. Introduction	153
6.1.1. Exploiting time online	153
6.1.2. Online search by simulation	154
6.2. Online algorithms for solving an MDP	155
6.2.1. Offline algorithms, online algorithms	155
6.2.2. Problem formalization	155
6.2.2.1. Forward search over a reasoning horizon	156
6.2.2.2. Tree or graph?	157
6.2.2.3. Complexity and efficiency of the forward search	157
6.2.3. Online heuristic search algorithms for MDPs	158
6.2.3.1. General principles	158
6.2.3.2. The RTDP algorithm	159
6.2.3.3. The <i>LAO</i> [*] algorithm	160
6.2.4. Kearns, Mansour and Ng's simulation-based algorithm	163
6.2.4.1. Complexity and convergence of Kearns <i>et al.</i> 's algorithm . .	165
6.2.4.2. Efficiency and practical considerations	165
6.2.5. Tesauro and Galperin's rollout algorithm	165
6.2.5.1. Complexity and convergence of the rollout algorithm . . .	166
6.2.5.2. Efficiency of the rollout algorithm	166
6.3. Controlling the search	167
6.3.1. Error bounds and pathology of the forward search	168
6.3.1.1. Pathology of the simulation-based forward search	168
6.3.1.2. Searching for a good compromise between depth and width	170
6.3.2. Iterative allocation of simulations	170
6.3.2.1. Multi-armed bandits and exploration in MDPs	171
6.3.3. Focused reinforcement learning	173
6.3.3.1. Global sampling error estimation	174
6.3.3.2. Organizing the search in successive trajectories	175
6.3.3.3. Convergence and practical considerations	176
6.3.4. Controlled rollout	178
6.3.4.1. Choice of the horizon	178
6.3.4.2. Iterative allocation of simulations	179
6.4. Conclusion	180
6.5. Bibliography	180
PART 2. BEYOND MDPs	185
Chapter 7. Partially Observable Markov Decision Processes	187
Alain DUTECH and Bruno SCHERRER	
7.1. Formal definitions for POMDPs	188
7.1.1. Definition of a POMDP	188

7.1.2. Performance criteria	189
7.1.3. Information state	190
7.1.3.1. Definition	190
7.1.3.2. Complete information state	191
7.1.3.3. Sufficient statistics	191
7.1.3.4. Belief states	191
7.1.4. Policy	193
7.1.5. Value function	195
7.2. Non-Markovian problems: incomplete information	196
7.2.1. Adapted policies	196
7.2.2. Discounted reward	197
7.2.2.1. Adapted stochastic policies	197
7.2.2.2. Adapted value function	198
7.2.2.3. Convergence of adapted algorithms	199
7.2.3. Adapted algorithms and adapted average reward criterion	201
7.3. Computation of an exact policy on information states	202
7.3.1. The general case	202
7.3.1.1. Finite horizon	202
7.3.1.2. Infinite horizon	203
7.3.2. Belief states and piecewise linear value function	203
7.3.2.1. Choice of the θ vectors	206
7.3.2.2. Infinite horizon	207
7.4. Exact value iteration algorithms	207
7.4.1. Steps for the dynamic programming operator	207
7.4.2. A parsimonious representation of V	210
7.4.2.1. Region	210
7.4.2.2. Parsimonious representation	212
7.4.2.3. Pruning of dominated vectors	213
7.4.2.4. Pruning	213
7.4.2.5. Choice of a vector for a belief state	215
7.4.3. The WITNESS algorithm	216
7.4.3.1. Neighborhood of a vector	216
7.4.3.2. The algorithm	218
7.4.4. Iterative pruning	219
7.4.4.1. Complete enumeration	219
7.4.4.2. Incremental enumeration	220
7.5. Policy iteration algorithms	222
7.6. Conclusion and perspectives	223
7.7. Bibliography	225
Chapter 8. Stochastic Games	229
Andriy BURKOV, Laëtitia MATIGNON and Brahim CHAIB-DRAA	
8.1. Introduction	229

8.2. Background on game theory	230
8.2.1. Some basic definitions	230
8.2.1.1. Criteria distinguishing different forms of games	230
8.2.2. Static games of complete information	232
8.2.2.1. Games in strategic form, pure strategy, mixed strategy	232
8.2.2.2. Zero-sum game and minimax	234
8.2.2.3. Equilibrium in dominating strategies	236
8.2.2.4. Nash equilibrium	238
8.2.3. Dynamic games of complete information	240
8.2.3.1. Games in extensive form with perfect information	241
8.2.3.2. Repeated games	243
8.3. Stochastic games	245
8.3.1. Definition and equilibrium of a stochastic game	246
8.3.2. Solving stochastic games	248
8.3.2.1. Game model available	249
8.3.2.2. Game model unavailable, \mathbf{A}_{-i} observed, equilibrium learners	252
8.3.2.3. Game model unavailable, \mathbf{A}_{-i} observed, opponent modeling	254
8.3.2.4. Game model unavailable, \mathbf{A}_{-i} not observed	256
8.3.3. Complexity and scalability of multi-agent learning algorithms . .	262
8.3.4. Beyond the search for an equilibrium	264
8.3.4.1. Efficient play	264
8.3.4.2. Regret minimization	265
8.3.4.3. Metastrategies	267
8.3.5. Discussion	267
8.4. Conclusion and outlook	269
8.5. Bibliography	270
Chapter 9. DEC-MDP/POMDP	277
Aurélie BEYNIER, François CHARPILLET, Daniel SZER and Abdel-Illah MOUADDIB	
9.1. Introduction	277
9.2. Preliminaries	278
9.3. Multiagent Markov decision processes	279
9.4. Decentralized control and local observability	280
9.4.1. Decentralized Markov decision processes	281
9.4.2. Multiagent team decision problems	283
9.4.3. Complexity	285
9.5. Sub-classes of DEC-POMDPs	285
9.5.1. Transition and observation independence	285
9.5.2. Goal oriented DEC-POMDPs	287
9.5.3. DEC-MDPs with constraints	288

9.5.3.1. Event-driven DEC-MDPs	289
9.5.3.2. Opportunity-cost DEC-MDPs	289
9.5.3.3. Problem statement	289
9.5.3.4. The OC-DEC-MDP model	290
9.5.4. Communication and DEC-POMDPs	292
9.5.4.1. DEC-POMDPs with communication	293
9.5.4.2. Complexity results	294
9.5.4.3. Discussion	294
9.6. Algorithms for solving DEC-POMDPs	295
9.6.1. Optimal algorithms	296
9.6.1.1. Dynamic programming for DEC-POMDPs	296
9.6.1.2. Heuristic search for DEC-POMDPs	299
9.6.1.3. Optimal algorithms for sub-classes of DEC-POMDPs	303
9.6.2. Approximate algorithms	303
9.6.2.1. Heuristics and approximate dynamic programming	304
9.6.2.2. Bounded memory	305
9.6.2.3. Co-evolutive algorithms	305
9.6.2.4. Gradient descent for policy search	307
9.6.2.5. Bayesian games	307
9.6.2.6. Heuristics for communicating agents	308
9.6.2.7. Approximate solutions for OC-DEC-MDPs	308
9.7. Applicative scenario: multirobot exploration	310
9.8. Conclusion and outlook	312
9.9. Bibliography	313
Chapter 10. Non-Standard Criteria	319
Matthieu BOUSSARD, Maroua BOUZID, Abdel-Illah MOUADDIB, Régis SABBADIN and Paul WENG	
10.1. Introduction	319
10.2. Multicriteria approaches	320
10.2.1. Multicriteria decision-making	321
10.2.2. Multicriteria MDPs	322
10.2.2.1. Operators for multicriteria decision-making	323
10.3. Robustness in MDPs	327
10.4. Possibilistic MDPs	329
10.4.1. Possibilistic counterpart of expected utility	330
10.4.2. Possibilistic dynamic programming	333
10.4.2.1. Finite horizon	333
10.4.2.2. Value iteration	334
10.4.2.3. Policy iteration	338
10.4.3. Extensions of possibilistic MDPs	338
10.4.3.1. Possibilistic reinforcement learning	339
10.4.3.2. Possibilistic partially observable MDPs	340
10.4.3.3. Possibilistic influence diagrams (PID)	342

10.5. Algebraic MDPs	342
10.5.1. Background	343
10.5.1.1. Semirings	343
10.5.1.2. Plausibility measures	344
10.5.1.3. Generalized expected utility	345
10.5.2. Definition of an algebraic MDP	345
10.5.3. Value function of a policy	347
10.5.4. Conditions	348
10.5.5. Examples of AMDPs	349
10.5.5.1. Probabilistic multicriteria AMDP	349
10.5.5.2. Possibilistic multicriteria AMDPs	350
10.5.5.3. AMDPs whose rewards are non-decreasing functions	351
10.6. Conclusion	354
10.7. Bibliography	355
PART 3. APPLICATIONS	361
Chapter 11. Online Learning for Micro-Object Manipulation	363
Guillaume LAURENT	
11.1. Introduction	363
11.2. Manipulation device	364
11.2.1. Micro-positioning by pushing	364
11.2.2. Manipulation device	365
11.2.3. Control loop	366
11.2.4. Representation of the manipulation task as an MDP	366
11.2.4.1. Definition of the state space	366
11.2.4.2. Definition of the action space	367
11.2.4.3. Definition of the reward function	367
11.2.4.4. Definition of an episode	367
11.3. Choice of the reinforcement learning algorithm	367
11.3.1. Characteristics of the MDP	367
11.3.2. A suitable algorithm: <i>STM-Q</i>	368
11.4. Experimental results	370
11.4.1. Experimental setup	370
11.4.2. Results	370
11.5. Conclusion	373
11.6. Bibliography	373
Chapter 12. Conservation of Biodiversity	375
Iadine CHADÈS	
12.1. Introduction	375
12.2. When to protect, survey or surrender cryptic endangered species	376
12.2.1. Surveying and managing the Sumatran tiger	376

12.2.2. The model	377
12.2.3. Results	377
12.2.4. Extension to more than one population	379
12.3. Can sea otters and abalone co-exist?	381
12.3.1. Abalone and sea otters: two endangered species	381
12.3.2. The models	382
12.3.2.1. Population dynamics of abalone	382
12.3.2.2. Sea otter population model	383
12.3.2.3. States	384
12.3.2.4. Decisions	385
12.3.2.5. Interaction between sea otters and abalone	386
12.3.2.6. Multicriteria objective and reward function	386
12.3.3. Methods	387
12.3.4. Results	387
12.3.4.1. Scenario 1: sea otter reintroduction and anti-poaching enforcement	387
12.3.4.2. Scenario 2: control of sea otters	389
12.3.4.3. Scenario 3: combined action of sea otter control and anti-poaching	389
12.3.5. Conclusion	389
12.4. Other applications in conservation biology and discussions	391
12.5. Bibliography	392
Chapter 13. Autonomous Helicopter Searching for a Landing Area in an Uncertain Environment	395
Patrick FABIANI and Florent TEICHTEIL-KÖNIGSBUCH	
13.1. Introduction	395
13.2. Exploration scenario	397
13.2.1. Planning problem	398
13.2.2. States and actions	399
13.2.3. Uncertainties	400
13.2.4. Optimization criterion	400
13.2.5. Formalization of the decision problem	401
13.3. Embedded control and decision architecture	401
13.3.1. Global view	401
13.3.2. Multi-thread planning triggered by the supervisor	403
13.3.2.1. Policy optimization	403
13.3.2.2. Dialogue with the supervisor	404
13.4. Incremental stochastic dynamic programming	404
13.4.1. Obtaining the initial safe policy quickly	405
13.4.2. Generating the sub-space of reachable states	405
13.4.3. Local policy optimization	406
13.4.4. Launching local replanning processes	407

13.5. Flight tests and return on experience	407
13.6. Conclusion	410
13.7. Bibliography	410
Chapter 14. Resource Consumption Control for an Autonomous Robot	413
Simon LE GLOANNEC and Abdel-Illah MOUADDIB	
14.1. The rover’s mission	414
14.2. Progressive processing formalism	415
14.3. MDP/PRU model	416
14.3.1. States	416
14.3.2. Actions	417
14.3.3. Transition function	418
14.3.4. Reward function	418
14.4. Policy calculation	418
14.4.1. Value function	419
14.4.2. Propagation algorithm	419
14.5. How to model a real mission	419
14.6. Extensions	422
14.7. Conclusion	423
14.8. Bibliography	423
Chapter 15. Operations Planning	425
Sylvie THIÉBAUX and Olivier BUFFET	
15.1. Operations planning	425
15.1.1. Intuition	425
15.1.1.1. Problem features	426
15.1.1.2. Plans	427
15.1.2. Formal definitions	428
15.1.2.1. Planning problem, operations	429
15.1.2.2. Execution	431
15.1.2.3. Decision epochs, states, plans	432
15.1.2.4. Objective	432
15.2. MDP value function approaches	433
15.2.1. Formalizations, CoMDP	433
15.2.1.1. States, actions, transitions	433
15.2.1.2. Rewards, costs	434
15.2.2. Algorithms	435
15.2.2.1. (L)RTDP	435
15.2.2.2. Memory management	435
15.2.2.3. Reduction of the number of updates	436
15.2.2.4. Hybrid algorithms	436
15.2.2.5. Algorithms with upper bounds	437

15.2.3. Heuristics	438
15.2.3.1. Basic heuristics	438
15.2.3.2. Heuristics obtained by relaxation of the CoMDP	439
15.2.3.3. Planning graph heuristics	440
15.3. Reinforcement learning: FPG	442
15.3.1. Employing approximate methods	442
15.3.2. Parameterized policy	443
15.3.2.1. Inputs	443
15.3.2.2. Outputs	443
15.3.2.3. Function approximator	444
15.3.3. Gradient methods	445
15.3.3.1. Terminating an execution	445
15.3.3.2. Choice of OLpomdp	445
15.3.3.3. Optimized criterion	445
15.3.4. Improving FPG	446
15.4. Experiments	446
15.5. Conclusion and outlook	448
15.6. Bibliography	450
Index	453

Preface

The present book discusses sequential decision-making under uncertainty and reinforcement learning, two classes of problems in artificial intelligence which can be formalized in the framework of Markov decision processes. It has been written for students, engineers and researchers likely to be interested in these fields and models.

The book is organized as follows:

- Part 1 provides an introduction to this domain and to efficient resolution techniques (Markov decision processes, reinforcement learning, approximate representations, factored representations, policy gradients and online resolution).
- Part 2 presents important extensions of Markov decision processes that make it possible to solve more complex sequential decision-making problems (partially observable Markov decision processes, Markov games, multi-agent approaches and non-classical criteria).
- Part 3 completes the book with example applications showing how Markov decision processes can be employed for various problems (micro-object manipulation, biodiversity preservation, high-level control of a helicopter, control of an exploration mission and operations planning).

It was not possible for this book to cover all research directions in this very active field. We give here some references to point the reader to some uncovered aspects. For example, we have decided not to cover continuous time reinforcement learning [MUN 01], relational reinforcement learning [DZE 01], hierarchical reinforcement learning [BAR 03], learning classifier systems [SIG 07] or predictive state representations [LIT 02].

In addition, we endeavor in each chapter to provide the reader with references to related work.

Additional information related to this book (e.g. *errata*) can be found at the following website: <http://www.loria.fr/projets/PDMIA/Book/>.

Bibliography

- [BAR 03] BARTO A. and MAHADEVAN S., “Recent advances in hierarchical reinforcement learning”, *Discrete Event Dynamic Systems*, vol. 13, no. 4, pp. 341–379, 2003.
- [DZE 01] DZEROSKI S., DE RAEDT L. and DRIESSENS K., “Relational reinforcement learning”, *Machine Learning*, vol. 43, no. 1-2, pp. 7–53, 2001.
- [LIT 02] LITTMAN M., SUTTON R. and SINGH S., “Predictive representations of state”, *Advances in Neural Information Processing Systems 14 (NIPS'01)*, MIT Press, Cambridge, MA, pp. 1555–1561, 2002.
- [MUN 01] MUNOS R. and MOORE A., “Variable resolution discretization in optimal control”, *Machine Learning*, vol. 49, pp. 291–323, 2001.
- [SIG 07] SIGAUD O. and WILSON S. W., “Learning classifier systems: a survey”, *Soft Computing*, vol. 11, no. 11, pp. 1065–1078, 2007.

List of Authors

Aurélie Beynier

Aurélie Beynier is currently an associate professor of computer science at the LIP6 Laboratory. Her PhD thesis has been defended in November 2006 at the University of Caen.

Matthieu Boussard

Matthieu Boussard holds a PhD in computer science from the University of Caen. He is a member of the MAD Group of the GREYC Laboratory.

Maroua Bouzid

Maroua Bouzid is an associate professor at the University of Caen. She is a member of the MAD Group of the GREYC Laboratory.

Olivier Buffet

Olivier Buffet is an INRIA junior research scientist. He is member of the Autonomous Intelligent Machines (MAIA) Team of the LORIA Laboratory.

Andriy Burkov

Andriy Burkov is a PhD student working under supervision of Professor Brahim Chaib-Draa at Laval University, Canada. His main research interests include multiagent learning and game theory.

Iadine Chadès

Iadine Chadès is currently a research fellow at CSIRO Sustainable Ecosystems, Australia, on leave from the French National Institute for Agricultural Research (INRA).

Brahim Chaib-Draa

Professor Brahim Chaib-Draa is the leader of the DAMAS Research Group on Agents and Multiagent Systems at the Computer Science and Software Engineering Department of Laval University, Canada.

François Charpillet

François Charpillet is an INRIA senior research scientist. He is the head of the Autonomous Intelligent Machines (MAIA) Team of the LORIA Laboratory.

Thomas Degris

Thomas Degris holds a PhD in computer science. After working in the video game industry, he is now a postdoctoral fellow at the University of Alberta.

Alain Dutech

Alain Dutech is an INRIA experienced research scientist. He is a member of the Autonomous Intelligent Machines (MAIA) Team of the LORIA Laboratory.

Patrick Fabiani

Patrick Fabiani is the director of the Systems Control and Flight Dynamics Department at ONERA. His research interests include models, methods and algorithms for sequential decision making and planning under uncertainty, applied to autonomous aerial robots in the ReSSAC project at ONERA.

Frédéric Garcia

Frédéric Garcia is a researcher in artificial intelligence at the Department of Applied Mathematics and Informatics at INRA (the French National Institute for Agricultural Research).

Guillaume Laurent

Guillaume Laurent is an associate professor at the ENSMM Graduate School at Besançon. He is a member of the Automatic Control and Micro-Mechatronic Systems Department of the FEMTO-ST Research Institute.

Simon Le Gloannec

Simon Le Gloannec holds a PhD from the University of Caen. He has worked with both the MAIA Team (LORIA laboratory) and the MAD Group (GREYC Laboratory). He currently holds a postdoctoral position at GREYC.

Laëtitia Matignon

Laëtitia Matignon holds a PhD from the University of Franche-Comté at Besançon. She is now a postdoctoral fellow of the Cooperative Decision-Theoretic Autonomous Agent System Group (MAD) of the GREYC Laboratory in Caen.

Abdel-Illah Mouaddib

Abdel-Illah Mouaddib is a full professor at the University of Caen. He is the head of the Cooperative Decision-Theoretic Autonomous Agent System Group (MAD) of the GREYC Laboratory.

Rémi Munos

Rémi Munos is senior researcher at INRIA Lille, France. He works in the fields of reinforcement learning, optimal control and decision theory.

Laurent Péret

Laurent Péret holds a PhD in artificial intelligence, focused on search methods for MDPs. Since 2005, he has been involved in various space programs as a flight dynamics engineer.

Emmanuel Rachelson

Emmanuel Rachelson is a postdoctoral fellow at the University of Liège, Belgium, working on reinforcement and statistical learning. He also works with Pr. Lagoudakis in Greece and with the Electricité de France (EDF) Research Center in Paris.

Régis Sabbadin

Régis Sabbadin has been a research scientist at INRA-Toulouse since 1999. His research focuses on planning under uncertainty, applied to natural resources management and agriculture.

Bruno Scherrer

Bruno Scherrer is an INRIA experienced research scientist. He is member of the Autonomous Intelligent Machines (MAIA) Team of the LORIA Laboratory.

Olivier Sigaud

Olivier Sigaud is a professor of computer science at the UPMC-Paris 6 University. He is the head of the “Motion” Group at the Institute of Intelligent Systems and Robotics (ISIR).

Daniel Szer

Daniel Szer obtained a Master's degree in computer science from UMass Dartmouth and a PhD in artificial intelligence from Henri-Poincaré University, Nancy. He works as an IT analyst in Paris.

Florent Teichteil-Königsbuch

Florent Teichteil-Königsbuch is a researcher in probabilistic sequential decision-making at ONERA. He got a PhD in artificial intelligence from SUPAERO in 2005.

Sylvie Thiébaux

Sylvie Thiébaux is an associate professor at the Australian National University and principal researcher at the National ICT Australia, specializing in automated planning in artificial intelligence.

Paul Weng

Paul Weng has been an associate professor at Paris 6 University since September 2007. Before his PhD obtained in 2006, he worked in finance in London.

Part 1

MDPs: Models and Methods

Chapter 1

Markov Decision Processes

1.1. Introduction

This book presents a decision problem type commonly called *sequential decision problems under uncertainty*. The first feature of such problems resides in the relation between the current decision and future decisions. Indeed, these problems do not consist of one, but several decision problems, presented in a sequence. At each step of this sequence, the *agent* (actor or decision-maker) needs to decide on the current action by taking into account its effect on the solution of future problems. This sequential feature is also typical of *planning* problems in artificial intelligence and is often linked with shortest path methods in graph theory. The second characteristic of the problems discussed in these pages is the uncertainty in the consequences of all available decisions (actions). Knowledge of its decision's effects is not available in advance to the agent in a deterministic form. As such, this problem deals with the various theories of decision under uncertainty which suggest different formalization and resolution approaches. Among these approaches, we need to mention specifically the standard theory of expected utility maximization.

Consequently, problems of sequential decision under uncertainty couple the two problematics of sequential decision and decision under uncertainty. *Markov decision problems* (MDPs) are a general mathematical formalism for representing shortest path problems in stochastic environments. This formalism is based on the theory of *Markov decision processes* (also written as MDPs). A Markov decision process relies on the notions of *state*, describing the current situation of the agent, *action* (or decision), affecting the dynamics of the process, and *reward*, observed for each

Chapter written by Frédéric GARCIA and Emmanuel RACHELSON.

4 Markov Decision Processes in AI

transition between states. Such a process describes the probability of triggering a transition to state s' and receiving a certain reward r when taking decision a in state s . Hence, an MDP can be described as a controlled Markov chain, where the control is given at each step by the chosen action. The process then visits a sequence of states and can be evaluated through the observed rewards. Solving an MDP consists of controlling the agent in order to reach an optimal behavior, i.e. to maximize its overall revenue. Because action effects are stochastic and, thus, can result in different possible states at the next stage of the decision process, the optimal control strategy cannot necessarily be represented as a single sequence of actions.¹ Consequently, solutions of an MDP are usually given as *universal plans* or *policies* (*strategies* or *decision rules*) specifying which action to undertake at each step of the decision process and for every possible state reached by the agent. Due to the uncertainty in actions' results, applying a given policy can result in different sequences of states/actions.

EXAMPLE 1.1. Let us illustrate these concepts with a simple car maintenance example. According to the current state of the car (breakdown, wear, age, etc.), an agent wishes to decide which is its best strategy (do nothing, replace parts preventively, repair, change car, etc.) in order to minimize the maintenance cost in the long run. Assuming the agent knows the consequences and the cost of each separate action in every possible state (e.g. we know the failure probability of an engine if the oil leak is not fixed), we can model this problem as an MDP. Solving this MDP will provide the agent with a policy indicating which is the optimal action to undertake in every state of the problem. This way, the sequence of actions performed as the car's state changes will allow the agent to always minimize the expected maintenance cost.

The theory of Markov decision processes and its generalizations will be developed in the next chapters. These models have become the most popular framework for representing and solving problems of sequential decision under uncertainty. This chapter presents the basics of MDP theory and optimization, in the case of an agent having a perfect knowledge of the decision process and of its state at every time step, when the agent's goal is to maximize its global revenue over time.

1.2. Markov decision problems

1.2.1. *Markov decision processes*

Markov decision processes are defined as *controlled stochastic processes* satisfying the Markov property and assigning reward values to state transitions [BER 87, PUT 94]. Formally, they are described by the 5-tuple (S, A, T, p, r) where:

1. Contrary to the deterministic approaches of classical planning.

- S is the state space in which the process' evolution takes place;
- A is the set of all possible actions which control the state dynamics;
- T is the set of time steps where decisions need to be made;
- $p()$ denotes the state transition probability function;
- $r()$ provides the reward function defined on state transitions.

Figure 1.1 represents an MDP, drawn as an influence diagram. At every time step t in T , action a_t is applied in the current state s_t , affecting the process in its transition to the next state s_{t+1} . Reward r_t is then obtained for this transition.

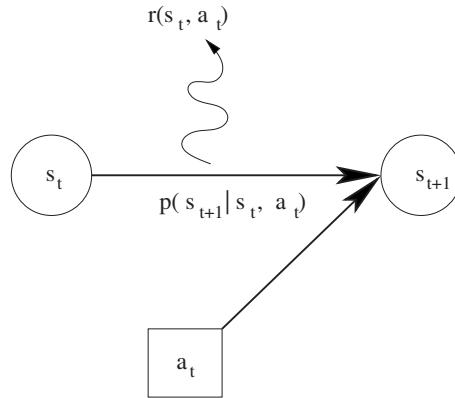


Figure 1.1. Markov decision process

The set T of decision epochs is a discrete set, subset of \mathbb{N} , which can either be finite or infinite (then we talk, respectively, about finite horizon or infinite horizon). A third case corresponds to the existence of a set of terminal states (or goal states). In this case, the process stops as soon as one of these states is encountered. Then, the horizon is then said to be indefinite. These problems are often related to stochastic shortest path problems. This case, however, can be seen as a specific case of infinite horizon MDPs with absorbing states and will not be presented in detail in this chapter (see Chapter 6, section 6.2.3 and Chapter 15).

In the most general case, the S and A sets are supposed finite, even though many results can be extended to countable or even continuous sets (see [BER 95] for an introduction to the continuous case). Generally, the set A of applicable actions can also depend on the current state: we define a subset A_s of applicable actions in state s . Similarly, S and A can change based on the time step t (S_t and A_t). However, in this chapter, for clarity of presentation, we will restrict ourselves to the standard case where A and S are constant throughout the process.

6 Markov Decision Processes in AI

The transition probabilities $p()$ characterize the state dynamics of the system, i.e. indicate which states are likely to appear after the current state. For a given action a , $p(s' | s, a)$ represents the probability for the system to transit to state s' after undertaking action a in state s . Because the $p()$ values are probabilities, we classically have $\forall s, a, \sum_{s'} p(s' | s, a) = 1$. This $p()$ function is usually represented in matrix form, where we write P_a the $|S| \times |S|$ matrix containing elements $\forall s, s' P_{a,s,s'} = p(s' | s, a)$. Consequently, the probability transitions of the decision process are given as $|A|$ matrices P_a . Since each line of these matrices sums to one, the P_a are said to be *stochastic* matrices.

The $p()$ probability distributions over the next state s' follow the fundamental property which gives their name to Markov decision processes. If we write $h_t = (s_0, a_0, \dots, s_{t-1}, a_{t-1}, s_t)$ the history of states and actions until time step t , then the probability of reaching state s_{t+1} consecutively to action a_t is only a function of a_t and s_t , and not of the entire history h_t . Let us write $P(x | y)$ the conditional probability of event x , provided that y is true, then we have

$$\forall h_t, a_t, s_{t+1} \quad P(s_{t+1} | h_t, a_t) = P(s_{t+1} | s_t, a_t) = p(s_{t+1} | s_t, a_t).$$

We should note here that the previous condition does not necessarily imply that the resulting stochastic process $(s_t)_{t \in T}$ itself respects the Markov property: this also depends on the action choice policy for a_t .

As a result of choosing action a , in state s , at time t , the deciding agent receives a reward $r_t = r(s, a) \in \mathbb{R}$. We can consider positive values of r_t as gains and negative values as costs. We also sometimes use a cost function $c()$ instead of the reward function $r()$. This reward can be received instantaneously at time t or accumulated between t and $t + 1$. The important feature is that this reward only depends on the simple input of the current state s and the current action a . A vector representation of the $r(s, a)$ reward function consists of $|A|$ vectors r_a of length $|S|$.

A common extension consists of considering random rewards. In this case, we will use their average value for the reward function $r(s, a) = \bar{r}(s, a)$. In particular, the reward obtained at time step t can depend on the final state s' of the transition. We then have a reward specified as $r(s, a, s')$. The value used for the reward vectors is $\bar{r}(s, a) = \sum_{s'} p(s' | s, a)r(s, a, s')$. In all cases, r_t is supposed bounded.

Finally, as for S and A , the transition and reward functions can vary across time. In this case they are written, respectively, as p_t and r_t . When these functions do not change from one step to the other, the process is said to be *stationary*: $\forall t \in T, p_t() = p(), r_t() = r()$. In the rest of this chapter, we will keep this stationarity hypothesis in the study of infinite horizon MDPs.

1.2.2. Action policies

Markov decision processes allow us to model the state evolution dynamics of a stochastic system when this system is controlled by an agent choosing and applying the actions a_t at every time step t . The procedure of choosing such actions is called an action policy, or strategy, and is written as π . A policy can decide deterministically upon the action to apply or can define a probability distribution over the possible applicable actions. Then, a policy can be based on the whole history h_t of the process (it is called history-dependent) or can only consider the current state s_t . This way, we obtain four main families of policies, as shown in Table 1.1.

Policy π_t	Deterministic	Stochastic
Markov	$s_t \longrightarrow a_t$	$a_t, s_t \longrightarrow [0, 1]$
History-dependent	$h_t \longrightarrow a_t$	$h_t, s_t \longrightarrow [0, 1]$

Table 1.1. Different policy families for MDPs

For a deterministic policy, $\pi_t(s_t)$ or $\pi_t(h_t)$ defines the chosen action a_t . For a stochastic policy, $\pi_t(a, s_t)$ or $\pi_t(a, h_t)$ represents the probability of selecting $a \in A$ for a_t .

These four families define the following sets:

- Π^{HA} for stochastic, history-dependent policies,
- Π^{HD} for deterministic, history-dependent policies,
- Π^{MA} for stochastic, Markov policies,
- Π^{MD} for deterministic, Markov policies.

These sets are included in each other, from the most general case of stochastic, history-dependent policies, to the very specific case of deterministic, Markov policies, as shown in Figure 1.2.

Independently of these different sets of policies, the policy definition can also depend explicitly on the time step. Hence, a policy is called *stationary* if $\forall t, \pi_t = \pi$. Among the stationary policies, deterministic, Markov ones are central to the study of MDPs. They represent the simplest model of decision strategies and their set is denoted by \mathcal{D} .

8 Markov Decision Processes in AI

DEFINITION 1.1 (stationary, Markov, deterministic policies). \mathcal{D} is the set of all functions π linking states of S to actions of A :

$$\pi : s \in S \longrightarrow \pi(s) \in A.$$

Another important set of policies, written as \mathcal{D}^A , is composed of stationary, Markov, stochastic policies. The \mathcal{D} and \mathcal{D}^A sets are important because they contain optimal policies for the most common criteria.

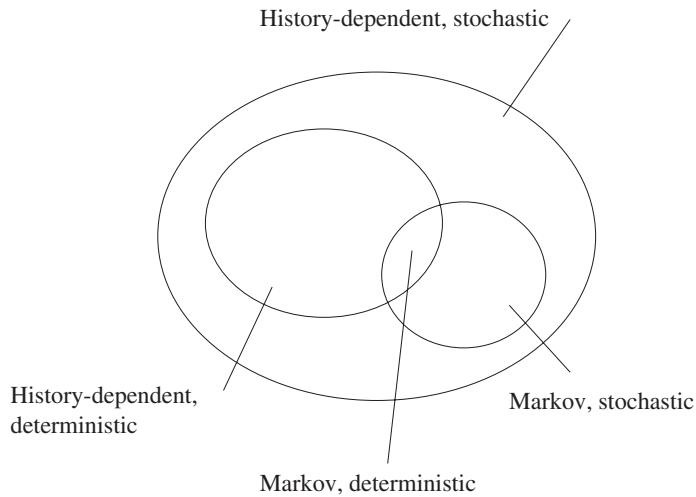


Figure 1.2. Relationship between the different sets of policies

1.2.3. Performance criterion

Solving a Markov decision problem implies searching for a policy, in a given set, which optimizes a performance (or optimality) criterion for the considered MDP. This criterion aims at characterizing the policies which will provide the best sequences of rewards. Formally, it corresponds to evaluating a policy based on a measure of the *expected cumulative sum* of instantaneous rewards along the trajectory. The main criteria studied in the theory of MDPs are:

- the finite criterion: $E[r_0 + r_1 + r_2 + \dots + r_{N-1} | s_0]$,
- the (γ) -discounted criterion: $E[r_0 + \gamma r_1 + \gamma^2 r_2 + \dots + \gamma^t r_t + \dots | s_0]$,
- the total reward criterion: $E[r_0 + r_1 + r_2 + \dots + r_t + \dots | s_0]$,
- the average criterion: $\lim_{n \rightarrow \infty} \frac{1}{n} E[r_0 + r_1 + r_2 + \dots + r_{n-1} | s_0]$.

These four criteria have two common features. First, they are additive in r_t , which is a simple way of aggregating all the rewards obtained along a trajectory. Second, they are defined with respect to an expectation operator $E[\cdot]$, taking into account the distribution of trajectories for a given policy and initial state. This choice of an expected cumulative sum is crucial since it makes it possible to establish the *Bellman optimality principle* [BEL 57] (“all sub-policies of an optimal policy are optimal sub-policies”). This principle underlies the dynamic programming approach and allows us to solve MDPs efficiently. Chapter 10 will present non-classical criteria for which this optimality principle cannot be formulated the same way.

In the rest of this chapter, we will successively characterize optimal policies for each of the above criteria. In each case, we will present adapted algorithms to obtain the optimal policies.

1.3. Value functions

The finite, discounted, total reward, and average criteria presented above allow the definition of a *value function*. For a given policy π and a criterion among the above, such a function links any starting state s to the value of the considered criterion. This value corresponds to the expected gain, according to the considered criterion, when we apply policy π , starting in s : $\forall \pi V^\pi : S \rightarrow \mathbb{R}$.

Let \mathcal{V} be the space of all functions mapping S to \mathbb{R} . This space can be identified to the vector space $\mathbb{R}^{|S|}$. The set \mathcal{V} has a natural partial order:

$$\forall U, V \in \mathcal{V} \quad U \leq V \iff \forall s \in S \quad U(s) \leq V(s).$$

The goal of an MDP solver is to characterize and find the optimal policies with respect to a given criterion (if such policies exist). Formally, it corresponds to finding $\pi^* \in \Pi^{HA}$ such that

$$\forall \pi \in \Pi^{HA} \quad \forall s \in S \quad V^\pi(s) \leq V^{\pi^*}(s)$$

or, again,

$$\pi^* \in \operatorname{argmax}_{\pi \in \Pi^{HA}} V^\pi.$$

Then, we can write $V^* = \max_{\pi \in \Pi^{HA}} V^\pi = V^{\pi^*}$. In the MDP framework, we often search for an optimal policy which is better than any other policy, independently of the starting state. We can immediately remark that the existence of such a policy is not obvious.

Value functions are a specific feature of Markov decision problems: the search for an optimal policy can be directly transformed into an optimization problem expressed in terms of value functions. Then, the resolution of the optimality equations is less complex than the exhaustive exploration of the whole set of Π^{HA} policies.

1.3.1. The finite criterion

A prior assumption linked to this criterion implies the agent has to control the system within N steps, where N is finite. The finite criterion naturally leads to define a value function linking every state s to the expected sum of the next N rewards obtained by applying policy π from s :

DEFINITION 1.2 (value function for the finite criterion). *Let $T = \{0, \dots, N - 1\}$, then we write*

$$\forall s \in S, \quad V_N^\pi(s) = E^\pi \left[\sum_{t=0}^{N-1} r_t \mid s_0 = s \right].$$

In this definition, $E^\pi[\cdot]$ denotes the mathematical expectation over the whole set of possible outcomes in the MDP's trajectory, while following policy π . E^π is associated with the probability distribution P^π over these outcomes.

Sometimes, in practice, it may be useful to add a terminal reward r_N to the criterion, this reward being only a function of the final state s_N . For this purpose, we can consider an artificial additional step where $\forall (s, a) \in S \times A$, $r_N(s, a) = r_N(s)$. This sort of requirement arises in particular when the goal is to command a system, towards a goal state, in N steps, and with optimal (least) cost.

1.3.2. The γ -discounted criterion

The discounted (or γ -discounted) criterion is the most commonly used, infinite horizon criterion. Its corresponding value function describes the limit, when N tends to infinity, of the expectation (given policy π) on the sum of the N future rewards, each reward being weighted by an *update* or *discount factor*.

DEFINITION 1.3 (value function for the discounted criterion). *Let $0 \leq \gamma < 1$; we write*

$$\begin{aligned} \forall s \in S, \quad V_\gamma^\pi(s) &= E^\pi [r_0 + \gamma r_1 + \gamma^2 r_2 + \dots + \gamma^t r_t + \dots \mid s_0 = s] \\ &= E^\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s \right]. \end{aligned}$$

The γ factor represents the value, considered from time t , of a unit reward perceived at time $t + 1$. If it were received at time $t + \tau$, this same unit reward would be perceived as γ^τ . Hence, this favors rewards received early in the process. Such an

expression requires the decision epochs to be regularly dispatched in \mathbb{N} . The main formal interest of the γ factor is to guarantee the convergence of the series used in the criterion and, thus, the existence of a value for the criterion in all cases. In practice, this discount factor is naturally introduced in the modeling process of economic decision problems by using $\gamma = \frac{1}{1+\tau}$, where τ is the inflation rate's value.

1.3.3. The total reward criterion

However, we can still choose $\gamma = 1$ in some specific cases of infinite horizon problems. When this makes sense, we can write the following definition.

DEFINITION 1.4 (value function for the total reward criterion).

$$V^\pi(s) = E^\pi \left[\sum_{t=0}^{\infty} r_t \mid s_0 = s \right].$$

This criterion is often used when we know the horizon is finite but do not know its exact value. In this case, the process will terminate in a finite number of steps with probability one, but an upper bound on this number of steps is not available. Such models often correspond to *optimal stopping* problems (where the agent's decision at each time step deals directly with the termination of the stochastic process), games or bets.

1.3.4. The average reward criterion

When decisions have to be made frequently with a discount factor close to one, or when it appears impossible to value directly the rewards, it is preferable to consider a criterion reflecting the average value of the received rewards, along the trajectories, instead of their weighted sum. Hence, we can form a value function by linking states to the average gain per step, on an infinite horizon application of policy π . This is how the average reward criterion $\rho^\pi(s)$ is defined as follows.

DEFINITION 1.5 (average reward value function).

$$\rho^\pi(s) = \lim_{n \rightarrow \infty} E^\pi \left[\frac{1}{n} \sum_{t=0}^{n-1} r_t \mid s_0 = s \right].$$

A policy π^* is said to be *gain-optimal* for the average criterion if $\rho^{\pi^*}(s) \geq \rho^\pi(s)$ for any policy π and state s .

The average reward criterion is used in particular in queueing control, communication networks or stock management.

1.4. Markov policies

1.4.1. Equivalence of history-dependent and Markov policies

We will establish here a fundamental property of MDPs for the criteria presented above. This property makes it possible to restrict the search for optimal policies only to the set of Markov policies, without considering the super-set Π^{HA} of history-dependent policies.

PROPOSITION 1.1. *Let $\pi \in \Pi^{HA}$ be a stochastic, history-dependent policy. For each initial state $s \in S$, there exists a stochastic Markov policy $\pi' \in \Pi^{MA}$ such that*

- 1) $V_N^{\pi'}(s) = V_N^\pi(s)$,
- 2) $V_\gamma^{\pi'}(s) = V_\gamma^\pi(s)$,
- 3) $V^{\pi'}(s) = V^\pi(s)$,
- 4) $\rho^{\pi'}(s) = \rho^\pi(s)$.

Proof. Let $s \in S$ be a starting state and π a stochastic, history-dependent policy. Let π' be the stochastic Markov policy defined from π and x according to

$$\forall(t, x, a) \in \mathbb{N} \times S \times A, \quad \pi'(a_t = a, s_t = x) = P^\pi(a_t = a \mid s_t = x, s_0 = s).$$

Thus, we have $P^{\pi'}(a_t = a \mid s_t = x) = P^\pi(a_t = a \mid s_t = x, s_0 = s)$. Let us show, by induction over t , that $P^\pi(s_t = x, a_t = a \mid s_0 = s) = P^{\pi'}(s_t = x, a_t = a \mid s_0 = s)$.

For $t = 0$, this equality is straightforward. Let us suppose the property is true until $t - 1$, then we have

$$\begin{aligned} P^\pi(s_t = x \mid s_0 = s) &= \sum_{i \in S} \sum_{a \in A} P^\pi(s_{t-1} = i, a_{t-1} = a \mid s_0 = s) p(x \mid i, a) \\ &= \sum_{i \in S} \sum_{a \in A} P^{\pi'}(s_{t-1} = i, a_{t-1} = a \mid s_0 = s) p(x \mid i, a) \\ &= P^{\pi'}(s_t = x \mid s_0 = s). \end{aligned}$$

So

$$\begin{aligned} P^{\pi'}(s_t = x, a_t = a \mid s_0 = s) &= P^{\pi'}(a_t = a \mid s_t = x) P^{\pi'}(s_t = x \mid s_0 = s) \\ &= P^\pi(a_t = a \mid s_t = x, s_0 = s) P^\pi(s_t = x \mid s_0 = s) \\ &= P^\pi(s_t = x, a_t = a \mid s_0 = s), \end{aligned}$$

which proves the property for t .

We can conclude by remarking that, for all $s \in S$,

$$\begin{aligned} V_N^\pi(s) &= \sum_{t=0}^{t=N-1} E^\pi[r(s_t, a_t) \mid s_0 = s], \\ V_\gamma^\pi(s) &= \sum_{t=0}^{t=\infty} \gamma^t E^\pi[r(s_t, a_t) \mid s_0 = s], \\ V^\pi(s) &= \sum_{t=0}^{t=\infty} E^\pi[r(s_t, a_t) \mid s_0 = s], \\ \rho^\pi(s) &= \lim_{n \rightarrow \infty} \frac{1}{n} \sum_{t=0}^{t=n-1} E^\pi[r(s_t, a_t) \mid s_0 = s], \end{aligned}$$

and

$$E^\pi[r(s_t, a_t) \mid s_0 = s] = \sum_{x \in S} \sum_{a \in A} r(x, a) P^\pi(s_t = x, a_t = a \mid s_0 = s). \quad \square$$

This result allows the following statement: when we know the process' initial state (or a probability distribution over this initial state), then every stochastic, history-dependent policy can be replaced by a stochastic, Markov policy which retains the same value function.

1.4.2. Markov policies and valued Markov chains

For any Markov policy $\pi \in \Pi^{MA}$, the stochastic process associated with the state variables s_t verifies, for all s_0, s_1, \dots, s_{t+1} ,

$$\begin{aligned} P^\pi(s_{t+1} \mid s_0, s_1, \dots, s_t) &= \sum_{a \in A} P^\pi(a_t = a \mid s_0, s_1, \dots, s_t) \\ &\quad \times P^\pi(s_{t+1} \mid s_0, s_1, \dots, s_t, a_t = a) \\ &= \sum_{a \in A} \pi(a, s_t) P^\pi(s_{t+1} \mid s_t, a_t = a) \\ &= P^\pi(s_{t+1} \mid s_t). \end{aligned}$$

Hence, this is a Markov process, forming a Markov chain whose transition matrix is denoted by P_π and is defined as

$$\forall s, s' \quad P_{\pi, s, s'} = P^\pi(s_{t+1} = s' \mid s_t = s) = \sum_a \pi(a, s) p(s' \mid s, a).$$

In the specific case of a deterministic policy ($\pi \in \Pi^{MD}$), $P_{\pi,s,s'}$ is directly equal to $p(s' | s, \pi(s))$. The P_π matrix is simply built by keeping, for each state s , the corresponding line in the P_a matrix, with $a = \pi(s)$. Similarly, we fill the r_π vector with $r(s, \pi(s))$ elements for $\pi \in \Pi^{MD}$ and with $\sum_a \pi(a, s) r(s, a)$ values for $\pi \in \Pi^{MA}$.

The triplet (S, P_π, r_π) defines a so-called *valued Markov process*, or *valued Markov chain* or *Markov reward process*. Such a chain is simply a standard Markov chain with reward values attached to the state transitions. We will see that evaluating a policy π consists of calculating characteristic asymptotic values of this chain (for infinite horizon criteria).

1.5. Characterization of optimal policies

1.5.1. The finite criterion

1.5.1.1. Optimality equations

Suppose the agent is in state s , at the last decision epoch ($N - 1$), and has to choose an action. Quite obviously, since this is the last decision epoch, the best decision is to take the action which maximizes the upcoming instantaneous reward. This reward will add up to the ones previously received. Consequently,

$$\pi_{N-1}^*(s) \in \operatorname{argmax}_{a \in A} r_{N-1}(s, a)$$

and

$$V_1^*(s) = \max_{a \in A} r_{N-1}(s, a),$$

where π_{N-1}^* is the optimal policy at decision epoch $N - 1$ and V_1^* is the optimal value function for a length one horizon corresponding to this optimal policy.

Suppose now the agent is in state s at decision epoch $N - 1$. Its choice of action will directly induce reward $r_{N-2}(s, a)$ and will take it randomly to a new state s' (according to the transition model p), at decision epoch $N - 1$. There, by following the optimal policy π_{N-1}^* , it will receive an average reward of $V_1^*(s')$. Consequently, the action choice at time step $N - 2$ leads in the best case to a maximum reward corresponding to the expected value of $r_{N-2}(s, a) + \sum_{s'} p_{N-2}(s' | s, a) V_1^*(s')$. So the decision problem at decision epoch $N - 2$ implies finding the action a which maximizes this sum. In other words,

$$\pi_{N-2}^*(s) \in \operatorname{argmax}_{a \in A} \left\{ r_{N-2}(s, a) + \sum_{s'} p_{N-2}(s' | s, a) V_1^*(s') \right\}$$

and

$$V_2^*(s) = \max_{a \in A} \left\{ r_{N-2}(s, a) + \sum_{s'} p_{N-2}(s' | s, a) V_1^*(s') \right\}.$$

This reasoning can be repeated until the first decision epoch where

$$\pi_0^*(s) \in \operatorname{argmax}_{a \in A} \left\{ r_0(s, a) + \sum_{s'} p_0(s' | s, a) V_{N-1}^*(s') \right\}$$

and

$$V_N^*(s) = \max_{a \in A} \left\{ r_0(s, a) + \sum_{s'} p_0(s' | s, a) V_{N-1}^*(s') \right\}.$$

This finally leads to the following theorem.

THEOREM 1.1 (optimality equation for the finite criterion). *Let $N < \infty$. The optimal value functions $V^* = (V_N^*, \dots, V_1^*)$ are the unique solution to the set of equations:*

$$\forall s \in S \quad V_{n+1}^*(s) = \max_{a \in A} \left\{ r_{N-1-n}(s, a) + \sum_{s'} p_{N-1-n}(s' | s, a) V_n^*(s') \right\} \quad (1.1)$$

with $n = 0, \dots, N - 1$ and $V_0 = 0$. Then, optimal policies $\pi^* = (\pi_0^*, \pi_1^*, \dots, \pi_{N-1}^*)$ for the finite criterion are characterized as

$$\forall s \in S \quad \pi_t^*(s) \in \operatorname{argmax}_{a \in A} \left\{ r_t(s, a) + \sum_{s'} p_t(s' | s, a) V_{N-1-t}^*(s') \right\}$$

for $t = 0, \dots, N - 1$.

It is worth noting that in the finite criterion's case, optimal policies are deterministic and Markov, but non-stationary (the best action choice depends on the decision epoch t).

1.5.1.2. Evaluation of a deterministic Markov policy

Let π be a deterministic, Markov policy. The same procedure as above allows us to define its value function V_N^π .

THEOREM 1.2 (characterization of a policy's N -step value function V_N). *Let $N < \infty$ be an integer and $\pi = (\pi_0, \pi_1, \dots, \pi_{N-1})$ a Markov policy. Then $V_N^\pi = V_N$, with $(V_N, V_{N-1}, \dots, V_1)$ being the solution to the set of linear equations,*

$$\begin{aligned} \forall s \in S, \quad V_{n+1}(s) &= r_{N-1-n}(s, \pi_{N-1-n}(s)) \\ &\quad + \sum_{s'} p_{N-1-n}(s' | s, \pi_{N-1-n}(s)) V_n(s') \end{aligned}$$

for $n = 0, \dots, N-1$ and $V_0 = 0$.

1.5.2. The discounted criterion

The discounted criterion is the most commonly used criterion in infinite horizon frameworks. It allows a rather simple characterization of the optimal value function and its associated policies. Recall that the MDP is still supposed stationary.

1.5.2.1. Evaluation of a stationary Markov policy

Given a stationary Markov policy $\pi \in \mathcal{D}^A$, we can define the L_π operator, mapping elements of \mathcal{V} into \mathcal{V} . Recall that \mathcal{V} is a topological vector space using the supremum norm: $\forall V \in \mathcal{V}, \|V\| = \max_{s \in S} |V(s)|$.

DEFINITION 1.6 (operator L_π). *For any $\pi \in \mathcal{D}^A$,*

$$\forall V \in \mathcal{V} \quad L_\pi V = r_\pi + \gamma P_\pi V.$$

A first statement allows us to link the value function V_γ^π of any stationary policy $\pi \in \mathcal{D}^A$ with this operator.

THEOREM 1.3 (characterization of V_γ^π). *Let $\gamma < 1$ and let $\pi \in \mathcal{D}^A$ be a Markov stationary policy. Then V_γ^π is the only solution of equation $V = L_\pi V$:*

$$\forall s \in S \quad V(s) = r_\pi(s) + \gamma \sum_{s' \in S} P_{\pi, s, s'} V(s') \tag{1.2}$$

and $V_\gamma^\pi = (I - \gamma P_\pi)^{-1} r_\pi$.

Proof. Let V be the solution of $V = L_\pi V$. Then we have $(I - \gamma P_\pi)V = r_\pi$. Matrix P_π being a stochastic matrix, all the eigenvalues of γP_π have an absolute value lower than $\gamma < 1$, so matrix $I - \gamma P_\pi$ is invertible. Consequently,

$$(I - \gamma P_\pi)^{-1} = \sum_{k=0}^{\infty} \gamma^k P_\pi^k$$

and

$$V = (I - \gamma P_\pi)^{-1} r_\pi = \sum_{k=0}^{\infty} \gamma^k P_\pi^k r_\pi.$$

Yet

$$\begin{aligned} \forall s \in S \quad V_\gamma^\pi(s) &= E^\pi[r_0 + \gamma r_1 + \gamma^2 r_2 + \dots + \gamma^t r_t + \dots \mid s_0 = s] \\ &= \sum_{t=0}^{\infty} \gamma^t E^\pi[r(s_t, a_t) \mid s_0 = s] \\ &= \sum_{t=0}^{\infty} \gamma^t \sum_{s' \in S} \sum_{a \in A} P^\pi(s_t = s', a_t = a \mid s_0 = s) r(s', a) \\ &= \sum_{t=0}^{\infty} \gamma^t \sum_{s' \in S} \sum_{a \in A} q(a, s') P^\pi(s_t = s' \mid s_0 = s) r(s', a) \\ &= \sum_{t=0}^{\infty} \gamma^t \sum_{s' \in S} P^\pi(s_t = s' \mid s_0 = s) r_\pi(s') \\ &= \sum_{t=0}^{\infty} \gamma^t \sum_{s' \in S} P_{\pi, s, s'}^t r_\pi(s') \\ &= \sum_{t=0}^{\infty} \gamma^t P_\pi^t r_\pi(s), \end{aligned}$$

and finally $V = V_\gamma^\pi$. □

1.5.2.2. Optimality equations

Remember that we try to solve the optimization problem $\forall s \in S, V_\gamma^*(s) = \max_{\pi \in \Pi^{HA}} V_\gamma^\pi(s)$. A policy π^* is said to be optimal if it verifies $V_\gamma^{\pi^*} = V_\gamma^*$. According to Proposition 1.1, we have

$$\forall s \in S \quad V_\gamma^*(s) = \max_{\pi \in \Pi^{HA}} V_\gamma^\pi(s) = \max_{\pi \in \Pi^{MA}} V_\gamma^\pi(s).$$

Let us now introduce the L operator, called the *dynamic programming operator* and mapping the set of value functions to itself.

DEFINITION 1.7 (operator L).

$$\forall V \in \mathcal{V} \forall s \in S, \quad LV(s) = \max_{a \in A} \left(r(s, a) + \gamma \sum_{s' \in S} p(s' | s, a) V(s') \right)$$

or, in matrix notation,

$$\forall V \in \mathcal{V}, \quad LV = \max_{\pi \in \mathcal{D}} (r_\pi + \gamma P_\pi V).$$

Then we can state the discounted criterion's main theorem concerning value function optimality.

THEOREM 1.4 (Bellman equation). *Let $\gamma < 1$. V_γ^* is then the only solution of equation $V = LV$:*

$$\forall s \in S, \quad V(s) = \max_{a \in A} \left(r(s, a) + \gamma \sum_{s' \in S} p(s' | s, a) V(s') \right). \quad (1.3)$$

Proof. The proof of this theorem is a little long and takes several steps. For the sake of clarity, we start by summarizing these steps here.

1) First, we prove that for any value function V , the stationary Markov policies that maximize the one step expected reward have the same value, whether they are deterministic or stochastic. In other words,

$$\max_{\pi \in \mathcal{D}} (r_\pi + \gamma P_\pi V) = \max_{\pi \in \mathcal{D}^A} (r_\pi + \gamma P_\pi V).$$

This allows us to work directly with stochastic policies in the rest of the proof.

- 2) Then we prove that for all $V \in \mathcal{V}$, if $V \geq LV$, then $V \geq V_\gamma^*$.
- 3) Vice versa, if $V \leq LV$, then $V \leq V_\gamma^*$.
- 4) Hence, if any solution to $V = LV$ exists, it must be equal to V_γ^* .
- 5) Finally we prove that such a solution necessarily exists because L is a contraction mapping over \mathcal{V} .

We first prove that $\forall V \in \mathcal{V}$ and for $0 \leq \gamma \leq 1$,

$$LV = \max_{\pi \in \mathcal{D}} (r_\pi + \gamma P_\pi V) = \max_{\pi \in \mathcal{D}^A} (r_\pi + \gamma P_\pi V).$$

For this purpose, let us consider a value function V and $\delta \in \mathcal{D}^A$. Because $\delta(a, s)$ is positive for all s , we have

$$\begin{aligned} & \sum_a \delta(a, s) \left(r(s, a) + \gamma \sum_{s' \in S} p(s' | s, a) V(s') \right) \\ & \leq \sum_a \delta(a, s) \max_{a'} \left(r(s, a') + \gamma \sum_{s' \in S} p(s' | s, a') V(s') \right) \\ & \leq \sum_a \delta(a, s) LV(s) \\ & \leq LV(s). \end{aligned}$$

Thus, for all $\delta \in \mathcal{D}^A$,

$$r_\delta + \gamma P_\delta V \leq \max_{\pi \in \mathcal{D}} (r_\pi + \gamma P_\pi V)$$

and

$$\max_{\delta \in \mathcal{D}^A} (r_\delta + \gamma P_\delta V) \leq \max_{\pi \in \mathcal{D}} (r_\pi + \gamma P_\pi V).$$

The reverse inequality is immediate because $\mathcal{D} \subset \mathcal{D}^A$.

Then, we can show that $\forall V \in \mathcal{V}, V \geq LV \Rightarrow V \geq V_\gamma^*$ and $V \leq LV \Rightarrow V \leq V_\gamma^*$.

Let V be a value function such that $V \geq LV$. We have

$$V \geq \max_{\pi \in \mathcal{D}} \{r_\pi + \gamma P_\pi V\} = \max_{\pi \in \mathcal{D}^A} \{r_\pi + \gamma P_\pi V\}.$$

Consider $\pi = (\pi_0, \pi_1, \dots) \in \Pi^{MA}$. For all t , policy π_t belongs to \mathcal{D}^A , so

$$\begin{aligned} V & \geq r_{\pi_0} + \gamma P_{\pi_0} V \\ & \geq r_{\pi_0} + \gamma P_{\pi_0} (r_{\pi_1} + \gamma P_{\pi_1} V) \\ & \geq r_{\pi_0} + \gamma P_{\pi_0} r_{\pi_1} + \gamma^2 P_{\pi_0} P_{\pi_1} r_{\pi_2} + \cdots + \gamma^{n-1} P_{\pi_0} \cdots P_{\pi_{n-2}} r_{\pi_{n-1}} + \gamma^n P_\pi^n V. \end{aligned}$$

And so

$$V - V_\gamma^\pi \geq \gamma^n P_\pi^n V - \sum_{k=n}^{\infty} \gamma^k P_\pi^k r_{\pi_k}, \quad \text{since } V_\gamma^\pi = \sum_{k=0}^{\infty} \gamma^k P_\pi^k r_{\pi_k},$$

with $P_\pi^k = P_{\pi_0}P_{\pi_1}\cdots P_{\pi_{k-1}}$. The two right-hand side terms can be made as small as required by taking a large enough value for n , since $\|\gamma^n P_\pi^n V\| \leq \gamma^n \|V\|$ and $\|\sum_{k=n}^{\infty} \gamma^k P_\pi^k r_{\pi_k}\| \leq \sum_{k=n}^{\infty} \gamma^k R \leq \frac{\gamma^n}{1-\gamma} R$ with $R = \max_{s,a} r(s, a)$. Consequently,

$$V \geq \max_{\pi \in \Pi^{MA}} V_\gamma^\pi = \max_{\pi \in \Pi^{HA}} V_\gamma^\pi = V_\gamma^*.$$

Vice versa, consider V such that $V \leq LV$. Then $V \leq \max_{\pi \in \mathcal{D}} \{r_\pi + \gamma P_\pi V\}$. Suppose this max is reached for policy π^* . Then we can write

$$\begin{aligned} V &\leq r_{\pi^*} + \gamma P_{\pi^*} V \\ &\leq r_{\pi^*} + \gamma P_{\pi^*} (r_{\pi^*} + \gamma P_{\pi^*} V) \\ &\leq r_{\pi^*} + \gamma P_{\pi^*} r_{\pi^*} + \cdots + \gamma^{n-1} P_{\pi^*}^{n-1} r_{\pi^*} + \gamma^n P_{\pi^*}^n V \\ V - V_\gamma^{\pi^*} &\leq - \sum_{k=n}^{\infty} \gamma^k P_{\pi^*}^k r_{\pi^*} + \gamma^n P_{\pi^*}^n V. \end{aligned}$$

The right-hand side terms can be made as close to zero as necessary, so $V - V_\gamma^{\pi^*} \leq 0$ and $V \leq V_\gamma^{\pi^*} \leq V_\gamma^*$.

This way, we have shown that if $V \geq LV$, then $V \geq V_\gamma^*$ and if $V \leq LV$, then $V \leq V_\gamma^*$. This implies $V = LV \Rightarrow V = V_\gamma^*$: any solution to $LV = V$ is necessarily equal to the optimal value function V_γ^* .

We finally need to prove that such a solution exists. For this purpose, let us recall Banach fixed-point theorem.

THEOREM 1.5 (Banach fixed-point theorem). *Let \mathcal{U} be a Banach space (i.e. a complete, normed, vector space) and T a contraction mapping on \mathcal{U} (i.e. $\exists 0 \leq \lambda < 1$ such that $\forall (u, v) \in \mathcal{U}^2$, $\|Tu - Tv\| \leq \lambda \|u - v\|$). Then*

- 1) *there exists a single $u^* \in \mathcal{U}$ such that $Tu^* = u^*$;*
- 2) *for all $u_0 \in \mathcal{U}$, the sequence $(u_n)_{n \geq 0}$ defined by $u_{n+1} = Tu_n = T^{n+1}u_0$ converges towards u^* .*

The set \mathcal{V} , equipped with its supremum norm, is a finite-dimensional, normed vector space and, hence, is a complete vector space. So all we need to prove is that the L operator is a contraction mapping for this norm.

PROPOSITION 1.2. *For a discount factor $\gamma < 1$, the dynamic programming operator L defined by $LV = \max_{\pi \in \mathcal{D}} (r_\pi + \gamma P_\pi V)$ is a contraction mapping on \mathcal{V} .*

Proof. Let U and V be two value functions in \mathcal{V} and $s \in S$.

Suppose $LV(s) \geq LU(s)$. Consider a_s^* such that

$$a_s^* \in \operatorname{argmax}_{a \in A} \left(r(s, a) + \gamma \sum_{s' \in S} p(s' | s, a) V(s') \right).$$

Then we can write

$$\begin{aligned} 0 &\leq |LV(s) - LU(s)| = LV(s) - LU(s) \\ &\leq r(s, a_s^*) + \gamma \sum_{s' \in S} p(s' | s, a_s^*) V(s') - r(s, a_s^*) - \gamma \sum_{s' \in S} p(s' | s, a_s^*) U(s') \\ &\leq \gamma \sum_{s' \in S} p(s' | s, a_s^*) (V(s') - U(s')) \\ &\leq \gamma \sum_{s' \in S} p(s' | s, a_s^*) \|V - U\| \\ &\leq \gamma \|V - U\|. \end{aligned}$$

Consequently,

$$\|LV - LU\| = \max_s |LV(s) - LU(s)| \leq \gamma \|V - U\|. \quad \square$$

Finally, this contraction property guarantees the existence of a fixed point for operator L . Because of the above, this fixed point is unique and equal to V_γ^* . \square

We can complete the study of optimal value functions for the discounted criterion with this last theorem.

THEOREM 1.6 (characterization of optimal policies). *Let $\gamma < 1$. We have:*

1) $\pi^* \in \Pi^{HA}$ is an optimal policy $\Leftrightarrow V_\gamma^{\pi^*}$ is a solution of $LV = V$ and $V_\gamma^{\pi^*} = V_\gamma^*$;

2) any stationary policy π^* verifying $\pi^* \in \operatorname{argmax}_{\pi \in \mathcal{D}} \{r_\pi + \gamma P_\pi V_\gamma^*\}$ is an optimal policy.

Proof. The first equivalence is immediate because of the previous theorem. Consider $\pi^* \in \operatorname{argmax}_{\pi \in \mathcal{D}} \{r_\pi + \gamma P_\pi V_\gamma^*\}$. Then we can write

$$\begin{aligned}
L_{\pi^*} V_\gamma^* &= r_{\pi^*} + \gamma P_{\pi^*} V_\gamma^* \\
&= \max_{\pi \in \mathcal{D}} \{r_\pi + \gamma P_\pi V_\gamma^*\} \\
&= LV_\gamma^* \\
&= V_\gamma^*.
\end{aligned}$$

The previous theorem guaranteed the solution of $V = L_{\pi^*} V$ to be unique. Thus $V_\gamma^* = V_\gamma^{\pi^*}$ and so, finally, π^* is optimal. \square

1.5.3. The total reward criterion

The total reward criterion has a major mathematical drawback: it is based on a limit (when the horizon tends to infinity), but the existence of this limit can only be guaranteed under some specific assumptions. We will consider here two classes of problems for which this limit necessarily exists and is finite for at least one policy: namely, positive and negative models.

DEFINITION 1.8. Let $\pi \in \Pi^{HA}$. We can define functions V_+^π and V_-^π as

$$\begin{aligned}
V_+^\pi(s) &= E^\pi \left[\sum_{t=0}^{\infty} \max(r_t, 0) \mid s_0 = s \right], \\
V_-^\pi(s) &= E^\pi \left[\sum_{t=0}^{\infty} \max(-r_t, 0) \mid s_0 = s \right].
\end{aligned}$$

Let us, respectively, write \mathcal{V}^+ and \mathcal{V}^- as the sets of positive and negative functions of \mathcal{V} .

Assuming that for any policy π and any state s , $V_+^\pi(s)$ or $V_-^\pi(s)$ is finite, then we can guarantee the existence of a limit (finite or infinite) for V^π , written as

$$\forall s \in S, \quad V^\pi(s) = V_+^\pi(s) - V_-^\pi(s).$$

Positively bounded MDPs, also called positive MDPs, verify:

- for all s , there exists at least one action $a \in A$ such that $r(s, a) \geq 0$,
- $V_+^\pi(s) < \infty$ for all $s \in S$ and for all $\pi \in \Pi^{HA}$.

Negative MDPs verify:

- $r(s, a) \leq 0$ for all $s \in S$ and $a \in A$,

- there exists $\pi \in \Pi^{HA}$ such that $V^\pi(s) > -\infty$ for all $s \in S$.

The existence of a policy π , for which $V_\pi(s)$ is finite for all $s \in S$, can typically be ensured by the presence of a reachable absorbing state s_∞ with null reward:

$$\forall s_0, \quad P^\pi(\exists t^* s_{t^*} = s_\infty) = 1,$$

with

$$p(s_\infty | s_\infty, \pi(s_\infty)) = 1, \quad \text{and} \quad r(s_\infty, \pi(s_\infty)) = 0.$$

In a positive model, an optimal policy has the largest possible positive total reward. The agent tries to extend its trajectories as much as possible to gather positive rewards. For a negative model, an optimal policy has the largest possible negative reward (i.e. as close as possible to zero). The agent thus tries to terminate trajectories as fast as possible to minimize its loss. Consequently, the two models are not completely symmetric.

The following results provide important properties of positive and negative models. To present them we first need to introduce a new definition of the L^π and L operators.

DEFINITION 1.9 (operators L_π and L for the total reward criterion). *Let π be a stationary policy in \mathcal{D}^A :*

$$\forall V \in \mathcal{V}, \quad L_\pi V = r_\pi + P_\pi V$$

and

$$\forall V \in \mathcal{V}, \quad LV = \max_{\pi \in \mathcal{D}} (r_\pi + P_\pi V).$$

Then we can prove, for positive and negative MDPs, the following.

THEOREM 1.7 [PUT 94]. *Consider a positive MDP:*

- 1) for all stationary policy $\pi \in \mathcal{D}$, V^π is the minimum solution of $V = L_\pi V$ among the set \mathcal{V}^+ ;
- 2) V^* is the minimum solution of equation $V = LV$ in \mathcal{V}^+ ;
- 3) a policy $\pi^* \in \Pi^{HA}$ is optimal $\Leftrightarrow V^{\pi^*} = LV^{\pi^*}$;
- 4) if $\pi^* \in \operatorname{argmax}_{\pi \in \mathcal{D}} (r_\pi + P_\pi V^*)$ and if $\lim_{N \rightarrow \infty} P_{\pi^*}^N V^*(s) = 0$ for all $s \in S$, then π^* is optimal.

THEOREM 1.8 [PUT 94]. *Consider a negative MDP:*

- 1) for all stationary policy $\pi \in \mathcal{D}$, V^π is the maximal solution of equation $V = L_\pi V$ among the set \mathcal{V}^- ;

- 2) V^* is the maximal solution of $V = LV$ in \mathcal{V}^- ;
- 3) every policy verifying $\pi^* \in \operatorname{argmax}_{\pi \in \mathcal{D}}(r_\pi + P_\pi V^*)$ is an optimal policy.

Note that for a negative MDP, we can find a policy π , such that $V^\pi = LV^\pi$, which is not an optimal policy (see [PUT 94, Example 7.3.1]).

1.5.4. The average reward criterion

The average reward criterion's theoretical analysis is more complex than for the other criteria. It deals with the underlying valued Markov process' behavior in the limit. In this section, we will restrict ourselves to the main results for the cases of:

- recurrent MDPs (for any deterministic Markov policy, the corresponding Markov chain is made up of a unique recurrent class);
- unichain MDPs (the corresponding Markov chain is made of a unique recurrent class plus some transitory states);
- multichain MDPs (there exists at least one policy such that the corresponding Markov chain has two or more irreducible recurrent classes).

Additionally we will assume that, for all policies, the corresponding Markov chain is aperiodic.

1.5.4.1. Evaluation of a stationary Markov policy

Let $\pi \in \mathcal{D}^A$ be a stationary policy and (S, P_π, r_π) the associated valued Markov process. Recall that the average reward criterion is defined as

$$\forall s \in S, \quad \rho^\pi(s) = \lim_{N \rightarrow \infty} E^\pi \left[\frac{1}{N} \sum_{t=0}^{N-1} r_\pi(s_t) \mid s_0 = s \right].$$

In matrix form, we have

$$\rho^\pi = \lim_{N \rightarrow \infty} \frac{1}{N} \sum_{t=0}^{N-1} P_\pi^t r_\pi.$$

Let $P_\pi^* = \lim_{N \rightarrow \infty} \frac{1}{N} \sum_{t=0}^{N-1} P_\pi^t$ be the *limiting matrix* of P_π . We can show that P_π^* exists, and is a stochastic matrix for all finite state spaces S . Moreover, P_π^* verifies

$$P_\pi P_\pi^* = P_\pi^* P_\pi = P_\pi^* P_\pi^* = P_\pi^*.$$

The $P_{\pi,s,s'}^*$ coefficient can be interpreted as the amount of time the system will spend in state s' after leaving state s . Moreover, for aperiodic MDPs, we have $P_\pi^* = \lim_{N \rightarrow \infty} P_\pi^N$ and $P_{\pi,s,s'}^*$ can be seen as the probability of ending up in state s' after leaving state s . Finally, for a unichain MDP, the P_π^* has all its rows equal to each other and to the *stationary distribution* μ_π of the Markov process controlled by policy π . Hence $\forall s, s', P_{\pi,s,s'}^* = \mu_\pi(s')$.

From the previous definition of ρ^π , we can conclude that $\rho^\pi = P_\pi^* r_\pi$. Thus, for a unichain MDP, we can write that $\rho(s) = \rho$ is constant for all s , with

$$\rho = \sum_{s \in S} \mu_\pi(s) r_\pi(s).$$

In the more general case of a multichain MDP, $\rho(s)$ is constant for each recurrent class.

This first characterization of ρ_π involves the P_π^* matrix. But this matrix is hard to calculate. However, we can find ρ_π differently, by introducing a new value function for the average reward criterion. This value function is called *relative value function*.

DEFINITION 1.10 (relative value function for the average reward criterion).

$$\forall s \in S, \quad U^\pi(s) = E^\pi \left[\sum_{t=0}^{\infty} (r_t - \rho^\pi) \mid s_0 = s \right].$$

In vector notation, we have

$$\begin{aligned} U^\pi &= \sum_{t=0}^{\infty} P_\pi^t (r_\pi - \rho^\pi) \\ &= \sum_{t=0}^{\infty} P_\pi^t (r_\pi - P_\pi^* r_\pi) \\ &= \sum_{t=0}^{\infty} (P_\pi^t - P_\pi^*) r_\pi \\ &= \left(I - P_\pi^* + \sum_{t=1}^{\infty} (P_\pi - P_\pi^*)^t \right) r_\pi \\ &= \left(-P_\pi^* + (I - P_\pi + P_\pi^*)^{-1} \right) r_\pi. \end{aligned}$$

This is true because the $(I - P_\pi + P_\pi^*)$ matrix is invertible and because for all $t > 0$, $(P_\pi - P_\pi^*)^t = P_\pi^t - P_\pi^*$. Multiplying this formula by $(I - P_\pi + P_\pi^*)$ yields

$$\begin{aligned} U^\pi &= (I - P_\pi + P_\pi^*)^{-1} (I - P_\pi^* (I - P_\pi + P_\pi^*)) r_\pi \\ &= (I - P_\pi + P_\pi^*)^{-1} (I - P_\pi^*) r_\pi. \end{aligned}$$

We usually write $H_{P_\pi} = (I - P_\pi + P_\pi^*)^{-1} (I - P_\pi^*)$ the *deviation matrix* of P_π . This matrix is such that

$$U^\pi = H_{P_\pi} r_\pi.$$

Then, H_{P_π} is the pseudo-inverse of $(I - P_\pi)$, which makes the link between this definition of U^π and the original expression of V_γ^π established in Theorem 1.3.

The following result holds for any valued Markov process (recurrent, unichain or multichain).

THEOREM 1.9 [PUT 94]. *Let (S, P_π, r_π) be the valued Markov process associated with a stationary policy $\pi \in \mathcal{D}^A$. We can state that:*

- 1) if ρ^π and U^π are, respectively, the average reward and the relative value function of π :
 - a) $(I - P_\pi)\rho^\pi = 0$,
 - b) $\rho^\pi + (I - P_\pi)U^\pi = r_\pi$;
- 2) vice versa, if ρ and U verify the two previous equalities, then:
 - a) $\rho = P_\pi^* r_\pi = \rho^\pi$,
 - b) $U = H_{P_\pi} r_\pi + u$, where $(I - P_\pi)u = 0$,
 - c) furthermore, if $P_\pi^* U = 0$, then $U = H_{P_\pi} r_\pi = U^\pi$.

It is important to note that the relative value function U^π is the unique solution of $(I - P_\pi)U = (I - P_\pi^*)r_\pi$ such that $P_\pi^* U = 0$. This value function is obtained by forming the pseudo-inverse H_{P_π} of $(I - P_\pi)$.

In the case of a unichain process, the first equation above can be simplified into $\rho_\pi(s) = \rho_\pi$, and the second one can be written as

$$\forall s \in S \quad U(s) + \rho = r_\pi(s) + \sum_{s' \in S} P_{\pi,s,s'} U(s'). \quad (1.4)$$

Then, any solution (ρ, U) to this equation verifies $\rho = \rho_\pi$ and $U = U_\pi + ke$ where k is a real number and e is the vector with all elements equal to one. Furthermore, if $\sum_{s \in S} \mu_\pi(s)U(s) = 0$, then $U = U_\pi$. It is relevant to compare this final equation to the optimality equation for the discounted criterion.

1.5.4.2. Optimality equations

We can now state the optimality conditions for the average reward criterion. Recall that the goal is to find policies $\pi^* \in \Pi^{HA}$ such that

$$\rho_{\pi^*} = \max_{\pi \in \Pi^{HA}} \rho_{\pi} = \rho^*.$$

The main result concerning multichain MDPs is the following.

THEOREM 1.10 [PUT 94] (multichain optimality equation). *There exists a solution (ρ, U) to the set of equations defined, for all $s \in S$, as*

$$\begin{aligned} \rho(s) &= \max_{a \in A} \sum_{s' \in S} p(s' | s, a) \rho(s'), \\ U(s) + \rho(s) &= \max_{a \in B_s} \left(r(s, a) + \sum_{s' \in S} p(s' | s, a) U(s') \right), \end{aligned}$$

with

$$B_s = \left\{ a \in A \mid \sum_{s' \in S} p(s' | s, a) \rho(s') = \rho(s) \right\}.$$

Then we have $\rho = \rho^*$.

In the unichain case, ρ is constant and $B_s = A$. The optimality equations can be simplified into

$$\forall s \in S \quad U(s) + \rho = \max_{a \in A} \left(r(s, a) + \sum_{s' \in S} p(s' | s, a) U(s') \right). \quad (1.5)$$

Linking the optimality equation solutions to the optimal policies is done through the following theorem.

THEOREM 1.11 [PUT 94]. *Let (ρ, U) be a solution to the optimality equation. Then there exists a gain-optimal, deterministic, stationary, Markov policy $\pi^* \in \mathcal{D}$, characterized by*

$$\forall s \in S \quad \pi^*(s) \in \operatorname{argmax}_{a \in B_s} \left(r(s, a) + \sum_{s' \in S} p(s' | s, a) U(s') \right).$$

Note that there can exist gain-optimal policies π^* such that $(\rho^{\pi^*}, U^{\pi^*})$ do not respect the optimality equation.

Finally, it is noticeable that the optimality equations solutions are not unique; if (ρ^*, U) is a solution, then $(\rho^*, U + ke)$ also is, for any real number k . In particular, there can be several relative value functions which are solutions, all defining different policies, but all corresponding to the same optimal ρ^* . Then it is useful to search, among these different solutions, for the ones that maximize the relative value function. In this last case, we talk about *bias-optimality*.

1.6. Optimization algorithms for MDPs

1.6.1. The finite criterion

This is a rather simple case. The optimality equations make it possible to recursively calculate the optimal value functions V_1^*, \dots, V_N^* from the last stage to the first. This process is described in Algorithm 1.1.

Algorithm 1.1: Finite horizon dynamic programming

```

 $V_0 \leftarrow 0$ 
for  $n \leftarrow 0$  to  $N - 1$  do
    for  $s \in S$  do
         $V_{n+1}^*(s) = \max_{a \in A} \left\{ r_{N-1-n}(s, a) + \sum_{s'} p_{N-1-n}(s' | s, a) V_n^*(s') \right\}$ 
         $\pi_{N-1-n}(s) \in \operatorname{argmax}_{a \in A} \left\{ r_{N-1-n}(s, a) + \sum_{s'} p_{N-1-n}(s' | s, a) V_n^*(s') \right\}$ 
    return  $V^*, \pi^*$ 

```

The space and time complexity of this algorithm is in $O(N|S|^2|A|)$.

1.6.2. The discounted criterion

There are three main classes of methods designed to solve Markov decision problems with a discounted criterion: linear programming, value iteration and policy iteration. All three search for optimal policies in \mathcal{D} .

1.6.2.1. Linear programming

We can immediately check that if $V \in \mathcal{V}$ minimizes $\sum_{s \in S} V(s)$ under the constraint $V \geq LV$, then $V = V_\gamma^*$. Indeed, it was shown in the proof of Theorem 1.4

that $V \geq LV$ implied $V \geq V_\gamma^*$. So $\sum_{s \in S} V(s) \geq \sum_{s \in S} V_\gamma^*(s)$. Consequently, we can search for the optimal value function V_γ^* by solving the associated linear program, as shown in Algorithm 1.2.

Algorithm 1.2: Linear programming for the discounted criterion

Solve

$$\min_{V \in \mathcal{V}} \sum_{s \in S} V(s)$$

with

$$V(s) \geq r(s, a) + \gamma \sum_{s' \in S} p(s' | s, a)V(s'), \quad \forall s \in S, a \in A$$

```

for  $s \in S$  do
   $\pi(s) \in \operatorname{argmax}_{a \in A} \left\{ r(s, a) + \gamma \sum_{s'} p(s' | s, a)V(s') \right\}$ 
return  $V, \pi$ 

```

This approach was first introduced by [D'E 63]. If S and A have, respectively, sizes n and m , and if $p()$ and $r()$ are coded over b bits, then the complexity of such a linear programming method over rational numbers is polynomial in $|S|$, $|A|$, and b , with rather long resolution times [LIT 95]. However, Chapter 4 will show that such methods can still be efficient in the framework of factored MDPs.

1.6.2.2. The value iteration algorithm

Value Iteration is the most common method used to solve MDPs. It relies on the direct resolution of the Bellman optimality equation $V = LV$. For this purpose, it uses the contraction property discussed previously and builds a fixed-point, iterative procedure over value functions, hence its name of *value iteration* [BEL 57, BER 87, PUT 94].

As proven by Theorem 1.4, the solution to the optimality equation can be obtained as the limit of the $V_{n+1} = LV_n$ sequence, for any initialization V_0 . We can then show that a maximum number of iterations which is polynomial in $|S|$, $|A|$, b , $1/(1 - \gamma) \log(1/(1 - \gamma))$ is necessary to reach a greedy policy corresponding to π^* . Each of these iterations has complexity $O(|A||S|^2)$ [PAP 87]. Beyond this number of iterations, the V_n sequence becomes closer and closer to V^* but the corresponding greedy policy does not change anymore.

In practice, several stopping criteria can be considered to cease the iterations. The most common one consists of stopping whenever $\|V_{n+1} - V_n\| < \epsilon$, where ϵ is an *a priori* tolerance threshold. This leads to Algorithm 1.3.

Algorithm 1.3: Value iteration algorithm – discounted criterion

```

Initialize  $V_0 \in \mathcal{V}$ 
 $n \leftarrow 0$ 
repeat
  for  $s \in S$  do
    
$$V_{n+1}(s) = \max_{a \in A} \left\{ r(s, a) + \gamma \sum_{s' \in S} p(s' | s, a) V_n(s') \right\}$$

     $n \leftarrow n + 1$ 
  until  $\|V_{n+1} - V_n\| < \epsilon$ 
  for  $s \in S$  do
    
$$\pi(s) \in \operatorname{argmax}_{a \in A} \left\{ r(s, a) + \gamma \sum_{s'} p(s' | s, a) V_n(s') \right\}$$

return  $V_n, \pi$ 
```

At the last iteration n , we have $\|V_n - V_\gamma^*\| < \epsilon'$ with $\epsilon' = \frac{2\gamma}{1-\gamma}\epsilon$ (see Chapter 3 for more details).

We can improve the algorithm's convergence velocity by slightly modifying the way V_{n+1} is calculated. The idea consists of using $V_{n+1}(s)$ instead of $V_n(s)$ when this value is already available. This variant is called the Gauss-Seidel algorithm. Numbering the states of S from 1 to $|S|$, we obtain Algorithm 1.4.

This idea can be pushed even further in the case where the states updated at each iteration are selected randomly (or following a given non-static ordering) in S . This leads to the general case of *asynchronous dynamic programming* [BER 89].

Finally, it is also possible to refine the algorithm by pruning sub-optimal actions as early as possible. This reduces the maximization phase's complexity.

1.6.2.3. The policy iteration algorithm

A last main class of resolution algorithms for the discounted criterion deals with methods directly iterating in the space of policies.

Algorithm 1.4: Value iteration algorithm – Gauss-Seidel

```

Initialize  $V_0 \in \mathcal{V}$ 
 $n \leftarrow 0$ 
repeat
  for  $i \leftarrow 1$  to  $|S|$  do
    
$$V_{n+1}(s_i) = \max_{a \in A} \left\{ r(s, a) + \gamma \sum_{1 \leq j < i} p(s_j | s, a) V_{n+1}(s_j) \right.$$

    
$$\left. + \gamma \sum_{i \leq j \leq |S|} p(s_j | s, a) V_n(s_j) \right\}$$

     $n \leftarrow n + 1$ 
  until  $\|V_{n+1} - V_n\| < \epsilon$ 
  for  $s \in S$  do
    
$$\pi(s) \in \operatorname{argmax}_{a \in A} \left\{ r(s, a) + \gamma \sum_{s'} p(s' | s, a) V_n(s') \right\}$$

return  $V_n, \pi$ 

```

Consider a stationary policy $\pi \in \mathcal{D}$ and V_γ^π its value function. The policy iteration algorithm makes use of the following property.

PROPERTY 1.1 (one step look-ahead policy improvement). *Let $\pi \in \mathcal{D}$. Any policy π^+ defined by*

$$\pi^+ \in \operatorname{argmax}_{\delta \in \mathcal{D}} \{r_\delta + \gamma P_\delta V_\gamma^\pi\}$$

verifies

$$V_\gamma^{\pi^+} \geq V_\gamma^\pi$$

with $V_\gamma^{\pi^+} = V_\gamma^\pi \Leftrightarrow \pi = \pi^$.*

Proof. We have

$$\begin{aligned} r_{\pi^+} + \gamma P_{\pi^+} V_\gamma^\pi &= \max_{\delta \in \mathcal{D}} \{r_\delta + \gamma P_\delta V_\gamma^\pi\} \\ &\geq r_\pi + \gamma P_\pi V_\gamma^\pi \\ &\geq V_\gamma^\pi \end{aligned}$$

because $V_\gamma^\pi = r_\pi + \gamma P_\pi V_\gamma^\pi$. So

$$\begin{aligned} r_{\pi^+} + \gamma P_{\pi^+} V_\gamma^{\pi^+} + \gamma P_{\pi^+} (V_\gamma^\pi - V_\gamma^{\pi^+}) &\geq V_\gamma^\pi \\ V_\gamma^{\pi^+} - \gamma P_{\pi^+} V_\gamma^{\pi^+} &\geq V_\gamma^\pi - \gamma P_{\pi^+} V_\gamma^\pi \\ (I - \gamma P_{\pi^+}) V_\gamma^{\pi^+} &\geq (I - \gamma P_{\pi^+}) V_\gamma^\pi \\ V_\gamma^{\pi^+} &\geq V_\gamma^\pi \end{aligned}$$

because if $u \geq v$, $(I - \gamma P_{\pi^+})^{-1}u = u + \gamma P_{\pi^+}u + \gamma^2 P_{\pi^+}^2 u^2 \dots \geq v + \gamma P_{\pi^+}v + \gamma^2 P_{\pi^+}^2 v^2 \dots \geq (I - \gamma P_{\pi^+})^{-1}v$.

The equality is only possible if $\max_{\delta \in \mathcal{D}} \{r_\delta + \gamma P_\delta V_\gamma^\pi\} = V_\gamma^\pi$, i.e. $V_\gamma^\pi = V_\gamma^*$. \square

The policy iteration algorithm is presented in Algorithm 1.5: let π_n be the considered policy at iteration n . First, the linear system $V_n = L_{\pi_n} V_n$ is solved to calculate V^{π_n} . Then, in a second step, the policy is improved by introducing $\pi_{n+1} \in \operatorname{argmax}_{\delta \in \mathcal{D}} \{r_\delta + \gamma P_\delta V_n\}$. The algorithm is stopped whenever $\pi_n = \pi_{n+1}$ (no possible improvements anymore).

The sequence of V_n is a monotonous, increasing sequence of value functions. It is upper-bounded by V^* and converges. Since the number of possible policies is discrete, the sequence of policies π_n converges in a finite number of iterations. After convergence, $V_n = V_\gamma^*$ and π_n is an optimal policy.

Algorithm 1.5: Policy iteration algorithm – discounted criterion

```

Initialize  $\pi_0 \in \mathcal{D}$ 
 $n \leftarrow 0$ 
repeat
  Solve
    
$$V_n(s) = r(s, \pi_n(s)) + \gamma \sum_{s' \in S} p(s' | s, \pi_n(s)) V_n(s'), \quad \forall s \in S$$

    for  $s \in S$  do
      
$$\pi_{n+1}(s) \in \operatorname{argmax}_{a \in A} \left\{ r(s, a) + \gamma \sum_{s' \in S} p(s' | s, a) V_n(s') \right\}$$

   $n \leftarrow n + 1$ 
until  $\pi_n = \pi_{n+1}$ 
return  $V_n, \pi_{n+1}$ 

```

Policy iteration has complexity $O(|A||S|^2) + O(|S|^3)$ per iteration, with a maximum number of iterations polynomial in $|S|, |A|, b$ for a constant γ [PAP 87].

Here also, it is possible to improve the algorithm's efficiency by simplifying the current policy's evaluation phase. One option is to solve iteratively equation $V_n = L_{\pi_n} V_n$, as in value iteration (but without the max operator) with an early stopping in the iterations. This provides an estimate of the current policy's value function and results in the *modified policy iteration* algorithm (Algorithm 1.6).

Algorithm 1.6: Modified policy iteration algorithm – discounted criterion

```

Initialize  $V_0 \in \mathcal{V}$  such that  $LV_0 \geq V_0$ 
 $flag \leftarrow 0$ 
 $n \leftarrow 0$ 
repeat
  for  $s \in S$  do
     $\pi_{n+1}(s) \in \operatorname{argmax}_{a \in A} \left\{ r(s, a) + \gamma \sum_{s' \in S} p(s' | s, a) V_n(s') \right\}$ 
    ( $\pi_{n+1}(s) = \pi_n(s)$  if possible)
     $V_n^0(s) = \max_{a \in A} \left\{ r(s, a) + \gamma \sum_{s' \in S} p(s' | s, a) V_n(s') \right\}$ 
  if  $\|V_n^0 - V_n\| < \epsilon$  then  $flag \leftarrow 1$ 
  otherwise
     $m \leftarrow 0$ 
    repeat
      for  $s \in S$  do
         $V_n^{m+1}(s) = r(s, \pi_{n+1}(s)) + \gamma \sum_{s' \in S} p(s' | s, \pi_{n+1}(s)) V_n^m(s')$ 
       $m \leftarrow m + 1$ 
    until  $\|V_n^{m+1} - V_n^m\| < \delta$ 
     $V_{n+1} \leftarrow V_n^m$ 
     $n \leftarrow n + 1$ 
  until  $flag = 1$ 
return  $V_n, \pi_{n+1}$ 

```

This algorithm combines some characteristics of value and policy iteration. It converges for all values of δ towards an optimal policy, for $\epsilon \rightarrow 0$, under the assumption that $LV_0 \geq V_0$. This condition is verified, for example, when we choose

$$V_0(s) = \frac{1}{1 - \gamma} \min_{s' \in S} \min_{a \in A} r(s', a),$$

for all $s \in S$.

In practice, policy iteration algorithms (especially the modified policy iteration algorithm) appear more efficient than value iteration ones and should be preferred.

1.6.3. The total reward criterion

1.6.3.1. Positive MDPs

For positive models, the value iteration algorithm converges monotonously towards V^* under the assumption that $0 \leq V_0 \leq V^*$.

Concerning the policy iteration algorithm for positive MDPs (Algorithm 1.7), we need to impose an initial condition on V_0 guaranteeing that V_n will remain in \mathcal{V}^+ at every iteration. The computation of V_n can be adapted by forcing the $V_n(s)$ values to zero for all recurrent states in the Markov chain defined by P_{π_n} . Then, this algorithm converges in a finite number of iterations towards V^* and π^* . Similarly, by assuming $LV_0 \geq V_0$ and $V_0 \leq V^*$, we can prove convergence to an optimal policy for the modified policy iteration algorithm. In practice, $V_0 = 0$ is a sufficient condition.

Algorithm 1.7: Policy iteration algorithm – total reward criterion – positive MDPs

```

Initialize  $\pi_0 \in \mathcal{D}$  with  $r_{\pi_0} \geq 0$ 
 $n \leftarrow 0$ 
repeat
    Calculate the minimum solution of
    
$$V_n(s) = r(s, \pi_n(s)) + \sum_{s' \in S} p(s' | s, \pi_n(s))V_n(s'), \quad \forall s \in S$$

    for  $s \in S$  do
        
$$\pi_{n+1}(s) \in \operatorname{argmax}_{a \in A} \left\{ r(s, a) + \sum_{s' \in S} p(s' | s, a)V_n(s') \right\}$$

        ( $\pi_{n+1}(s) = \pi_n(s)$  if possible)
     $n \leftarrow n + 1$ 
until  $\pi_n = \pi_{n+1}$ 
return  $V_n, \pi_{n+1}$ 
```

1.6.3.2. Negative MDPs

The value iteration algorithm converges monotonously towards V^* for any initialization respecting $V^* \leq V_0 \leq 0$. However, policy iteration algorithms do not necessarily converge to optimal policies and can stop on sub-optimal policies.

1.6.4. The average criterion

As for the discounted criterion, the average reward criterion problems can be solved using a number of dynamic programming techniques in order to calculate optimal gain policies. This section will present the two most important ones, in the simplified case of unichain MDPs, for which the average reward ρ is constant. The stopping criterion will be based on the seminorm span over \mathcal{V} : $\forall V \in \mathcal{V}$, $\text{span}(V) = \max_{s \in S} V(s) - \min_{s \in S} V(s)$. Contrarily to $\|V\|$, which measures the gap between V and 0, the seminorm $\text{span}(V)$ measures the gap between V and a constant vector.

1.6.4.1. Relative value iteration algorithm

Algorithm 1.8 is a value iteration algorithm on the relative value function $U(s) = V(s) - \rho$.

Algorithm 1.8: Relative value iteration algorithm – average reward criterion

```

Initialize  $U_0 \in \mathcal{V}$ 
Choose  $s^* \in S$ 
 $n \leftarrow 0$ 
repeat
   $\rho_{n+1} = \max_{a \in A} \left\{ r(s^*, a) + \sum_{s' \in S} p(s' | s^*, a) U_n(s') \right\}$ 
  for  $s \in S$  do
     $U_{n+1}(s) = \max_{a \in A} \left\{ r(s, a) + \sum_{s' \in S} p(s' | s, a) U_n(s') \right\} - \rho_{n+1}$ 
     $n \leftarrow n + 1$ 
  until  $\text{span}(U_{n+1} - U_n) < \epsilon$ 
  for  $s \in S$  do
     $\pi(s) \in \operatorname{argmax}_{a \in A} \left\{ r(s, a) + \sum_{s'} p(s' | s, a) U_n(s') \right\}$ 
return  $\rho_n, U_n, \pi$ 

```

Under several technical hypotheses, for any $\epsilon \rightarrow 0$, we can prove that this algorithm converges to an optimal solution (ρ^*, V^*) respecting the optimality equation (equation (1.5)). Consequently, the relative value iteration algorithm converges to an optimal policy π^* (see [PUT 94, Theorem 8.5.3] for a more detailed discussion). For a unichain MDP, we can guarantee this convergence if $p(s | s, a) > 0$ for all s and a .

1.6.4.2. *Modified policy iteration algorithm*

Algorithm 1.9 is a modified policy iteration algorithm. It does not rely on the resolution of equation (1.4) to evaluate the relative value function.

Algorithm 1.9: Modified policy iteration algorithm – average reward criterion

```

Initialize  $V_0 \in \mathcal{V}$ 
 $flag \leftarrow 0$ 
 $n \leftarrow 0$ 
repeat
  for  $s \in S$  do
     $\pi_{n+1}(s) \in \operatorname{argmax}_{a \in A} \left\{ r(s, a) + \sum_{s' \in S} p(s' | s, a) V_n(s') \right\}$ 
    ( $\pi_{n+1}(s) = \pi_n(s)$  if possible)
     $V_n^0(s) = \max_{a \in A} \left\{ r(s, a) + \sum_{s' \in S} p(s' | s, a) V_n(s') \right\}$ 
  if  $\text{span}(V_n^0 - V_n) < \epsilon$  then  $flag \leftarrow 1$ 
  otherwise
     $m \leftarrow 0$ 
    repeat
      for  $s \in S$  do
         $V_n^{m+1}(s) = r(s, \pi_{n+1}(s)) + \sum_{s' \in S} p(s' | s, \pi_{n+1}(s)) V_n^m(s')$ 
       $m \leftarrow m + 1$ 
    until  $\text{span}(V_n^{m+1} - V_n^m) < \delta$ 
     $V_{n+1} \leftarrow V_n^m$ 
     $n \leftarrow n + 1$ 
  until  $flag = 1$ 
return  $V_n, \pi_{n+1}$ 

```

For a high value of δ , the algorithm is equivalent to the standard value iteration algorithm (on regular value functions instead of relative ones, since the average reward ρ is not explicitly calculated here). For small values of δ (close to zero), it turns out to be a classical policy iteration method. Under the same technical assumptions as before, we can prove convergence of this algorithm for all values of δ , towards an optimal policy (for all $\epsilon \rightarrow 0$). More specifically, upon termination of the algorithm, we have

$$\min_{s \in S} (V_n^0(s) - V_n(s)) \leq \rho^{\pi_{n+1}} \leq \rho^* \leq \max_{s \in S} (V^0(s) - V_n(s)),$$

Which ensures that $|\rho^{\pi_{n+1}} - \rho^*| \leq \epsilon$.

1.7. Conclusion and outlook

This chapter was dedicated to introducing the framework of Markov decision processes, with its decision model, optimality criteria and optimization algorithms. This framework has become, in recent years, a premier methodological tool to model, analyze and solve problems of sequential decision under uncertainty in artificial intelligence.

Despite its genericity, the theoretical framework presented in the previous pages also has its limitations, even in formal terms. First of all, it relies on the assumption that the agent has a perfect knowledge of the transition and reward models defining the problem at hand. We will see in Chapter 2 how reinforcement learning relaxes this hypothesis. Then, Markov decision problems assume the agent is perfectly and instantaneously informed of its current state. However, in many real-life cases where an agent interacts with its environment, the agent cannot directly access the state of the environment. Instead, it relies on a set of differentiated perceptions and observations which help him build a non-exhaustive knowledge about the hidden state of its world. Chapter 7 will present a method to extend MDPs in order to formalize partial observability of the world's state. Also, another formal limitation deals with the single-agent character of MDP problems. Many situations require modeling several agents interacting together inside the same environment. We will present extensions to the MDP framework, related to multi-agent modeling, in Chapters 8 and 9. Finally, a last limitation of the MDP model comes from the expression of the optimality criteria as an expectation over a sum of rewards. Chapter 10 will show how we can extend the representations of uncertainty and preferences of a given agent to other formalisms.

Applications of the MDP framework to real-life problems are growing everyday in various domains. From industrial processes management to agro-ecosystems, from robotics to military operations management, the constraints of large, complex problems have drawn researchers to question the efficiency of their resolution methods. Consequently, several chapters of this book will be dedicated to recent methods allowing to overcome the usual limitations of dynamic programming. Among these techniques, we should specifically mention value function approximation methods (Chapter 3), factored representations (Chapter 4), optimization of parametric policies (Chapter 5) or online resolution approaches (Chapter 6).

1.8. Bibliography

- [BEL 57] BELLMAN R. E., *Dynamic Programming*, Princeton University Press, Princeton, NJ, 1957.
- [BER 87] BERTSEKAS D. P., *Dynamic Programming: Deterministic and Stochastic Models*, Prentice-Hall, Englewood Cliffs, NJ, 1987.

- [BER 89] BERTSEKAS D. P. and TSITSIKLIS J. N., *Parallel and Distributed Computation: Numerical Methods*, Prentice-Hall, Englewood Cliffs, NJ, 1989.
- [BER 95] BERTSEKAS D., *Dynamic Programming and Optimal Control*, Athena Scientific, Belmont, MA, 1995.
- [D'E 63] D'EPENOUX F., "A probabilistic production and inventory problem", *Management Science*, vol. 10, pp. 98–108, 1963.
- [LIT 95] LITTMAN M., DEAN T. and Kaelbling L., "On the Complexity of Solving Markov Decision Problems", *Proceedings of the 11th Annual Conference on Uncertainty in Artificial Intelligence (UAI'95)*, Montreal, Canada, 1995.
- [PAP 87] PAPADIMITRIOU C. H. and TSITSIKLIS J. N., "The Complexity of Markov Decision Processes", *Mathematics of Operations Research*, vol. 12, no. 3, pp. 441–450, 1987.
- [PUT 94] PUTERMAN M., *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, John Wiley & Sons, New York, 1994.

Chapter 2

Reinforcement Learning

2.1. Introduction

In Chapter 1, we presented planning methods in which the agent knows the transition and reward functions of the Markov decision problem it faces. In this chapter, we present *reinforcement learning* methods, where the transition and reward functions are not known in advance.

EXAMPLE 2.1. Let us consider again the case of a car used in the introduction of the previous chapter (see section 1.1). If we must look after a type of car never encountered before and do not possess the corresponding manual, we cannot directly model the problem as an MDP. We must first determine the probability of each breakdown, the cost of each repair operation and so on. In such a case, reinforcement learning is a way to determine through incremental experience the best way to look after the car by trial and error, eventually without even determining explicitly all probabilities and costs, just relying on a locally updated value of each action in each situation.

2.1.1. *Historical overview*

Our presentation strongly relies on Sutton and Barto's book [SUT 98]. However, before presenting the main concepts of reinforcement learning, we give a brief overview of the successive stages of research that led to the current formal understanding of the domain from the computer science viewpoint.

Chapter written by Olivier SIGAUD and Frédéric GARCIA.

Most reinforcement learning methods rely on simple principles coming from the study of animal or human cognition, such as the increased tendency to perform an action in a context if its consequences are generally positive in that context.

The first stages of computer science research that led to the reinforcement learning framework are dating back to 1960. In 1961, Michie [MIC 61] described a system playing tic-tac-toe by trial and error. Then Michie and Chambers [MIC 68] designed in 1968 a software maintaining an inverted pendulum's balance. In parallel, Samuel [SAM 59] presented a program learning to play checkers using a time difference principle. Both components, trial-and-error exploration and learning action sequences from time difference principles, are the basis of all subsequent reinforcement learning systems.

The union of both principles was achieved by Klop [KLO 72, KLO 75]. Following his work, Sutton and Barto implemented in 1981 [SUT 81] a linear perceptron whose update formula directly derives from the Rescorla-Wagner equation [RES 72], coming from experimental psychology. The Rescorla-Wagner equation can now be seen as the TD(0) equation (see section 2.5.1) approximated by a linear perceptron. This historical background explains the name *neuro-dynamic programming* used in the early stages of the field [BER 96].

In 1983, Barto, Sutton and Anderson [BAR 83] proposed AHC-LEARNING¹ considered as the first true reinforcement learning algorithm. Quite interestingly, AHC-LEARNING is an actor-critic approach that is now at the heart of the recent dialog between computational modeling and neurophysiological understanding of reinforcement learning in animals.

Finally, the mathematical formalization of reinforcement learning as we know it today dates back to 1988 when Sutton [SUT 88] and then Watkins [WAT 89] linked their work to the optimal control framework proposed by Bellman in 1957, through the notion of MDP [BER 95].

2.2. Reinforcement learning: a global view

Reinforcement learning stands at the intersection between the field of dynamic programming presented in the previous chapter and the field of machine learning. As a result, there are two ways to introduce the domain, either by explaining the conceptual differences with the dynamic programming context, or by contrasting reinforcement learning with other machine learning paradigms. We take the former standpoint in what follows and the latter in section 2.2.2. Then we introduce the

¹. *Adaptive Heuristic Critic learning*.

exploration/exploitation trade-off and a standard estimation method that are central to reinforcement learning.

2.2.1. Reinforcement learning as approximate dynamic programming

As we have seen in the previous chapter, the value function of a state typically reflects an estimate of the cumulated reward an agent might expect from being in this state given its current behavior. This notion is closely related to the notion of evaluation function in game theory. It distinguishes reinforcement learning approaches from all other simulation-based methods such as evolutionary algorithms that can also build optimal policies but without using the time structure of the underlying sequential decision problems.

Most reinforcement learning methods presented in this chapter are closely related to the dynamic programming algorithms presented in the previous chapter. Indeed, reinforcement learning can be seen as an extension of dynamic programming in the case where the dynamics of the problem is not known in advance.

Reinforcement learning methods are said to be model-free or model-based depending on whether they build a model of the transition and reward functions $p(s' | s, a)$ and $r(s, a)$ of the underlying MDP. As a matter of fact, dynamic programming methods can be seen as a special case of indirect (aka model-based) reinforcement learning methods where the model is perfect. When the model is unknown, model-based methods must build it online. In the discrete case studied in this chapter, this is done through a simple cumulative approach based on the maximum likelihood principle. In parallel, dynamic programming methods can be applied to the increasingly accurate model.

Direct (aka model-free) methods do not build a model: the hidden $p(s' | s, a)$ and $r(s, a)$ parameters are not estimated, but the value function V is updated locally while experimenting. This approach is advantageous in terms of memory use and, historically, reinforcement learning was initially restricted to direct methods. The central idea of direct reinforcement learning consists of improving a policy *locally* after each interaction with the environment. The update is local, thus it does not require a global evaluation of the policy. In practice, however, most direct reinforcement learning algorithms do not work directly on the policy, but iterate on an approximate value function as defined in the previous chapter.

All methods presented in this chapter deal with the case where the transition and reward functions of the MDP are unknown and an optimal value function V^* or a function Q^* representing the value of each action in each state will be approximated through experience, either on a model-based or a model-free basis.

2.2.2. Temporal, non-supervised and trial-and-error based learning

Reinforcement learning can be distinguished from other forms of learning based on the following characteristics:

- Reinforcement learning deals with temporal sequences. In contrast with one-step supervised or non-supervised learning problems where the order in which the examples are presented is not relevant, the choice of an action at a given time step will have consequences on the examples that are received at the subsequent time steps. This is particularly critical in the context of *delayed reward* problems where a reward may be received far after the important choices have been made.
- In contrast with supervised learning, the environment does not tell the agent what would be the best possible action. Instead, the agent may just receive a scalar reward representing the value of its action and it must *explore* the possible alternative actions to determine whether its action was the best or not.
- Thus, in order to determine the best possible action in any situation, the agent must try a lot of actions, through a trial-and-error process that implies some exploration, giving rise to the *exploration/exploitation trade-off* that is central to reinforcement learning.

2.2.3. Exploration versus exploitation

Exploitation consists of doing again actions which have proven fruitful in the past, whereas exploration consists of trying new actions, looking for a larger cumulated reward, but eventually leading to a worse performance. In theory, as long as the agent has not explored all possible actions in all situations, it cannot be sure that the best policy it knows is optimal. The problem is even more accurate when the environment is stochastic, leading to the necessity to try each action in each situation several times to get a reliable estimate of its average value. As a result, all convergence proofs for reinforcement learning algorithms assume that each transition will be experienced infinitely often [WAT 92]. In practice, however, we must make do with a partial exploration. Dealing with the exploration/exploitation trade-off consists of determining how the agent should explore to get as fast as possible a policy that is optimal or close enough to the optimum.

The agent may explore states as well as actions. With respect to states, the most natural choice consists of just following the dynamics of the agent-environment system, using the state resulting from the previous action as state of the next learning step. This way, exploration is focused on the relevant part of the state space that will be covered in practice by the agent. Online planning methods such as RTDP rely on this idea (see Chapter 6). Furthermore, there are many practical settings where doing otherwise is extremely difficult. In robotics, for instance, it may be hard not to follow the natural dynamics of the robot. Note, however, that in settings where there is a

state or set of states where the system stays once it enters it, it is generally necessary to reinitialize the system somewhere else to keep exploring.

With respect to actions, sampling the action uniformly in the action space at each iteration satisfies the convergence criterion, ensuring a maximum exploration, but such a choice is inefficient for two reasons. First, the value of the best action in each state is updated as often as the value of the worst action, which results in robust, but slow learning. Second, the cumulated reward along the experiment is just the average performance over all possible policies, which may be inadequate when learning is performed on the target system.

On the contrary, choosing at each iteration the greedy action with respect to the current known values, i.e. performing a greedy policy is not satisfactory either, because it generally converges to a suboptimal policy or may even diverge.

Thus reinforcement learning algorithms are based on a trade-off between full exploration and full exploitation which generally consists of performing the greedy action most of the time and an exploratory action from time to time. In this setting, finding the optimal way of tuning the rate of exploratory actions along an experiment is still an open problem.

Methods to deal with the exploration/exploitation trade-off can be classified into two categories: undirected methods and directed methods [THR 92].

Undirected methods use few information from the learning experiments beyond the value function itself. For instance, we may [BER 96]:

- follow the greedy policy along N_1 iterations, then perform random exploration along N_2 iterations;
- follow at each iteration the greedy policy with probability $1 - \epsilon$ or a random policy with probability ϵ , with $\epsilon \in [0, 1]$; these methods are called ϵ -greedy;
- draw an action according to a Boltzmann distribution, i.e. the probability of drawing action a is

$$p_T(a) = \frac{\exp\left(-\frac{Q_n(s_n, a)}{T}\right)}{\sum_{a'} \exp\left(-\frac{Q_n(s_n, a')}{T}\right)} \quad \text{with } \lim_{n \rightarrow \infty} T = 0,$$

where $Q_n(s, a)$ is the action-value function of performing action a in state s ; these methods are called *softmax*. The roulette wheel selection method is a particular softmax method in which T is constant instead of decreasing.

These diverse exploration functions call upon parameters (N_1 , N_2 , ϵ and T) controlling the exploration rate. The useful cases are $N_2 > 0$, $T < +\infty$ and $\epsilon < 1$, which experimentally ensure fast convergence.

In contrast, directed methods use specific exploration heuristics based on the information available from learning. The majority of such methods boil down to adding some exploration bonus to $Q(s, a)$ [MEU 96]. This bonus can be local such as in the interval estimation method [KAE 93], or propagated from state to state during learning [MEU 99]. Simple definitions of this exploration bonus can lead to efficient methods:

- in the *recency-based method*, the bonus is $\varepsilon \sqrt{\delta n_{sa}}$, where δn_{sa} is the number of iterations since the last execution of action a in state s , and where $\varepsilon \in [0, 1[$;
- in the *uncertainty estimation method*, the bonus is $\frac{c}{n_{sa}}$, where c is a constant and n_{sa} is the number of times action a was chosen in state s .

These different exploration methods can be used in the context of any reinforcement learning algorithm. Indeed, in all the temporal difference learning algorithms presented below, convergence towards optimal values is guaranteed provided that the Markov assumption is fulfilled and that each state is visited an infinite number of times.

2.2.4. General preliminaries on estimation methods

Before presenting general temporal difference methods in the next section, we will present Monte Carlo methods as a specific instance of these methods. For the sake of clarity, we will consider a policy π such that the corresponding Markov chain $p(s_{t+1} \mid s_t) = p(s_{t+1} \mid s_t, \pi(s_t))$ reaches from any initial state an (absorbing) terminal state T with a null reward. We consider the total reward criterion and we try to approximate $V(s) = E(\sum_{t=0}^{\infty} R_t \mid s_0 = s)$.

A simple way of performing this estimation consists of using the average cumulated reward over different trajectories obtained by following a policy π . If $R_k(s)$ is the expected utility in state s along trajectory k , then an estimation of the value function V in s based on the average after $k + 1$ trajectories is

$$\forall s \in S, \quad V_{k+1}(s) = \frac{R_1(s) + R_2(s) + \cdots + R_k(s) + R_{k+1}(s)}{k + 1}. \quad (2.1)$$

To avoid storing all the rewards, this computation can be reformulated in an incremental way:

$$\forall s \in S, \quad V_{k+1}(s) = V_k(s) + \frac{1}{k + 1} [R_{k+1}(s) - V_k(s)]. \quad (2.2)$$

To get $V_{k+1}(s)$, one just needs to store $V_k(s)$ and k . We can even avoid storing k , using an even more generic formula:

$$\forall s \in S, \quad V_{k+1}(s) = V_k(s) + \alpha [R_{k+1}(s) - V_k(s)], \quad (2.3)$$

where α is positive and should decrease along time, and we get

$$\lim_{k \rightarrow \infty} V_k(s) = V^\pi(s). \quad (2.4)$$

This incremental estimation method is at the heart of temporal difference methods, but is also present in Monte Carlo.

2.3. Monte Carlo methods

The Monte Carlo approach consists of performing a large number of trajectories from all states s in S , and estimating $V(s)$ as an average of the cumulated rewards observed along these trajectories. In each trial, the agent records its transitions and rewards, and updates the estimates of the value of the encountered states according to a discounted reward scheme. The value of each state then converges to $V^\pi(s)$ for each s if the agent follows policy π .

Thus the main feature of Monte Carlo methods lies in the incremental estimation of the value of a state given a series of cumulated reward values resulting from running a set of trajectories. The estimation method itself is the one presented in section 2.2.4.

More formally, let (s_0, s_1, \dots, s_N) be a trajectory consistent with the policy π and the unknown transition function $p()$, and let (r_1, r_2, \dots, r_N) be the rewards observed along this trajectory.

In the Monte Carlo method, the N values $V(s_t)$, $t = 0, \dots, N - 1$, are updated according to

$$V(s_t) \leftarrow V(s_t) + \alpha(s_t)(r_{t+1} + r_{t+2} + \dots + r_N - V(s_t)) \quad (2.5)$$

with the learning rates $\alpha(s_t)$ converging to 0 along the iterations. Then V converges almost surely towards V^π under very general assumptions [BER 96].

This method is called “every-visit” because the value of a state can be updated several times along the same trajectory. The error terms corresponding to all these updates are not independent, giving rise to a bias in the estimation of the V function along a finite number of trajectories [BER 96, page 190]. A simple solution to this bias problem consists of only updating $V(s)$ on the first visit of s in each trajectory. This “first-visit” method is not biased. Experimentally, the average quadratic error of the *first-visit* method tends to be lower than that of the *every-visit* method [SIN 96].

2.4. From Monte Carlo to temporal difference methods

The standard Monte Carlo methods above update the value function at the end of each trajectory. They can be improved so as to perform updates after every transition.

The update rule (2.5) can be rewritten in the following way, giving rise to a more incremental method:²

$$\begin{aligned} V(s_t) \leftarrow & V(s_t) + \alpha(s_t) \left((r_{t+1} + \gamma V(s_{t+1}) - V(s_t)) \right. \\ & + (r_{t+2} + \gamma V(s_{t+2}) - V(s_{t+1})) \\ & + \dots \\ & \left. + (r_N + \gamma V(s_N) - V(s_{N-1})) \right) \end{aligned}$$

or

$$V(s_t) \leftarrow V(s_t) + \alpha(s_t)(\delta_t + \delta_{t+1} + \dots + \delta_{N-1}) \quad (2.6)$$

by defining the temporal difference error δ_t by

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t), \quad t = 0, \dots, N-1.$$

The error δ_t can be interpreted in each state as a measure of the difference between the current estimation $V(s_t)$ and the corrected estimation $r_{t+1} + V(s_{t+1})$. It can be calculated as soon as the transition (s_t, r_{t+1}, s_{t+1}) has been observed, giving rise to an “online” version of the update rule (2.6). Thus we can start updating V without waiting for the end of the trajectory.

$$V(s_l) \leftarrow V(s_l) + \alpha(s_l)\delta_t, \quad l = 0, \dots, t. \quad (2.7)$$

Once again, if a trajectory can visit the same state several times, the online version can differ from the original version (2.6). However, it still converges almost surely and the first-visit method still seems more efficient in practice [SIN 96].

Thus, whereas standard Monte Carlo methods require that each trajectory is finished to perform updates, the online version presented just above can be said to be incremental. We can now turn to temporal difference methods that combine the incrementality property of dynamic programming and the experience-based update mechanism of Monte Carlo methods.

2.5. Temporal difference methods

So far, we have seen two classes of behavior optimization algorithms:

2. γ is 1, it is introduced in the equations to highlight the relationship with the temporal difference error used in temporal difference methods. Furthermore, we use $V(s_N) = V(T) = 0$.

– Dynamic programming algorithms apply when the agent knows the transition and reward functions. They perform their updates locally, which results in the possibility to act without waiting for the end of all iterations. However, they require a perfect knowledge of the underlying MDP functions.

– By contrast, Monte Carlo methods do not require any knowledge of the transition and reward functions, but they are not local.

Temporal difference methods combine properties of both previous methods. They rely on an incremental estimation of the value or action-value functions. Like Monte Carlo methods, they perform this estimation based on the experience of the agent and can do without a model of the underlying MDP. However, they combine this estimation using local estimation propagation mechanisms coming from dynamic programming, resulting in their incremental properties.

Thus temporal difference methods, which are at the heart of most reinforcement learning algorithms, are characterized by this combination of estimation methods with local updates incremental properties.

2.5.1. The TD(0) algorithm

The different criteria that can be used to determine the cumulated reward along a trajectory from local rewards have been presented in the previous chapter. Among these criteria, we will focus in this section on the discounted reward that leads to the most classical studies and proofs.

The basic temporal difference algorithm is TD. We denote it here by TD(0) for reasons that will get clear in the section dedicated to eligibility traces. This algorithm relies on a comparison between the actually received reward and the reward expected from the previous estimations.

If the estimated values $V(s_t)$ and $V(s_{t+1})$ in states s_t and s_{t+1} were exact, we would have

$$V(s_t) = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots, \quad (2.8)$$

$$V(s_{t+1}) = r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} + \dots. \quad (2.9)$$

Thus we would have

$$V(s_t) = r_{t+1} + \gamma V(s_{t+1}). \quad (2.10)$$

As stated before in section 2.4, the temporal difference error $\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$ measures the error between the current estimation $V(s_t)$ and the

corrected estimation $r_{t+1} + V(s_{t+1})$. The temporal difference method consists of correcting this error little by little by modifying $V()$ according to a Widrow-Hoff equation, often used in neural networks:

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] = V(s_t) + \alpha\delta_t. \quad (2.11)$$

This update equation immediately reveals the connection between temporal difference methods, Monte Carlo methods and dynamic programming. Indeed, it combines two features:

- As in dynamic programming algorithms, the estimate of $V(s_t)$ is updated as a function of the estimate of $V(s_{t+1})$. Thus the estimation is propagated to the current state from the successor states.
- As in Monte Carlo methods, each of these values results from an estimation based on the experience of the agent along its interactions with its environment.

We can thus see that temporal difference methods and, in particular, TD(0), are based on two coupled convergence processes, the first process estimating the immediate reward in each state and the second process approximating the value function resulting from these estimates by propagating them along transitions.

In the context of TD(0), the updates are local each time the agent performs a transition in its environment, relying on an information limited to (s_t, r_{t+1}, s_{t+1}) . The convergence of TD(0) was proven by Dayan and Sejnowski [DAY 94].

However, we must note that knowing the exact value of all states is not enough to determine what to do. If the agent does not know which action results in reaching any particular state, i.e. if the agent does not have a model of the transition function, knowing V does not help it determine its policy. This is why similar algorithms based on state-action pairs and computing the action-value function Q were developed, as we will show below.

2.5.2. The SARSA algorithm

As we just explained, finding V through $V = LV$ does not result in a policy if the model of transitions is unknown. To solve this problem, Watkins [WAT 89] introduced the action-value function Q , whose knowledge is similar to the knowledge of V when p is known.

DEFINITION 2.1 (action-value function Q). *The action-value function of a fixed policy π whose value function is V^π is*

$$\forall s \in S, a \in A, \quad Q^\pi(s, a) = r(s, a) + \gamma \sum_{s'} p(s' | s, a) V^\pi(s').$$

The value of $Q^\pi(s, a)$ is interpreted as the expected value when starting from s , executing a and then following the policy π afterwards. We have $V^\pi(x) = Q^\pi(x, \pi(x))$ and the corresponding Bellman equation is

$$\forall s \in S, a \in A \quad Q^*(s, a) = r(s, a) + \gamma \sum_{s'} p(s' | s, a) \max_b Q^*(s', b).$$

Then we have

$$\begin{aligned} \forall s \in S, \quad V^*(s) &= \max_a Q^*(s, a), \\ \pi^*(s) &= \operatorname{argmax}_a Q^*(s, a). \end{aligned}$$

The SARSA algorithm is similar to TD(0) in all respects but it works on state-action pairs rather than on states. Its update equation is

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]. \quad (2.12)$$

The information necessary to perform such an update is $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$, hence the name of the algorithm.

The SARSA algorithm suffers from one conceptual drawback: performing the updates as stated above implies knowing in advance what will be the next action a_{t+1} for any possible next state s_{t+1} . As a result, the learning process is tightly coupled to the current policy (the algorithm is called “on-policy”) and this complicates the exploration process. As a result, proving the convergence of SARSA was more difficult than proving the convergence of “off-policy” algorithms such as Q-learning, presented below, thus the corresponding convergence proof was published much later [SIN 00].

Note, however, that empirical studies often demonstrate the better performance of SARSA compared to Q-learning.

2.5.3. The Q-learning algorithm

The Q-learning algorithm can be seen as a simplification of the SARSA algorithm, given that it is no more necessary to determine the action at the next step to calculate updates. Its update equation is

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]. \quad (2.13)$$

Algorithm 2.1: Q-learning

```

/*  $\alpha_t$  is the learning rate */  

Initialize( $Q_0$ )  

for  $t \leftarrow 0$  to  $T_{tot} - 1$  do  

     $s_t \leftarrow \text{ChooseState}$   

     $a_t \leftarrow \text{ChooseAction}$   

     $(s_{t+1}, r_{t+1}) \leftarrow \text{Simulate}(s_t, a_t)$   

    {update  $Q_t$ :}  

    begin  

         $Q_{t+1} \leftarrow Q_t$   

         $\delta_t \leftarrow r_{t+1} + \gamma \max_b Q_t(s_{t+1}, b) - Q_t(s_t, a_t)$   

         $Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha_t(s_t, a_t)\delta_t$   

    end  

return  $Q_{T_{tot}}$ 

```

The main difference between SARSA and Q-learning lies in the definition of the error term. The $Q(s_{t+1}, a_{t+1})$ term in equation (2.12) is replaced by $\max_a Q(s_{t+1}, a)$ in equation (2.13). It would be equivalent in the context of a greedy policy since we would have $a_{t+1} = \arg \max_a Q(s_{t+1}, a)$. However, given the necessity to deal with the exploration/exploitation trade-off, we can perform a non-greedy action choice while still using the max term in the update rule. Thus SARSA performs updates as a function of the actually chosen actions whereas Q-learning performs updates as a function of the optimal actions irrespective of the actions performed, which is simpler.

This greater simplicity resulted both in earlier proofs of its convergence [WAT 92] and in the fact that it has been the most used algorithm over years, despite its eventual lower performance.

The Q-learning algorithm is shown in Algorithm 2.1. Updates are based on instantaneously available information. In this algorithm, the T_{tot} parameter corresponds to the number of iterations. There is here one learning rate $\alpha_t(s, a)$ for each state-action pair, it decreases at each visit of the corresponding pair. The Simulate function returns a new state and the corresponding reward according to the dynamics of the system. The choice of the current state and of the executed action is performed by functions ChooseState and ChooseAction and will be discussed hereafter. The Initialize function initializes the Q function with Q_0 , which is often initialized with null values, whereas more adequate choices can highly improve the performance.

The Q-learning algorithm can also be seen as a stochastic formulation of the *value iteration* algorithm presented in the previous chapter. Indeed, *value iteration* can be

expressed in terms of action value function:

$$\begin{aligned}
 V_{n+1}(s) &= \max_{a \in A} \overbrace{\left\{ r(s, a) + \gamma \sum_{s' \in S} p(s' | s, a) V_n(s') \right\}}^{Q_n(s, a)} \\
 \Rightarrow Q_{n+1}(s, a) &= r(s, a) + \gamma \sum_{s' \in S} p(s' | s, a) V_{n+1}(s') \\
 \Rightarrow Q_{n+1}(s, a) &= r(s, a) + \gamma \sum_{s' \in S} p(s' | s, a) \max_{a' \in A} Q_n(s', a').
 \end{aligned}$$

The Q-learning algorithm is obtained by replacing $r(s, a) + \sum_{s'} p(s' | s, a) \max_{a' \in A} Q(s', a')$ by its simplest unbiased estimator built from the current transition $r_{t+1} + \max_{a' \in A} Q(s_{t+1}, a')$.

The convergence of this algorithm is proved [WAT 89, JAA 94] (Q_n converges almost surely to Q^*) under the following assumptions:

- S and A are finite,
- each (s, a) pair is visited an infinite number of times,
- $\sum_n \alpha_n(s, a) = \infty$ and $\sum_n \alpha_n^2(s, a) < \infty$,
- $\gamma < 1$ or, if $\gamma = 1$, there exists an absorbing state with null reward for any policy.

The almost sure convergence means that, $\forall s, a$, the sequence of $Q_n(s, a)$ converges to $Q^*(s, a)$ with a probability equal to 1. In practice, the sequence $\alpha_n(s, a)$ is often defined as $\alpha_n(s, a) = \frac{1}{n_{sa}}$.

2.5.4. The TD(λ), SARSA(λ) and Q(λ) algorithms

The TD(0), SARSA and Q-learning algorithms only perform one update per time step in the state that the agent is visiting. As shown in Figure 2.1, this update process is particularly slow. Indeed, an agent deprived of any information on the structure of the value function needs at least n trials to propagate the immediate reward of a state to another state that is n transitions away. Before this propagation is achieved, if the initial values are null, the agent performs a random walk in the state space, which means that it needs an exponential number of steps as a function of n before reaching the reward “trail”.

A first naive way to solve the problem consists of using a memory of the trajectory and to propagate all the information backwards along the performed transitions each time a reward is reached. Such a memory of performed transitions is called an “eligibility trace”.

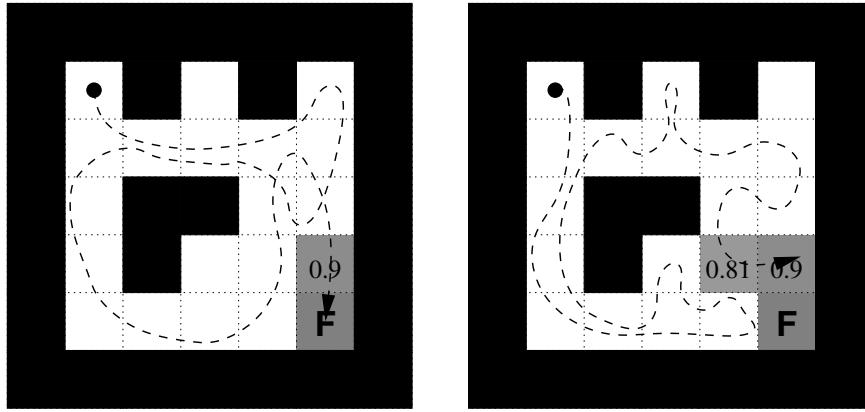


Figure 2.1. *Q-learning: first and second trial. We can see that, all values being initially null, the propagation of non-null values does not start until the agent finds the reward source for the first time, and only progresses once per trial*

Based on this idea, Sutton and Barto [SUT 98] proposed a class of algorithms called “ $\text{TD}(\lambda)$ ” that generalize $\text{TD}(0)$ to the case where the agent uses a memory of transitions. Later, SARSA and Q-learning have also been generalized into $\text{SARSA}(\lambda)$ and $Q(\lambda)$, by two different ways and two different authors for the latter [WAT 92, PEN 96].

These algorithms are more efficient than their standard counterpart, but they require more memory.

A problem with the naive approach above is that the required memory grows with the length of trajectories, which is obviously not feasible in the infinite horizon context.

In $\text{TD}(\lambda)$, $\text{SARSA}(\lambda)$ and $Q(\lambda)$, a more sophisticated approach that addresses the infinite horizon case is used.

We will discuss in section 2.6.1 another solution that performs several updates at each time step, in the indirect (or model-based) reinforcement learning framework. Let us start by describing the $\text{TD}(\lambda)$, $\text{SARSA}(\lambda)$ and $Q(\lambda)$ algorithms in more details.

2.5.5. Eligibility traces and $\text{TD}(\lambda)$

The originality of the $\text{TD}(\lambda)$ method lies in the proposal of a compromise between equations (2.6) and (2.7). Let $\lambda \in [0, 1]$ be a parameter. With the same notations as

before, the $\text{TD}(\lambda)$ algorithm defined by Sutton [SUT 88] is

$$V(s_t) \leftarrow V(s_t) + \alpha(s_t) \sum_{m=t}^{N-1} \lambda^{m-t} \delta_m, \quad t = 0, \dots, N-1. \quad (2.14)$$

The role of λ can be understood by rewriting equation (2.14) as follows:

$$V(s_t) \leftarrow V(s_t) + \alpha(s_t)(z_t^\lambda - V(s_t)).$$

Then we have

$$\begin{aligned} z_t^\lambda &= V(s_t) + \sum_{m=t}^{N-1} \lambda^{m-t} \delta_m \\ &= V(s_t) + \delta_t + \lambda \sum_{m=t+1}^{N-1} \lambda^{m-t-1} \delta_m \\ &= V(s_t) + \delta_t + \lambda(z_{t+1}^\lambda - V(s_{t+1})) \\ &= V(s_t) + r_{t+1} + V(s_{t+1}) - V(s_t) + \lambda(z_{t+1}^\lambda - V(s_{t+1})) \\ &= r_{t+1} + (\lambda z_{t+1}^\lambda + (1-\lambda)V(s_{t+1})). \end{aligned}$$

In the $\lambda = 0$ case, this is equivalent to using a one step horizon, as in dynamic programming. Thus this is $\text{TD}(0)$.

If $\lambda = 1$, equation (2.14) can be rewritten as

$$V(s_t) \leftarrow V(s_t) + \alpha(s_t) \sum_{m=t}^{N-1} \delta_m, \quad t = 0, \dots, N-1,$$

which is exactly equation (2.5) in Monte Carlo methods.

For any λ , both *first-visit* and *every-visit* approaches can be considered. An *online* version of the $\text{TD}(\lambda)$ learning algorithm described by equation (2.14) is possible:

$$V(s_l) \leftarrow V(s_l) + \alpha(s_l) \lambda^{t-l} \delta_t, \quad l = 0, \dots, t, \quad (2.15)$$

as soon as the transition (s_t, r_{t+1}, s_{t+1}) is performed and the δ_t error is calculated.

Applying $\text{TD}(\lambda)$ to evaluate a policy π according to the discounted criterion implies some modifications to the standard algorithm (2.14) or (2.15).

In the $\gamma = 1$ case, the update rule is

$$V(s_t) \leftarrow V(s_t) + \alpha(s_t) \sum_{m=t}^{\infty} (\gamma\lambda)^{m-t} \delta_m. \quad (2.16)$$

It is then clear that an offline algorithm to calculate V is inadequate in the absence of an absorbing terminal state since the trajectory is potentially infinite. The online version of (2.16) is then defined as

$$V(s) \leftarrow V(s) + \alpha(s) z_t(s) \delta_t, \quad \forall s \in S, \quad (2.17)$$

as soon as the t th transition (s_t, r_{t+1}, s_{t+1}) is performed and the δ_t error is calculated. The eligibility trace $z_t(s)$ is defined as follows.

DEFINITION 2.2 (cumulative eligibility trace).

$$\begin{aligned} z_0(s) &= 0, \quad \forall s \in S, \\ z_n(s) &= \begin{cases} \gamma\lambda z_{n-1}(s) & \text{if } s \neq s_n, \\ \gamma\lambda z_{n-1}(s) + 1 & \text{if } s = s_n. \end{cases} \end{aligned}$$

The eligibility coefficient increases at each visit of the corresponding state and exponentially decreases during the other iterations until a new visit of that state (see Figure 2.2).

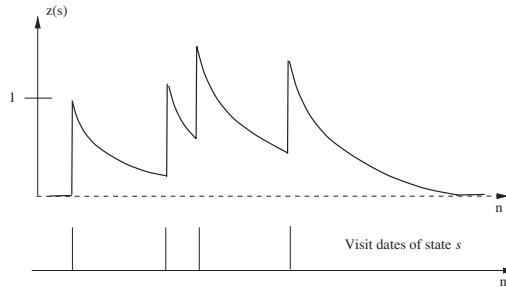


Figure 2.2. Cumulative eligibility trace: at each visit, one adds 1 to the previous value, thus this value can get over 1

In some cases, a slightly different definition of $z_n(s)$ seems to lead to a faster convergence of V .

DEFINITION 2.3 (eligibility trace with reinitialization).

$$\begin{aligned} z_0(s) &= 0 \quad \forall s \in S, \\ z_n(s) &= \begin{cases} \gamma\lambda z_{n-1}(s) & \text{if } s \neq s_n, \\ 1 & \text{if } s = s_n. \end{cases} \end{aligned}$$

Thus the value of the trace is bounded by 1, as shown in Figure 2.3.

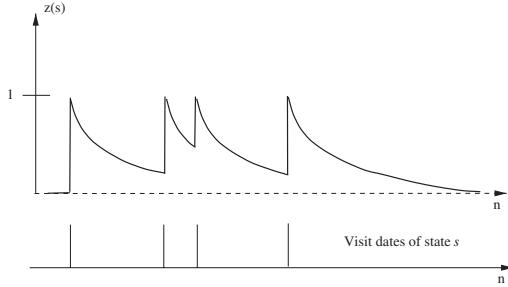


Figure 2.3. Eligibility trace with reinitialization: the value is set to 1 at each visit

The almost sure convergence of the $\text{TD}(\lambda)$ algorithm was shown for any λ , both with online and offline approaches, under classical assumptions of infinite visits of each state $s \in S$ and convergence of α towards 0 at each iteration n , such that $\sum_n \alpha_n(s) = \infty$ and $\sum_n \alpha_n^2(s) < \infty$ [JAA 94, BER 96].

The effect of the λ value is still poorly understood and tuning optimally its value for a given problem is still an open empirical problem.

A direct implementation of $\text{TD}(\lambda)$ based on eligibility traces is not efficient as soon as the state space S becomes large. A first approximate solution [SUT 98] consists of setting to 0 the value of all traces $z_n(s) < \varepsilon$, thus in stopping to maintain traces that have not been visited since more than $\frac{\log(\varepsilon)}{\log(\gamma\lambda)}$ transitions.

Another approximate method known as “truncated temporal differences”, or $\text{TTD}(\lambda)$ [CIC 95], is equivalent to storing a window of size m memorizing the m last visited state and updating from this window at each iteration n the value of the state visited in iteration $(n - m)$.

2.5.6. From $\text{TD}(\lambda)$ to $\text{SARSA}(\lambda)$

$\text{TD}(\lambda)$ can be applied in a reinforcement learning context to learn an optimal policy. To do so, we can couple $\text{TD}(\lambda)$ with an algorithm storing a sequence of policies π_t , as will become clear when presenting actor-critic approaches (see Chapter 5).

However, Q-learning directly integrates the temporal difference error idea. With the update rule of Q-learning:

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha_t \{ r_{t+1} + \gamma V_t(s_{t+1}) - Q_t(s_t, a_t) \}$$

for transition $(s_t, a_t, s_{t+1}, r_{t+1})$, and in the case where action a_t executed in state s_t is the optimal action for Q_t – that is for $a_t = \pi_{Q_t}(s_t) = \text{argmax}_b Q_t(s_t, b) -$,

then the error term is

$$r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t),$$

which is exactly that of TD(0). This can be generalized to $\lambda > 0$, through coupling TD(λ) and Q-learning methods.

The SARSA(λ) algorithm [RUM 94] is a first illustration. As shown in Algorithm 2.2, it adapts equation (2.17) to an action value function representation.

Algorithm 2.2: SARSA(λ)

```

/*  $\alpha_t$  is a learning rate */  

Initialize( $Q_0$ )  

 $z_0 \leftarrow 0$   

 $s_0 \leftarrow \text{ChooseState}$   

 $a_0 \leftarrow \text{ChooseAction}$   

for  $t \leftarrow 0$  until  $T_{tot} - 1$  do  

     $(s'_t, r_{t+1}) \leftarrow \text{Simulate}(s_t, a_t)$   

     $a'_t \leftarrow \text{ChooseAction}$   

    {update  $Q_t$  and  $z_t$ :}  

    begin  

         $\delta_t \leftarrow r_{t+1} + \gamma Q_t(s'_t, a'_t) - Q_t(s_t, a_t)$   

         $z_t(s_t, a_t) \leftarrow z_t(s_t, a_t) + 1$   

        for  $s \in S, a \in A$  do  

             $Q_{t+1}(s, a) \leftarrow Q_t(s, a) + \alpha_t(s, a)z_t(s, a)\delta_t$   

             $z_{t+1}(s, a) \leftarrow \gamma\lambda z_t(s, a)$   

    end  

    if  $s'_t$  non absorbing then  

         $s_{t+1} \leftarrow s'_t$  and  $a_{t+1} \leftarrow a'_t$   

    otherwise  

         $s_{t+1} \leftarrow \text{ChooseState}$   

         $a_{t+1} \leftarrow \text{ChooseAction}$   

return  $Q_{T_{tot}}$ 

```

The $z_t(s, a)$ eligibility trace is extended to state-action pairs and the state space exploration is guided by the dynamics of the system, unless a terminal state is reached.

2.5.7. Q(λ)

Dealing with the case where the optimal action $\pi_{Q_t}(s'_t)$ was not chosen leads to $Q(\lambda)$ algorithms proposed by Watkins [WAT 89] and Peng [PEN 94] (see also [SUT 98]).

Watkins' approach to $Q(\lambda)$ is characterized by the fact that it only considers $\lambda > 0$ along pieces of trajectories where the current policy π_{Q_t} has been followed. Thus, with respect to SARSA(λ), the update rules of Q_t and z_t are both modified, as shown in Algorithm 2.3. The problem with this approach is that, when the exploration rate is high, the z_t traces are often reset to 0 and $Q(\lambda)$ behaves closely to the original Q-learning.

Peng's approach to $Q(\lambda)$ solves this problem. It manages not to reset the trace z_t to 0 when an exploratory, non-optimal action is chosen. There are very few experimental results about this approach (see, however, [NDI 99]) and no comparison between TD(λ), SARSA(λ) and $Q(\lambda)$, thus drawing conclusions about these approaches is difficult. The only general conclusion that can be drawn is that using eligibility traces with $\lambda > 0$ reduces the number of necessary learning iterations to converge. The analysis in terms of computation time is more difficult, given the overhead resulting from the increased complexity of the algorithms.

Algorithm 2.3: $Q(\lambda)$

```

/*  $\alpha_t$  is a learning rate */  

Initialize( $Q_0$ )  

 $z_0 \leftarrow 0$   

 $s_0 \leftarrow \text{ChooseState}$   

 $a_0 \leftarrow \text{ChooseAction}$   

for  $t \leftarrow 0$  until  $T_{tot} - 1$  do  

     $(s'_t, r_{t+1}) \leftarrow \text{Simulate}(s_t, a_t)$   

     $a'_t \leftarrow \text{ChooseAction}$   

    {update  $Q_t$  and  $z_t$ :}  

    begin  

         $\delta_t \leftarrow r_{t+1} + \gamma \max_b Q_t(s'_t, b) - Q_t(s_t, a_t)$   

         $z_t(s_t, a_t) \leftarrow z_t(s_t, a_t) + 1$   

        for  $s \in S, a \in A$  do  

             $Q_{t+1}(s, a) \leftarrow Q_t(s, a) + \alpha_t(s, a)z_t(s, a)\delta_t$   

             $z_{t+1}(s, a) \leftarrow \begin{cases} 0 & \text{if } a'_t \neq \pi_{Q_t}(s'_t) \\ \gamma\lambda z_t(s, a) & \text{if } a'_t = \pi_{Q_t}(s'_t) \end{cases}$   

        end  

        if  $s'_t$  non absorbing then  

             $| s_{t+1} \leftarrow s'_t$  and  $a_{t+1} \leftarrow a'_t$   

        otherwise  

             $| s_{t+1} \leftarrow \text{ChooseState}$   

             $| a_{t+1} \leftarrow \text{ChooseAction}$   

return  $Q_{T_{tot}}$ 

```

2.5.8. The R-learning algorithm

All algorithms presented so far were based on the discounted criterion. The R-learning algorithm, proposed by Schwartz [SCH 93], is the adaptation of Q-learning to the average criterion and all previously explained principles are present.

The goal of this algorithm is to learn a policy π whose average reward ρ_π is as close as possible to the maximum average reward ρ^* of an optimal policy π^* . To ensure this property, R-learning maintains two correlated sequences ρ_t and R_t . The ρ_t sequence is only updated when the performed action was the greedy-action maximizing R_t in the current state s_t . This ρ_t sequence is an estimate of the criterion to maximize. As Q_t in Q-learning, R_t represents the relative value function U of a policy.

DEFINITION 2.4 (R value function). *We associate a new function R with a fixed policy π whose value function is U^π and whose average reward is ρ_π :*

$$\forall s \in S, a \in A, \quad R^\pi(s, a) = (r(s, a) - \rho_\pi) + \sum_{s'} p(s' | s, a) U^\pi(s').$$

Here again, $U^\pi(x) = R^\pi(x, \pi(x))$ and the Bellman equation applied to ρ^* and R^* becomes

$$\forall s \in S, a \in A, \quad R^*(s, a) = (r(s, a) - \rho^*) + \sum_{s'} p(s' | s, a) \max_b R^*(s', b) \quad (2.18)$$

with a guarantee that the average reward of policy $\pi_{R^*}(s) = \operatorname{argmax}_a R^*(s, a)$ is the optimal reward ρ^* .

As Q-learning, R-learning is a stochastic version of value iteration for equation (2.18).

Though there is no formal proof of the convergence of R-learning to the optimal policy, numerous experiments show that ρ_t efficiently approximates ρ^* under the same constraints as Q-learning.

Though R-learning is less famous and less used than Q-learning, it seems to perform more efficiently in practice [MAH 96b]. Few theoretical results are available on this point, but it has been shown that, in the finite horizon case, R-learning is very close to a parallel optimized version of Q-learning [GAR 98], which may explain its better empirical results.

Other average reward reinforcement learning algorithms have been proposed [MAH 96b]. Among them, the B algorithm [JAL 89] is an indirect method based on an adaptive estimation of the $p()$ and $r()$ functions. We can also mention Mahadevan's work [MAH 96a] who defined a model-based algorithm learning bias-optimal policies. Average reward reinforcement learning algorithms have been getting more popular recently (e.g. [BHA 07]).

Algorithm 2.4: R-learning

```

/*  $\alpha_t$  and  $\beta_t$  are learning rates */  

Initialize( $R_0, \rho_0$ )  

for  $t \leftarrow 0$  until  $T_{tot} - 1$  do  

     $s_t \leftarrow \text{ChooseState}$   

     $a_t \leftarrow \text{ChooseAction}$   

     $(s'_t, r_t) \leftarrow \text{Simulate}(s_t, a_t)$   

    {update  $R_t$  and  $\rho_t$ :}  

    begin  

         $R_{t+1} \leftarrow R_t$   

         $\delta_t \leftarrow r_t - \rho_t + \max_b R_t(s'_t, b) - R_t(s_t, a_t)$   

         $R_{t+1}(s_t, a_t) \leftarrow R_t(s_t, a_t) + \alpha_t(s_t, a_t)\delta_t$   

         $\rho_{t+1} \leftarrow \begin{cases} \rho_t & \text{if } a_t \neq \pi_{R_t}(s_t) \\ \rho_t + \beta_t\delta_t & \text{if } a_t = \pi_{R_t}(s_t) \end{cases}$   

    end  

return  $R_{T_{tot}}, \rho_{T_{tot}}$ 

```

We will now turn more generally to model-based approaches and we will present two algorithms that benefit from a proof of convergence in time polynomial in the size of the problem rather than exponential. These algorithms have also given rise to extension in the factored MDP case, as will be presented in Chapter 4.

2.6. Model-based methods: learning a model

In section 2.5.4, we highlighted the existence of a compromise between the learning velocity and the memory usage. The solution based on eligibility traces is limited by the fact that learning results from information extracted from the immediate past of the agent. We will now present a different approach where the agent builds a model of its interactions with the environment and can then use dynamic programming algorithms based on this model, independently from its current state.

The main question with this kind of approach is the following: Shall we wait for the most exact possible model before performing Bellman backups? Shall we rather intertwine learning the model and applying dynamic programming on it? The latter approach seems more efficient, as illustrated by Adaptive Real-Time Dynamic Programming (ARTDP) [BAR 95], where model learning, dynamic programming and execution are concurrent processes. DYNA architectures such as *Dyna* [SUT 90a], *Queue-Dyna* [PEN 93] and *Prioritized Sweeping* [MOO 93] are precursors of this approach.

2.6.1. Dyna architectures

All model-based reinforcement learning algorithms are based on the same motivation. Rather than waiting for actual transitions of the agent in its environment, one way to accelerate the propagation of values consists of building a model of the transition and reward functions and to use this model to apply a propagation algorithm independently of the actual behavior of the agent.

The DYNA architectures [SUT 90b], illustrated in Figure 2.4, were the first implementations of a system learning a model of transitions. Indeed, Sutton proposed this family of architectures to endow an agent with the capability to perform several updates at a given time step. To do so, several iterations of a dynamic programming algorithm are applied based on the model learned by the agent.

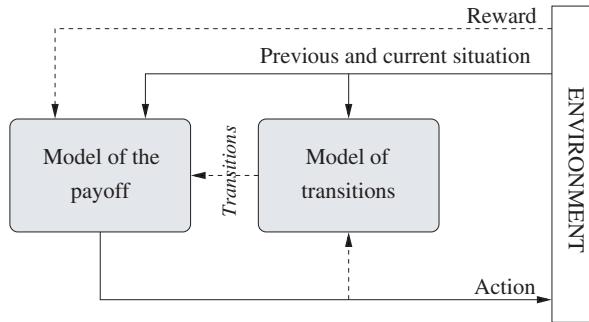


Figure 2.4. DYNA architectures combine a model of the reward function and a model of the transition function. The model of transitions is used to accelerate learning. Building both models requires a memory of the previous situation, which is not explicit on the figure

The model of transitions is a list of $\langle s_t, a_t, s_{t+1} \rangle$ triples indicating that, after the agent performs action a_t in state s_t , it may reach state s_{t+1} . Once this model is learned, the value iteration or policy iteration algorithms can be applied on it. We can also perform “virtual transitions” using the model and a local update rule, any number of times per time step, so as to accelerate the propagation of values. This approach is particularly appealing when using physical systems like a robot, where performing an actual transition is much more expensive than a virtual one.

Furthermore, if the reward function suddenly changes, the value propagation mechanism can help reconfigure the whole value function quickly. Finally, building a model of transitions endows the agent with a planning capability: if one state is known as desirable or considered as a goal, the agent can look forward into its transition graph for sequences of action that can result in reaching this state and execute the sequence that gives the highest probability of reaching it.

All *Dyna* architectures implement the principles described above. They differ from one another on the update rule or exploration strategy. The first implementation, DYNA-PI, also called *Dyna-AC* afterwards [SUT 98], is based on Policy Iteration.

Sutton [SUT 90b] showed that this system is less flexible than *Dyna-Q*, which is based on Q-learning. He finally proposed *Dyna-Q+*, a *Dyna-Q* augmented with active exploration capabilities.

Finally, the dynamic programming component of these systems can be improved by performing value updates more efficiently than a blind and systematic “sweep” of all transitions. Several improved sweeping mechanisms have been proposed, such as *Prioritized Sweeping* [MOO 93], *Focused Dyna* [PEN 92], *Experience Replay* [LIN 93] or *Trajectory Model Updates* [KUV 96].

2.6.2. The E^3 algorithm

The first algorithm converging in polynomial time with respect to the size of the problem is called E^3 , for *Explicit Explore and Exploit* [KEA 98]. Like DYNA architectures, E^3 builds a model of transitions and rewards. Nevertheless, instead of trying to build a complete model, it only memorizes the subset of the visited transitions that are involved in the optimal policy.

To build this model, the algorithm visits all states homogeneously, i.e. the agent chooses in any state the less often performed action in that state. Then it updates a model of the probability distribution over subsequent states, which constitute a model of the transition function.

This way to deal with the exploration/exploitation trade-off is based on the “pigeon hole principle” that stipulates that, with such an homogenous exploration, there will always be a time at which the model of the probability distribution for some state is close enough to the true transition function for that state. The authors define a notion of “known state” so that the number of visits necessary to consider a state as known remains polynomial in the size of the problem.

The model built by E^3 is an MDP containing all the “known” states with the observed transition probabilities and a unique absorbing state representing all the states that are not known yet. The algorithm then distinguishes two contexts:

- either there exists a policy that is close enough to the optimum based only on known states; in that case, dynamic programming can be applied to the known model;
- or this is not the case and the agent must keep exploring by taking actions leading to the absorbing state, so that more states get known.

This simple algorithm benefits from a proof of convergence in polynomial time to the optimal policy. The point is that, to decide in which of the above contexts it is, it

must determine whether performing more exploration beyond a certain horizon would improve the current best policy, which is itself a hard problem.

2.6.3. The R_{\max} algorithm

The R_{\max} algorithm [BRA 01] is an improvement over E^3 . It relies on the same general idea of building a model of known states and looking for an optimal policy in the finite horizon case under an average reward criterion. But, furthermore, it simplifies the management of the exploration/exploitation trade-off thanks to an optimistic initialization of the value function to the maximum expected immediate reward³ and it extends the algorithm from the MDP framework to the null sum stochastic games framework, allowing to take the presence of an opponent into account.

The main difference with E^3 comes from the fact that, thanks to the optimistic initialization of values, R_{\max} does not need to decide whether it should explore or exploit. Indeed, the agent just goes towards attractive states, and a state is more attractive than the others either because it is not known enough or because it is along the optimal trajectory. With respect to E^3 , R_{\max} is more robust in the sense that, in the presence of an opponent, the agent cannot control accurately the transitions of the system – thus the homogenous exploration strategy is difficult to ensure – whereas the optimistic exploration approach of R_{\max} guarantees that the agent will explore as efficiently as possible whatever the behavior of the opponent.

Nonetheless, as for E^3 , the proof of convergence of R_{\max} calls upon two unrealistic assumptions. First, it is assumed that the algorithm knows the horizon T beyond which trying to improve the model through exploration will not result in significant improvements of the policy. Second, it is assumed that the optimal policy on horizon T over the known model can be calculated at each time step, which is difficult in practice if T is large, as generally required in the context of a proof.

In practice, however, the authors show that the algorithm is efficient even with reasonable values of T , given that it performs better and better as T increases [BRA 03].

Other improvements have been proposed such as the heuristic sampling of transitions rather than a systematic exploration [PER 04, KEA 02].

This polynomial time exploration topic has been very active in recent years, with work around Probably Approximately Correct (PAC) approaches, like MBIE [STR 05] and *Delayed Q-learning* [STR 06]. A recent summary of these approaches and further improvements are proposed in [SZI 08].

3. Hence the name of the algorithm, R_{\max} .

2.7. Conclusion

Reinforcement learning is nowadays a very active research domain, at the interface between machine learning, statistical learning, behavior optimization, robotics, cognitive sciences and even neurophysiology. Various techniques have been developed to study the acquisition of an optimal behavior in a stochastic, dynamic and uncertain environment.

In this chapter, we have focused on the classical, basic methods that constitute the general background of the domain. This domain being very active, a lot of recent methods were not presented here.

Other classical approaches extending those presented here will be described in the next chapters of this book, such as online resolution methods (Chapter 6), dynamic programming with value function approximation (Chapter 3) or gradient methods to optimize parameterized policies (Chapter 5).

2.8. Bibliography

- [BAR 83] BARTO A., SUTTON R. and ANDERSON C. W., “Neuron-like adaptive elements that can solve difficult learning control problems”, *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-13, no. 5, pp. 834–846, 1983.
- [BAR 95] BARTO A., BRADTKE S. and SINGH S., “Learning to act using real-time dynamic programming”, *Artificial Intelligence*, vol. 72, pp. 81–138, 1995.
- [BER 95] BERTSEKAS D., *Dynamic Programming and Optimal Control*, Athena Scientific, Belmont, MA, 1995.
- [BER 96] BERTSEKAS D. and TSITSIKLIS J., *Neuro-Dynamic Programming*, Athena Scientific, Belmont, MA, 1996.
- [BHA 07] BHATNAGAR S., SUTTON R. S., GHAVAMZADEH M. and LEE M., “Incremental natural actor-critic algorithms”, *Advances in Neural Information Processing Systems*, MIT Press, 2007.
- [BRA 01] BRAFMAN R. I. and TENNENHOLTZ M., “R-max: a general polynomial time algorithm for near-optimal reinforcement learning”, *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI'01)*, Seattle, WA, pp. 953–958, 2001.
- [BRA 03] BRAFMAN R. I. and TENNENHOLTZ M., “Learning to coordinate efficiently: a model based approach”, *Journal of Artificial Intelligence Research*, vol. 19, pp. 11–23, 2003.
- [CIC 95] CICHOSZ P., “Truncating temporal differences: on the efficient implementation of TD(λ) for reinforcement learning”, *Journal of Artificial Intelligence Research*, vol. 2, pp. 287–318, 1995.
- [DAY 94] DAYAN P. and SEJNOWSKI T. J., “TD(λ) converges with probability 1”, *Machine Learning*, vol. 14, no. 3, pp. 295–301, 1994.

- [GAR 98] GARCIA F. and NDIAYE S., “A learning rate analysis of reinforcement-learning algorithms in finite-horizon”, *Proceedings of the 15th International Conference on Machine Learning (ICML'98)*, Morgan Kaufmann, San Mateo, CA, pp. 215–223, 1998.
- [JAA 94] JAAKKOLA T., JORDAN M. I. and SINGH S. P., “On the convergence of stochastic iterative dynamic programming algorithms”, *Neural Computation*, vol. 6, pp. 1185–1201, 1994.
- [JAL 89] JALALI A. and FERGUSON M., “Computationally efficient adaptative control algorithms for Markov chains”, *Proceedings of the IEEE Conference on Decision and Control (CDC'89)*, vol. 28, pp. 1283–1288, 1989.
- [KAE 93] KAEHLING L. P., *Learning in Embedded Systems*, MIT Press, Cambridge, MA, 1993.
- [KEA 98] KEARNS M. and SINGH S., “Near-optimal reinforcement learning in polynomial time”, *Machine Learning*, vol. 49, 1998.
- [KEA 02] KEARNS M. J., MANSOUR Y. and NG A. Y., “A sparse sampling algorithm for near-optimal planning in large Markov decision processes”, *Machine Learning*, vol. 49, no. 2-3, pp. 193–208, 2002.
- [KLO 72] KLOPF H. A., Brain function and adaptive systems, a heterostatic theory, Report no. Technical Report AFCRL-72-0164, Air Force Cambridge Research Laboratories, 1972.
- [KLO 75] KLOPF H. A., “A comparison of natural and artificial intelligence”, *SIGART newsletter*, vol. 53, pp. 11–13, 1975.
- [KUV 96] KUVAYEV L. and SUTTON R. S., “Model-based reinforcement learning with an approximate, learned model”, *Proceedings of the 9th Yale Workshop on Adaptive and Learning Systems*, New Haven, CT, Yale University Press, pp. 101–105, 1996.
- [LIN 93] LIN L.-J., “Scaling up reinforcement learning for robot control”, *Proceedings of the 10th International Conference on Machine Learning (ICML'93)*, Amherst, MA, pp. 182–189, 1993.
- [MAH 96a] MAHADEVAN S., “An average-reward reinforcement learning algorithm for computing bias-optimal policies”, *Proceedings of the National Conference on Artificial Intelligence (AAAI'96)*, vol. 13, 1996.
- [MAH 96b] MAHADEVAN S., “Average reward reinforcement learning: foundations, algorithms and empirical results”, *Machine Learning*, vol. 22, pp. 159–196, 1996.
- [MEU 96] MEULEAU N., Le dilemme entre exploration et exploitation dans l’apprentissage par renforcement, PhD thesis, Cemagref, université de Caen, 1996.
- [MEU 99] MEULEAU N. and BOURGINE P., “Exploration of multi-state environments: local measures and back-propagation of uncertainty”, *Machine Learning*, vol. 35, no. 2, pp. 117–154, 1999.
- [MIC 61] MICHEL D., “Trial and error”, *Science Survey*, vol. 2, pp. 129–145, 1961.
- [MIC 68] MICHEL D. and CHAMBERS R., “BOXES: an experiment in adaptive control”, *Machine Intelligence*, vol. 2, pp. 137–152, 1968.

- [MOO 93] MOORE A. and ATKESON C., "Prioritized sweeping: reinforcement learning with less data and less real time", *Machine Learning*, vol. 13, pp. 103–130, 1993.
- [NDI 99] NDIAYE S., Apprentissage par renforcement en horizon fini: application à la génération de règles pour la conduite de culture, PhD thesis, Paul Sabatier University, Toulouse, 1999.
- [PEN 92] PENG J. and WILLIAMS R., "Efficient learning and planning within the DYNA framework", MEYER J.-A., ROITBLAT H. L. and WILSON S. W., Eds., *Proceedings of the 2nd International Conference on Simulation of Adaptive Behavior (SAB'92)*, Cambridge, MA, MIT Press, pp. 281–290, 1992.
- [PEN 93] PENG J. and WILLIAMS R. J., "Efficient learning and planning within the Dyna framework", *Adaptive Behavior*, vol. 1, no. 4, pp. 437–454, 1993.
- [PEN 94] PENG J. and WILLIAMS R. J., "Incremental multi-step Q-learning", *Proceedings of the 11th International Conference on Machine Learning (ICML'94)*, Morgan Kaufmann, San Francisco, CA, pp. 226–232, 1994.
- [PEN 96] PENG J. and WILLIAMS R. J., "Incremental multi-step Q-learning", *Machine Learning*, vol. 22, pp. 283–290, 1996.
- [PER 04] PERET L. and GARCIA F., "On-line search for solving MDPs via heuristic sampling", *Proceedings of the European Conference on Artificial Intelligence (ECAI'04)*, 2004.
- [RES 72] RESCORLA R. A. and WAGNER A. R., "A theory of Pavlovian conditioning: variations in the effectiveness of reinforcement and nonreinforcement", BLACK A. H. and PROKAZY W. F., Eds., *Classical Conditioning II*, Appleton Century Croft, New York, pp. 64–99, 1972.
- [RUM 94] RUMMERY G. A. and NIRANJAN M., On-line Q-learning using connectionist systems, Report, Cambridge University Engineering Department, Cambridge, UK, 1994.
- [SAM 59] SAMUEL A., "Some studies in machine learning using the game of checkers", *IBM Journal of Research Development*, vol. 3, no. 3, pp. 210–229, 1959.
- [SCH 93] SCHWARTZ A., "A reinforcement learning method for maximizing undiscounted rewards", *Proceedings of the 10th International Conference on Machine Learning (ICML'93)*, Amherst, MA, 1993.
- [SIN 96] SINGH S. P. and SUTTON R. S., "Reinforcement learning with replacing eligibility traces", *Machine Learning*, vol. 22, no. 1, pp. 123–158, 1996.
- [SIN 00] SINGH S. P., JAAKKOLA T., LITTMAN M. L. and SZEPESVARI C., "Convergence results for single-step on-policy reinforcement learning algorithms", *Machine Learning*, vol. 38, no. 3, pp. 287–308, 2000.
- [STR 05] STREHL A. L. and LITTMAN M. L., "A theoretical analysis of model-based interval estimation", *Proceedings of the 22nd International Conference on Machine Learning (ICML'05)*, ACM Press, New York, pp. 856–863, 2005.
- [STR 06] STREHL A. L., LI L., WIEWIORA E., LANGFORD J. and LITTMAN M. L., "PAC model-free reinforcement learning", *Proceedings of the 23rd International Conference on Machine Learning (ICML'06)*, ACM Press, New York, pp. 881–888, 2006.

- [SUT 81] SUTTON R. and BARTO A., “Toward a modern theory of adaptive network: expectation and prediction”, *Psychological Review*, vol. 88, no. 2, pp. 135–170, 1981.
- [SUT 88] SUTTON R., “Learning to predict by the method of temporal differences”, *Machine Learning*, vol. 3, no. 1, pp. 9–44, 1988.
- [SUT 90a] SUTTON R. S., “Integrated architectures for learning, planning and reacting based on approximating dynamic programming”, *Proceedings of the 7th International Conference on Machine Learning (ICML'90)*, San Mateo, CA, pp. 216–224, 1990.
- [SUT 90b] SUTTON R. S., “Planning by incremental dynamic programming”, *Proceedings of the 8th International Conference on Machine Learning (ICML'91)*, pp. 353–357, Morgan Kaufmann, San Mateo, CA, 1990.
- [SUT 98] SUTTON R. S. and BARTO A. G., *Reinforcement Learning: An Introduction*, Bradford Book, MIT Press, Cambridge, MA, 1998.
- [SZI 08] SZITA I. and LŐRINCZ A., “The many faces of optimism: a unifying approach”, *ICML '08: Proceedings of the 25th International Conference on Machine Learning*, ACM Press, New York, pp. 1048–1055, 2008.
- [THR 92] THRUN S., “The role of exploration in learning control”, WHITE D. and SOFGE D., Eds., *Handbook for Intelligent Control: Neural, Fuzzy and Adaptive Approaches*, Van Nostrand Reinhold, Florence, Kentucky, 1992.
- [WAT 89] WATKINS C., Learning from delayed rewards, PhD thesis, Cambridge University, Cambridge, UK, 1989.
- [WAT 92] WATKINS C. and DAYAN P., “Q-learning”, *Machine Learning*, vol. 8, no. 3, pp. 279–292, 1992.

Chapter 3

Approximate Dynamic Programming

In any complex or large-scale sequential decision making problem, there is a crucial need to use function approximation to represent relevant functions, such as the value function or the policy.

The dynamic programming (DP) and reinforcement learning (RL) methods introduced in Chapters 1 and 2 make the implicit assumption that the value function can be perfectly represented (i.e. kept in memory), for example, by using a look-up table (with a finite number of entries) assigning a value to all possible states (assumed to be in finite number) of the system. Those methods are called “exact” because they provide an exact computation of the optimal solution of the considered problem (or at least, enable the computations to converge to this optimal solution). However, such methods often apply to toy problems only, since in most interesting applications, the number of possible states is so large (and possibly infinite if we consider continuous spaces) that a perfect representation of the function at all states is impossible. It becomes necessary to approximate the function by using a moderate number of coefficients (which can be stored in a computer), and therefore extend the range of DP and RL methods to methods using such “approximate” representations. These “approximate” methods combine DP and RL methods with function approximation tools.

EXAMPLE 3.1. Let us come back to the example of the car where maintenance operations should be optimized (see Chapter 1, section 1.1). We have seen that the number of possible states is very large, although the state may sometimes be factorized (see Chapter 4). However, even if we consider a single element of the

Chapter written by Rémi MUNOS.

car, the brakes for example, the number of possible states may be described by a continuous variable (thickness of break pads). The state of (this element of) the car may thus vary continuously in a given interval and all previously seen methods would not apply here since they assume that the number of possible states is finite. The tools introduced in this chapter enable us to take into account this problematic of continuous state space. This point is illustrated explicitly in the optimal replacement problem (see section 3.2.3).

In this chapter we study the use of function approximation (taking inspiration from statistical learning and approximation theory) for approximating the value function, and we generalize DP and RL methods to this approximate resolution setting. Performance bounds resulting from the use of approximate representations will be expressed in terms of the capacity and approximation power of the considered function spaces.

3.1. Introduction

The idea of using function approximation in DP comes back to the early days of this domain. For example, Samuel [SAM 67] used linear approximation of the value function for the game of checkers; Bellman and Dreyfus [BEL 59] used polynomials in order to increase the velocity of DP. A first theoretical analysis is undertaken in [REE 77]. More recently, RL and DP combined with function approximation enabled us to solve successfully several large-scale applications; for example, the program *TD-Gammon* [TES 95], using a neural network, produced a world champion program in backgammon. Other application domains include operational research and task scheduling [ZHA 95], e.g. the control of a set of lifts [CRI 96], factory maintenance [MAH 97], dynamic channel allocation [SIN 97] and seats allocation on planes [GOS 04]. Some specific applications are described more precisely in Part 3 of this book.

The purpose of this chapter is to present briefly the new problems that appear when one considers *approximate solutions* to PD and RL problems and to provide bounds on the performance loss resulting of these approximations.

For simplicity of presentation, we consider here only the case of a *discounted reward* problem in *infinite-time horizon*. Thus, for a given policy π (mapping an action to each possible state), the value function V^π is defined as the expectation of the sum of future discounted rewards:

$$V^\pi(s) \stackrel{\text{def}}{=} \mathbb{E} \left[\sum_{t \geq 0} \gamma^t r(s_t, \pi(s_t)) \mid s_0 = s, \pi \right], \quad (3.1)$$

where $\gamma \in [0, 1)$ is the discount factor.

Other criteria (e.g. finite-time horizon or undiscounted problems) are subject to similar conceptual analysis, but may show differences in the mathematical formalism. For these extensions, we refer the reader to the work of Bertsekas and Tsitsiklis [BER 96].

The approach followed in this chapter is to build an approximation of the optimal value function, hoping that the performance of a greedy policy with respect to (w.r.t.) such an approximation will provide near-optimal performance. This hope is justified by the fact that if V is a good approximation of the optimal value function V^* , then the performance V^π of the policy π which is greedy w.r.t. V is close to the performance V^* of an optimal policy π^* . Recall that we say that a policy $\pi : S \mapsto A$ is greedy w.r.t. a function V if, at any state $s \in S$, the action $\pi(s)$ is an action that maximizes the immediate reward plus the discounted expectation of V at the next state, i.e.

$$\pi(s) \in \arg \max_{a \in A} \left[r(s, a) + \gamma \sum_{s' \in S} p(s' | s, a) V(s') \right].$$

Indeed we have the following result [BER 96, page 262].

PROPOSITION 3.1. *Let V be a real-valued function defined on S and let π be a policy greedy w.r.t. V . Then, the performance loss resulting from using policy π instead of the optimal policy (i.e. difference between the optimal value function V^* and the value function V^π) is bounded as*

$$\|V^* - V^\pi\|_\infty \leq \frac{2\gamma}{1-\gamma} \|V - V^*\|_\infty, \quad (3.2)$$

where $\|\cdot\|_\infty$ denotes the supremum norm, written as L^∞ (i.e. $\|f\|_\infty \stackrel{\text{def}}{=} \max_{s \in S} |f(s)|$), and γ is the discount factor.

Note that, in general, the value function V^π is different from the function V . This bound gives an argument justifying our approach of searching to construct a good approximation of the optimal value function (i.e. small $\|V - V^*\|$) in order to deduce a policy whose performance is close to the optimum (i.e. small $\|V^* - V^\pi\|$). Since the proof is elementary we include it now.

Proof. We remind the definition of the Bellman operators L and L_π (defined in Chapter 1, section 1.5.2): for any real-valued function W defined over S ,

$$LW(s) \stackrel{\text{def}}{=} \max_{a \in A} \left[r(s, a) + \gamma \sum_{s' \in S} p(s' | s, a) W(s') \right],$$

$$L_\pi W(s) \stackrel{\text{def}}{=} r(s, \pi(s)) + \gamma \sum_{s' \in S} p(s' | s, \pi(s)) W(s').$$

L_π also writes (in vector notation) $L_\pi W = r_\pi + \gamma P_\pi W$, where P_π is the transition matrix for policy π (i.e. whose elements (s, s') are $p(s' | s, \pi(s))$), and r_π the vector with components $r(s, \pi(s))$.

We have the property that L and L_π are contraction operators in L^∞ norm, with a contraction factor γ . This means that, for any couple of functions W_1 and W_2 defined over S , we have $\|LW_1 - LW_2\|_\infty \leq \gamma \|W_1 - W_2\|_\infty$ and similarly for the operator L_π (the proof is elementary).

From the fact that V^* and V^π are fixed points of the operators L and L_π , respectively (which means that $V^* = LV^*$ and $V^\pi = L_\pi V^\pi$), by using the triangle inequality, it comes that

$$\begin{aligned} \|V^* - V^\pi\|_\infty &\leq \|LV^* - L_\pi V\|_\infty + \|L_\pi V - L_\pi V^\pi\|_\infty \\ &= \|LV^* - LV\|_\infty + \gamma \|V - V^\pi\|_\infty \\ &\leq \gamma \|V^* - V\|_\infty + \gamma (\|V - V^*\|_\infty + \|V^* - V^\pi\|_\infty), \end{aligned}$$

where we use at the second line the fact that the policy π is greedy with respect to V , i.e. $LV = L_\pi V$. We deduce the bound on the performance loss $V^* - V^\pi$ resulting from using policy π instead of an optimal policy π^* :

$$\|V^* - V^\pi\|_\infty \leq \frac{2\gamma}{1-\gamma} \|V^* - V\|_\infty. \quad \square$$

In order to build a good approximation of the optimal value function, we need to generalize the previously seen DP and RL algorithms. We now start by presenting the approximate value iteration algorithm. Then, in sections 3.3 and 3.4, we consider the approximate policy iteration algorithm and the Bellman residual minimization algorithm. Finally, in section 3.5, we explain the limits of the usual L^∞ -norm analysis of DP and explore extensions to L^p -norm analysis (for $p \geq 1$) and establish preliminary links with the field of Statistical Learning. In particular, we provide finite-time (i.e. non-asymptotic) performance bounds in terms of the capacity and richness of the considered function spaces used in the approximations.

3.2. Approximate value iteration (AVI)

Again, we consider solving a Markov decision problem (MDP) [PUT 94] using a discounted criterion under an infinite horizon setting. The set of states is assumed to be large (but finite for the moment, in order to keep notations simple).

From what has been said in Chapter 1, the *value iteration* algorithm consists of calculating the optimal value function V^* by successive evaluations V_n calculated by

the iteration $V_{n+1} = LV_n$, where L is the Bellman operator. Thanks to the contraction property (in L^∞ -norm) of L , the iterates V_n converge to V^* (the fixed-point of L) when $n \rightarrow \infty$ (since we have $\|V_{n+1} - V^*\|_\infty = \|TV_n - TV^*\|_\infty \leq \gamma\|V_n - V^*\|_\infty \leq \gamma^n\|V_1 - V^*\|_\infty$).

When the number of states is so large that an exact representation of the function V_n is impossible to memorize, we need to consider approximate representations, which results in the so-called *approximate value iteration* (AVI) algorithm. AVI is very popular and has been implemented from the early works on DP [BEL 59, SAM 59] and more recently in the context of RL [BER 96, SUT 98] with many variants, such as the *fitted Q-iteration* [ERN 05].

Let us write \mathcal{F} the space of considered approximation functions. For example, \mathcal{F} may be the span of a finite set of generative functions (called *features*): Any function in \mathcal{F} is thus defined as a linear combination of the features weighted by some real coefficients. This is the so-called *linear approximation*.

The AVI algorithm builds a sequence of functions $V_n \in \mathcal{F}$ calculated according to the iterations:

$$V_{n+1} = \mathcal{A}LV_n, \quad (3.3)$$

where L is the Bellman operator and \mathcal{A} an *approximation operator* from functions in \mathcal{F} . For example, in the case of linear approximation, \mathcal{A} is the projection operator onto \mathcal{F} , which means that $\mathcal{A}f \in \mathcal{F}$ is the function in \mathcal{F} with the minimum distance to f : $\|\mathcal{A}f - f\| = \inf_{g \in \mathcal{F}} \|g - f\|$ (for some norm $\|\cdot\|$).

Thus, AVI consists of a sequence of iterations where, at each round, a new representation $V_{n+1} \in \mathcal{F}$ is obtained by projecting onto \mathcal{F} the Bellman image of the previous approximation V_n . The iteration process (3.3) is illustrated in Figure 3.1.

When the approximation operation is performed based on data (samples) (e.g. when considering a projection minimizing an empirical distance), then we call this *supervised learning* or *regression* (see, e.g. [HAS 01]). This case is illustrated in the next section.

3.2.1. Sample-based implementation and supervised learning

For illustration, a sample-based implementation of AVI could be defined as follows: at step n , we select K states $(s_k)_{1 \leq k \leq K}$ independently sampled from a given distribution μ over the state space S . We calculate the Bellman image of V_n at those states, thus defining the values $\{v_k \stackrel{\text{def}}{=} LV_n(s_k)\}$. Then we make a call to a supervised learning algorithm provided with the learning data $\{(s_k, v_k)\}_{1 \leq k \leq K}$

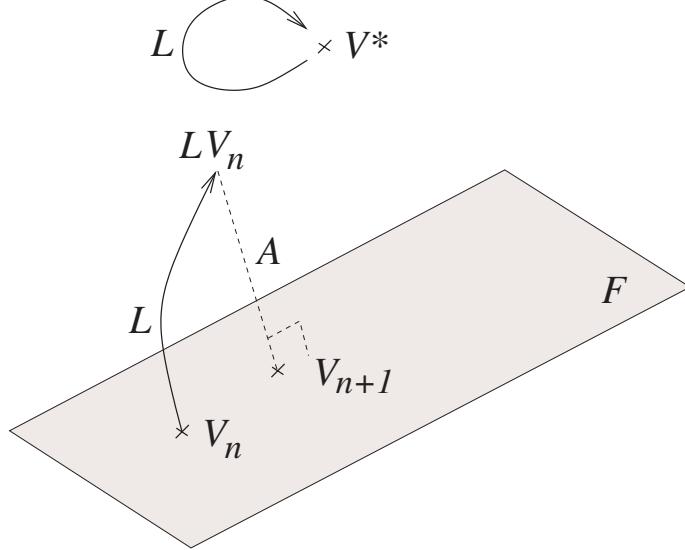


Figure 3.1. Schematic representation of an AVI iteration: the approximation space \mathcal{F} is a vector space of finite dimension. $V_n \in \mathcal{F}$ is the approximation at time n . The Bellman operator L is applied to V_n (in general, LV_n does not belong to \mathcal{F} , i.e. it is not representable in this approximation architecture), then the projection A onto \mathcal{F} defines the next approximation $V_{n+1} = ALV_n$. The optimal value function V^* (fixed-point of L) is also shown

(input, desired output), which returns a function $V_{n+1} \in \mathcal{F}$ minimizing an empirical error, such as

$$V_{n+1} = \arg \min_{f \in \mathcal{F}} \frac{1}{K} \sum_{1 \leq k \leq K} (f(s_k) - v_k)^2. \quad (3.4)$$

This minimization problem is defined in terms of the quadratic norm L^2 , like this is the case in least-squares methods, locally linear regression, neural networks and many other supervised learning algorithms. Of course, there exist other minimization problems using different norms, such as the L^1 -norm (absolute value) or its variants (such as the ϵ -insensitive L^1 norm used in Support Vector Machines [VAP 98]) as well as regularized¹ version [FAR 09] which are often used. This regression problem is a specific case of supervised learning (or statistical learning). We will not go further in the details of the important issues (overfitting, bias-variance trade-off) of this domain and refer the interested reader to usual references, such as [HAS 01].

1. Also called penalized, i.e. where a penalty term is added to the empirical error in order to prevent from selecting highly irregular functions.

Let us simply mention that linear approximation consists of performing a projection onto a vector space spanned by a finite family of features, which includes decomposition with splines, Fourier bases, radial basis functions, wavelets. Sometimes, a better regression is obtained when the family of generative functions upon which the projection is performed is chosen according to the smoothness and regularities of the function we wish to approximate. Such cases, referred to as *non-linear approximation*, may be particularly efficient when the target function possesses local regularities (e.g. in adaptive wavelet bases [MAL 97] such functions may be represented sparsely, i.e. with a small number of non-zero coefficients). Greedy algorithms, like *matching pursuit* and other variants [DAV 97] select the best basis functions among a redundant dictionary. Approximation theory studies the approximation error in terms of the regularities of the target function [DEV 98]. In statistical learning, examples of other non-linear approximation tools that are very popular are *neural networks*, *locally weighted regression* [ATK 97], *Support Vector Machines* and *kernel methods* in *Reproducing Kernel Hilbert Spaces* [VAP 97, VAP 98].

3.2.2. Analysis of the AVI algorithm

Consider the AVI algorithm defined by the iteration (3.3) and define

$$\varepsilon_n \stackrel{\text{def}}{=} LV_n - V_{n+1} \quad (3.5)$$

as the *approximation error* at round n . In general, AVI does not converge to the optimal value function V^* (in opposition to the value iteration algorithm) since V^* usually does not belong to the representation space \mathcal{F} . Actually, even if $V^* \in \mathcal{F}$, we would have no guarantee that the iterates V_n converge to V^* . In reality, AVI may oscillate or even diverge, as illustrated in very simple examples described in [BAI 95, TSI 96b] and [BER 96, page 334]. However, this algorithm is very popular because it has shown good results in real applications.

In order to understand the reason for this variety of observed behaviors depending on the applications, we wish to analyze the AVI algorithm and establish performance bounds. A first result provides a bound on the performance loss (w.r.t. optimal performance) resulting from the use of a policy π_n greedy w.r.t. V_n , as a function of the approximation errors ε_n .

PROPOSITION 3.2 [BER 96]. *Writing π_n a greedy policy w.r.t. the approximate V_n , and V^{π_n} the value function corresponding to that policy, we have*

$$\limsup_{n \rightarrow \infty} \|V^* - V^{\pi_n}\|_\infty \leq \frac{2\gamma}{(1-\gamma)^2} \limsup_{n \rightarrow \infty} \|\varepsilon_n\|_\infty. \quad (3.6)$$

Proof. From (3.2) applied to V_n , we deduce that

$$|V^* - V^{\pi_n}|_\infty \leq \frac{2\gamma}{1-\gamma} |V^* - V_n|_\infty. \quad (3.7)$$

Moreover,

$$\begin{aligned} |V^* - V_{n+1}|_\infty &\leq \|LV^* - LV_n\|_\infty + \|LV_n - V_{n+1}\|_\infty \\ &\leq \gamma \|V^* - V_n\|_\infty + \|\varepsilon_n\|_\infty. \end{aligned}$$

Now, taking the limit sup, it comes that

$$\limsup_{n \rightarrow \infty} \|V^* - V_n\|_\infty \leq \frac{1}{1-\gamma} \limsup_{n \rightarrow \infty} \|\varepsilon_n\|_\infty,$$

which, combined to (3.7), leads to (3.6). \square

Notice that this bound makes use of the L^∞ -norm of the approximation errors ε_n , which means that it depends on the worst possible error $\varepsilon_n(s)$ over all the domain (when s sweeps S). This uniform error is generally difficult to control, especially for large state space problems. In addition, it is not very useful from a practical point of view since most function approximation techniques and supervised learning algorithms solve, as illustrated in section 3.2.1, an empirical minimization problem in L^2 or L^1 norm. We will see in section 3.5 a possible extension of Proposition 3.2 to L^p -norms (for $p \geq 1$). However, let us mention existing works [GOR 95, GUE 01] on function approximation using L^∞ -norm such as the *averagers* in the field of DP.

3.2.3. Numerical illustration

Here we illustrate the behavior of the AVI algorithm for an optimal replacement problem, excerpted from [RUS 96]. We also show on this example that the previous results generalize naturally to the case of continuous state spaces.

A 1D variable $s_t \in S \stackrel{\text{def}}{=} [0, s_{\max}]$ measures the accumulated use of a product (e.g. the odometer may measure the global state of a car). $s_t = 0$ denotes a brand new product. At each discrete time t (e.g. each year), there are two possible decisions: either keep ($a_t = K$) or replace ($a_t = R$) the product, in which case, an additional cost C_{replace} (for selling the current product and buying a new one) is suffered.

We assume transitions follow an exponential law with parameter β (with a truncated tail): if the next state y is larger than a given maximum value s_{\max} (e.g. a critical state for the car), then a new state is immediately drawn and a penalty

cost $C_{\text{death}} > C_{\text{replace}}$ is received. Thus, by writing $p(\cdot | s, a)$ the transition density function of the next state given that the current state is s and the chosen action is $a \in \{\text{K}, \text{R}\}$ (which means that for any subset $B \subset S$ the probability of being in B at the next state is $\int_B p(ds' | s, a)$), we define

$$p(s' | s, \text{R}) \stackrel{\text{def}}{=} \begin{cases} q(s') & \text{if } s' \in [0, s_{\max}], \\ 0 & \text{otherwise;} \end{cases}$$

$$p(s' | s, \text{K}) \stackrel{\text{def}}{=} \begin{cases} q(s' - s) & \text{if } s' \in [s, s_{\max}], \\ q(s' - s + s_{\max}) & \text{if } s' \in [0, s), \\ 0 & \text{otherwise;} \end{cases}$$

with $q(s) \stackrel{\text{def}}{=} \beta e^{-\beta s} / (1 - e^{-\beta s_{\max}})$ (the truncated exponential density).

The immediate cost function (opposite of a reward) $c(s)$ is the sum of a slowly increasing monotonous function (which may correspond to maintenance costs for the car) and a punctually discontinuous cost function (e.g. insurance fees). The immediate cost function and the optimal value function (numerically calculated by using a very fine grid) are shown in Figure 3.2 for the numerical values: $\gamma = 0.6$, $\beta = 0.6$, $C_{\text{replace}} = 50$, $C_{\text{dead}} = 70$ and $s_{\max} = 10$.

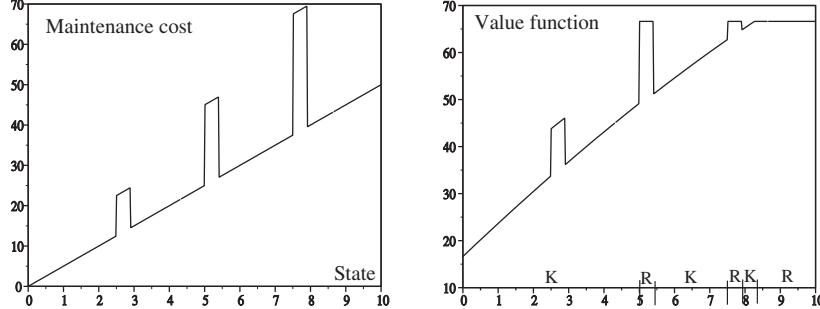


Figure 3.2. Immediate cost function c and optimal value function V^* . The letters R and K on the right figure indicate the optimal policy (greedy w.r.t. V^*) as a function of the state

We consider the implementation of the sampled-based AVI algorithm described in section 3.2.1. The states $\{s_k \stackrel{\text{def}}{=} ks_{\max}/K\}_{0 \leq k < K}$ (with $K = 200$) are uniformly sampled over the domain S . The approximation function space \mathcal{F} is defined by the vector space of dimension $M = 20$ generated by a cosine family:

$$\mathcal{F} \stackrel{\text{def}}{=} \left\{ V_{\alpha}(s) \stackrel{\text{def}}{=} \sum_{m=1}^M \alpha_m \cos \left(m\pi \frac{s}{s_{\max}} \right) \right\}_{\alpha \in \mathbb{R}^M}.$$

Thus, at each step n , a new approximation $V_{n+1} \in \mathcal{F}$ is obtained by solving the least-squares fitting problem:

$$V_{n+1} = \arg \min_{f \in \mathcal{F}} \frac{1}{K} \sum_{k=1}^K [f(s_k) - LV_n(s_k)]^2.$$

We start with an initial value function $V_0 = 0$. Figure 3.3 represents the first iteration: the values $\{LV_0(s_k)\}_{1 \leq k \leq K}$ are shown by the crosses on the left figure and the corresponding best fit $V_1 \in \mathcal{F}$ (best approximation in the space \mathcal{F}). Figure 3.4 illustrates analogously the second iteration. Figure 3.5 shows the approximate value function $V_{20} \in \mathcal{F}$ obtained after 20 iterations. In this simulation, the AVI algorithm performs well and a good approximation of the optimal value function V^* is obtained in \mathcal{F} .

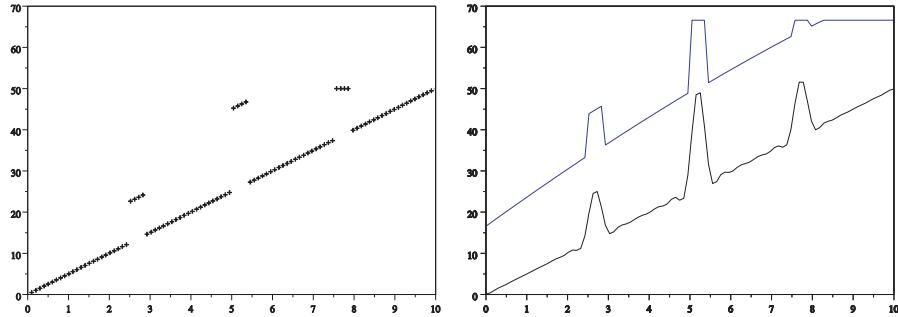


Figure 3.3. Values $\{LV_0(s_k)\}_{1 \leq k \leq K}$ (left figure), best fit $V_1 \in \mathcal{F}$ of LV_0 , as well as the optimal value function (right figure)

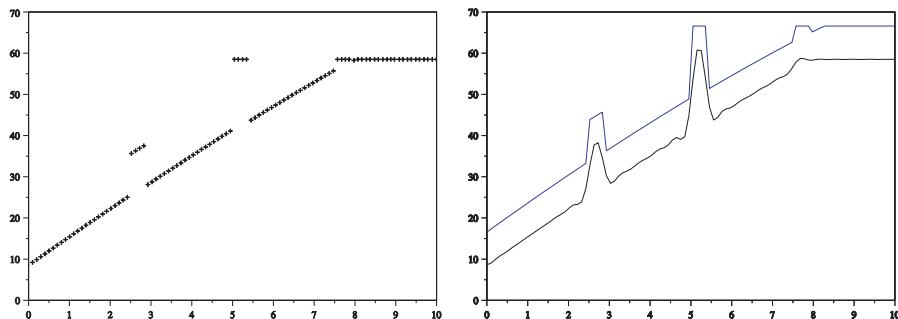


Figure 3.4. Values $\{LV_1(s_k)\}_{1 \leq k \leq K}$ (left figure), best fit $V_2 \in \mathcal{F}$ of LV_1 , as well as the optimal value function (right figure)

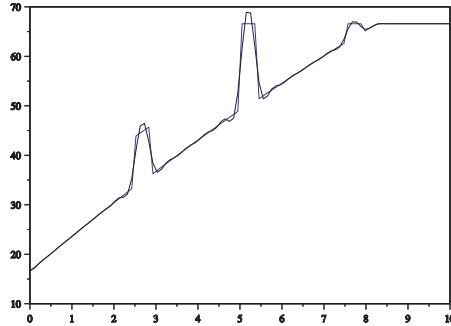


Figure 3.5. Approximation $V_{20} \in \mathcal{F}$ obtained at the 20th iteration and the optimal value function

3.3. Approximate policy iteration (API)

We now consider the *Approximate Policy Iteration* algorithm (API) [BER 96] which generalizes the policy iteration algorithm described in Chapter 1 to the use of function approximation. The algorithm is defined by the iteration of the two steps:

- *Approximate policy evaluation step*: for a policy π_n , an approximation V_n of the value function V^{π_n} is generated.
- *Policy improvement step*: A new policy π_{n+1} is generated as a greedy policy w.r.t. V_n , i.e. such that for all $s \in S$,

$$\pi_{n+1}(s) \in \arg \max_{a \in A} \left[r(s, a) + \gamma \sum_{s' \in S} p(s' | s, a) V_n(s') \right]. \quad (3.8)$$

Section 3.3.1 provides a bound on the performance loss $\|V^{\pi_n} - V^*\|_\infty$ resulting from the use of the policies generated by the API algorithm, instead of the optimal one, in terms of the value function approximation errors $\|V_n - V^{\pi_n}\|_\infty$. Section 3.3.2 describes the approximate policy evaluation step and section 3.3.3 details the case of *linear approximation* providing an extension of TD(λ) algorithm as well as least-squares methods, and finally an implementation using the Q-value function that does not need the *prior* knowledge of the transition probabilities is presented.

3.3.1. Analysis in L^∞ -norm of the API algorithm

We write π_n the policy derived by API algorithm at step n . Let V_n be an approximation of the value function V^{π_n} and π_{n+1} a policy greedy w.r.t. V_n (defined by (3.8)). The next result (stated in [BER 96, page 276]) gives a bound on the loss $V^* - V^{\pi_n}$ resulting from the use of the policy π_n instead of the optimal policy, as a function of the *approximation errors* $V_n - V^{\pi_n}$ in L^∞ -norm.

PROPOSITION 3.3. *We have*

$$\limsup_{n \rightarrow \infty} \|V^* - V^{\pi_n}\|_\infty \leq \frac{2\gamma}{(1-\gamma)^2} \limsup_{n \rightarrow \infty} \|V_n - V^{\pi_n}\|_\infty. \quad (3.9)$$

Proof. Define $e_n \stackrel{\text{def}}{=} V_n - V^{\pi_n}$ the *approximation error* of V^{π_n} by V_n , $g_n \stackrel{\text{def}}{=} V^{\pi_{n+1}} - V^{\pi_n}$ the *performance gain from one iteration to the next* and $l_n \stackrel{\text{def}}{=} V^* - V^{\pi_n}$ the *performance loss* due to the use of policy π_n instead of π^* (the latter being the quantity that we wish to bound).

If the approximation error is small, then the performance of the next policy cannot be much worse than the current one. Indeed, using vector notation, we have, componentwise,

$$\begin{aligned} g_n &= V^{\pi_{n+1}} - V^{\pi_n} \\ &= L_{\pi_{n+1}} V^{\pi_{n+1}} - L_{\pi_{n+1}} V^{\pi_n} + L_{\pi_{n+1}} V^{\pi_n} - L_{\pi_{n+1}} V_n \\ &\quad + L_{\pi_{n+1}} V_n - L_{\pi_n} V_n + L_{\pi_n} V_n - L_{\pi_n} V^{\pi_n} \\ &\geq \gamma P_{\pi_{n+1}} g_n - \gamma (P_{\pi_{n+1}} - P_{\pi_n}) e_n \end{aligned}$$

(where we used the definition of the Bellman operators and the fact that π_{n+1} is greedy w.r.t. V_n , thus $L_{\pi_{n+1}} V_n = LV_n \geq L_{\pi_n} V_n$, so $(I - \gamma P_{\pi_{n+1}})g_n \geq -\gamma(P_{\pi_{n+1}} - P_{\pi_n})e_n$). Moreover, the matrix $P_{\pi_{n+1}}$ is a stochastic matrix, thus its eigenvalues have a modulus less than or equal to one. And since $\gamma < 1$, the matrix $I - \gamma P_{\pi_{n+1}}$ does not have a null eigenvalue and thus is invertible. Since its inverse $(I - \gamma P_{\pi_{n+1}})^{-1}$, which also writes $\sum_{t \geq 0} (P_{\pi_{n+1}})^t$, only contains positive elements, we deduce that

$$g_n \geq -\gamma (I - \gamma P_{\pi_{n+1}})^{-1} (P_{\pi_{n+1}} - P_{\pi_n}) e_n. \quad (3.10)$$

Now we can bound the loss l_{n+1} at the next iteration in terms of the current loss l_n : since $L_{\pi^*} V_n \leq L_{\pi_{n+1}} V_n$, we have

$$\begin{aligned} l_{n+1} &= V^* - V^{\pi_{n+1}} \\ &= L_{\pi^*} V^* - L_{\pi^*} V^{\pi_n} + L_{\pi^*} V^{\pi_n} - L_{\pi^*} V_n + L_{\pi^*} V_n - L_{\pi_{n+1}} V_n \\ &\quad + L_{\pi_{n+1}} V_n - L_{\pi_{n+1}} V^{\pi_n} + L_{\pi_{n+1}} V^{\pi_n} - L_{\pi_{n+1}} V^{\pi_{n+1}} \\ &\leq \gamma [P_{\pi^*} l_n - P_{\pi_{n+1}} g_n + (P_{\pi_{n+1}} - P_{\pi^*}) e_n]. \end{aligned}$$

From which we deduce, using (3.10), that

$$\begin{aligned} l_{n+1} &\leq \gamma P_{\pi^*} l_n + \gamma [\gamma P_{\pi_{n+1}} (I - \gamma P_{\pi_{n+1}})^{-1} (P_{\pi_{n+1}} - P_{\pi_n}) + P_{\pi_{n+1}} - P_{\pi^*}] e_n \\ &\leq \gamma P_{\pi^*} l_n + \gamma [P_{\pi_{n+1}} (I - \gamma P_{\pi_{n+1}})^{-1} (I - \gamma P_{\pi_n}) - P_{\pi^*}] e_n. \end{aligned}$$

Writing $f_n \stackrel{\text{def}}{=} \gamma[P_{\pi_{n+1}}(I - \gamma P_{\pi_{n+1}})^{-1}(I - \gamma P_{\pi_n}) - P_{\pi^*}]e_n$, this last inequality may be rewritten

$$l_{n+1} \leq \gamma P_{\pi^*} l_n + f_n.$$

Taking the upper limit, we have

$$(I - \gamma P_{\pi^*}) \limsup_{n \rightarrow \infty} l_n \leq \limsup_{n \rightarrow \infty} f_n,$$

and using the same argument as before, we deduce that

$$\limsup_{n \rightarrow \infty} l_n \leq (I - \gamma P_{\pi^*})^{-1} \limsup_{n \rightarrow \infty} f_n. \quad (3.11)$$

This inequality, having only positive terms, is preserved in norm:

$$\begin{aligned} \limsup_{n \rightarrow \infty} \|l_n\|_\infty &\leq \frac{\gamma}{1-\gamma} \limsup_{n \rightarrow \infty} \|P_{\pi_{n+1}}(I - \gamma P_{\pi_{n+1}})^{-1}(I + \gamma P_{\pi_n}) + P_{\pi^*}\|_\infty \|e_n\|_\infty \\ &\leq \frac{\gamma}{1-\gamma} \left(\frac{1+\gamma}{1-\gamma} + 1 \right) \limsup_{n \rightarrow \infty} \|e_n\|_\infty = \frac{2\gamma}{(1-\gamma)^2} \limsup_{n \rightarrow \infty} \|e_n\|_\infty \end{aligned}$$

(where we used the fact that all stochastic matrices P have a norm $\|P\|_\infty = 1$). \square

3.3.2. Approximate policy evaluation

We now study the approximate policy evaluation step of API, i.e. for a given policy π we wish to build a good approximation of the value function V^π .

Let us remember that the value function V^π is defined as the expectation of the sum of discounted rewards, when we follow the policy π , see (3.1), and that V^π solves the Bellman equation: $V^\pi = L_\pi V^\pi$, where L_π is the Bellman operator defined by $L_\pi W \stackrel{\text{def}}{=} r_\pi + \gamma P_\pi W$, for any vector W .

Several methods enable us to find an approximation of V^π :

– *Iterative methods.* Analogously to the AVI algorithm, the operator L_π combined with an approximation operator \mathcal{A} is iterated according to $V_{n+1} = \mathcal{A}L_\pi V_n$. A result similar to that in Proposition 3.2 may easily be deduced.

– *Linear system methods.* Since the Bellman operator L_π is affine, V^π is the solution of a linear system: $(I - \gamma P_\pi)V^\pi = r_\pi$, and usual methods for solving approximately linear systems may apply. The main drawback of those methods is their computational complexity when the number of states is large.

– *Monte Carlo (MC) methods.* From the definition of the value function (3.1), an unbiased estimate of $V^\pi(s)$ is obtained by simulating $M \geq 1$ trajectories starting from the initial state s , using policy π , and taking the empirical average of the sum of discounted rewards obtained along those M trajectories. The variance of this estimate is $O(1/M)$. If we repeat this process from initial states $\{s_k\}_{1 \leq k \leq K}$ sampled from a distribution μ over S , and derive estimates $\{v_k\}$ of $\{V^\pi(s_k)\}$, then a good approximation in \mathcal{F} may be found by solving the least-squares fitting problem:

$$\arg \min_{f \in \mathcal{F}} \frac{1}{K} \sum_{k=1}^K (f(s_k) - v_k)^2.$$

Again, this is a projection of V^π onto \mathcal{F} (using the empirical L^2 -norm defined by the samples). There exist several variance reduction techniques that make it possible to improve the accuracy of the estimate. For example, from a first rough approximation of the value function, we may use an MC method estimating the residual which enables us to make a correction in order to improve the current estimate, and this process may be iterated (recursive Monte Carlo methods) [MUN 06].

– *Temporal Difference (TD) algorithms.* TD algorithms [SUT 88] are based on stochastic approximation algorithms [KUS 97] for finding the fixed-point of contractant operators. The generalization of those methods to approximate functions is not obvious in general (no convergence proof for $\lambda < 1$) but such approaches have been applied with great success, for example, in the game of backgammon [TES 95]. In the case of linear approximation (the method will be described in the next section), there exist theoretical guarantees on the resulting performance [TSI 96a].

– *Least-squares methods.* In the case of linear approximation, we may also consider very efficient methods based on least-squares fitting. This setting is considered in the next section.

3.3.3. Linear approximation and least-squares methods

In this section we consider the specific case of linear approximation. This case has been often studied in combination with TD(λ) algorithms [TSI 96a] (see section 2.5) or least-squares methods (*Least-Squares Temporal Differences* LSTD(0) [BRA 96], LSTD(λ) [BOY 99]) and applied, with success, to control problems [LAG 03].

The value function is approximated by a linear combination of K basis functions called *features* $(\phi_k)_{1 \leq k \leq K}$, which means that the approximation space \mathcal{F} is a vector space spanned by those features:

$$\mathcal{F} \stackrel{\text{def}}{=} \left\{ V_\alpha(s) \stackrel{\text{def}}{=} \sum_{k=1}^K \alpha_k \phi_k(s), \alpha \in \mathbb{R}^K \right\}.$$

Our goal is to find a parameter $\alpha \in \mathbb{R}^K$ such that V_α is close to V^π . Let us start with the direct extension of the TD(λ) algorithm previously seen (Chapter 2).

3.3.3.1. TD(λ)

The TD(λ) algorithm is defined like in Chapter 2. We use an eligibility trace $z \in \mathbb{R}^K$ with same dimension (K) as the parameter α , initialized to zero. From an initial state, we generate a trajectory (s_0, s_1, s_2, \dots) by using policy π . At each time t , we calculate the temporal difference for the current approximation V_α :

$$d_t \stackrel{\text{def}}{=} r(s_t, \pi(s_t)) + \gamma V_\alpha(s_{t+1}) - V_\alpha(s_t)$$

and update both the parameter α and the trace z :

$$\alpha_{t+1} = \alpha_t + \eta_t d_t z_t,$$

$$z_{t+1} = \lambda \gamma z_t + \phi(s_{t+1}),$$

where the η_t is a decreasing sequence and $\phi : S \rightarrow \mathbb{R}^K$ the vector of components ϕ_k .

This algorithm builds a sequence of approximations V_{α_t} that converges, under some ergodicity assumption of the corresponding Markov chain, to a function whose approximation error (distance to V^π) is bounded in terms of the minimum approximation error using functions in \mathcal{F} .

PROPOSITION 3.4 [TSI 96a]. *Assume that the steps (η_t) satisfy $\sum_{t \geq 0} \eta_t = \infty$ and $\sum_{t \geq 0} \eta_t^2 < \infty$, that there exists a distribution μ over S such that $\forall s, s' \in S, \lim_{t \rightarrow \infty} P(s_t = s' | s_0 = s) = \mu(s')$ and that the vectors $(\phi_k)_{1 \leq k \leq K}$ are linearly independent. Then α_t converges. Write α^* its limit. We have*

$$|V_{\alpha^*} - V^\pi|_\mu \leq \frac{1 - \lambda\gamma}{1 - \gamma} \inf_\alpha \|V_\alpha - V^\pi\|_\mu, \quad (3.12)$$

where $\|\cdot\|_\mu$ means the L^2 -norm weighted by the distribution μ , i.e. $\|f\|_\mu \stackrel{\text{def}}{=} [\sum_{s \in S} f(s)^2 \mu(s)]^{1/2}$.

When $\lambda = 1$, we obtain the result that the Monte Carlo estimate gives the best approximation of V^π in \mathcal{F} , i.e. the projection of V^π onto \mathcal{F} . Now, if $\lambda < 1$, the approximation quality deteriorates (introduction of a bias), but the variance of the estimate is smaller, thus approximating its value up to some given accuracy may be easier.

From its stochastic approximation aspect, TD(λ) is very costly in terms of experimental data, in the sense that it requires the observation of many transitions $s_t, a_t \rightarrow s_{t+1}$ (and several times the same transitions) in order to see the parameter α converging. This problem has motivated the introduction of least-squares methods which are much more parsimonious in terms of data.

3.3.3.2. Least-squares methods

Least-squares methods (*Least-Squares Temporal Differences* [BRA 96, BOY 99, LAG 03]) are based on the property that since the approximate function V_α is linear in α and the operator L_π is affine, then the Bellman mapping $\alpha \rightarrow R_\alpha \stackrel{\text{def}}{=} L_\pi V_\alpha - V_\alpha$ is also affine. Since the Bellman residual of the desired value function V^π is zero, i.e. $L_\pi V^\pi - V^\pi = 0$, it seems reasonable to search for a parameter α such that the Bellman residual R_α is as close as possible to 0. Two approaches are generally considered (see [SCH 03, MUN 03]):

– *The quadratic residual (QR) solution*: the parameter α_{QR} minimizes the norm of the Bellman residual R_α (see Figure 3.6):

$$\alpha_{\text{QR}} = \arg \min_{\alpha \in \mathbb{R}^K} \|R_\alpha\|_\mu,$$

for some norm $\|\cdot\|_\mu$.

– *The temporal difference (TD) solution*: the parameter α_{TD} is such that the residual $R_{\alpha_{\text{TD}}}$ is orthogonal to all features ϕ_k , thus to \mathcal{F} (see Figure 3.6). Thus $V_{\alpha_{\text{TD}}}$ is the fixed-point of the combined Bellman operator L_π followed by the orthogonal projection \mathcal{A} onto \mathcal{F} (according to the L^2 -norm weighted by the distribution μ). It is the same solution as that obtained by TD(λ) for $\lambda = 0$ (which justifies its name).

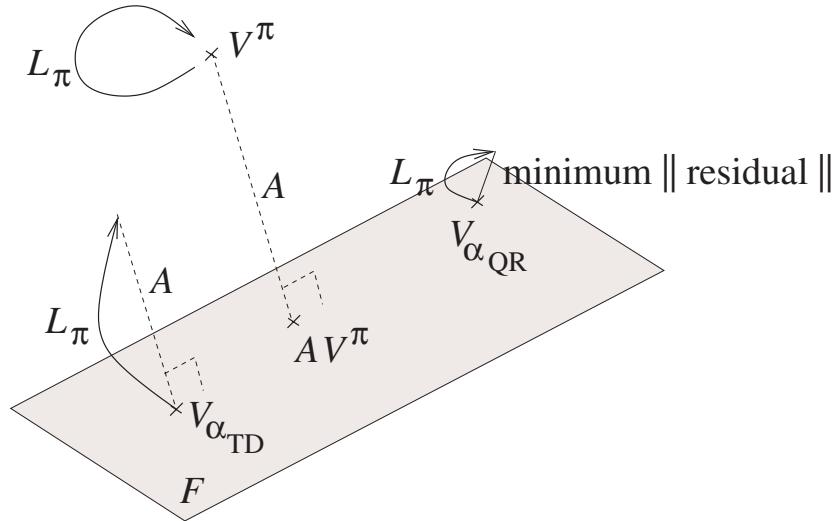


Figure 3.6. Approximation of the value function V^π in \mathcal{F} . The best possible approximation AV^π is the projection of V^π onto \mathcal{F} . The quadratic residual solution $V_{\alpha_{\text{QR}}}$ minimizes (in \mathcal{F}) the norm $\|L_\pi V_\alpha - V_\alpha\|$. The temporal difference solution $V_{\alpha_{\text{TD}}}$ is such that $\mathcal{A}L_\pi V_{\alpha_{\text{TD}}} = V_{\alpha_{\text{TD}}}$

In both cases, the parameter α is obtained by solving a linear system of size K (the number of parameters). Those methods are called *projection methods* [JUD 98] because we search for a parameter α such that the residual is orthogonal to a set of test functions (the features ϕ_k in the TD case, the derivatives $\partial_{\alpha_k} R_\alpha$ in the QR case). Let us consider the corresponding linear systems.

Quadratic residual solution. Since the mapping $\alpha \rightarrow R_\alpha$ is affine, the mapping $\alpha \rightarrow \|R_\alpha\|_\mu^2$ (for a L^2 norm weighted by μ) is quadratic. Its minimum (obtained by setting its gradient to zero) is thus the solution to the linear system: $A\alpha = b$, with the squared matrix A and the vector b (of size K) being defined as

$$\begin{cases} A_{ij} \stackrel{\text{def}}{=} \langle \phi_i - \gamma P_\pi \phi_i, \phi_j - \gamma P_\pi \phi_j \rangle_\mu, & \text{for } 1 \leq i, j \leq K \\ b_i \stackrel{\text{def}}{=} \langle \phi_i - \gamma P_\pi \phi_i, r_\pi \rangle_\mu, & \text{for } 1 \leq i \leq K, \end{cases} \quad (3.13)$$

where the inner product $\langle u, v \rangle_\mu$ of two functions u and v (defined on S) is defined by $\langle u, v \rangle_\mu \stackrel{\text{def}}{=} \sum_{s \in S} u(s)v(s)\mu(s)$.

This system always possesses a (unique) solution when $\mu > 0$ and the features ϕ_k are linearly independent (since in that case, the matrix A is positive definite). Let us notice that this problem may be considered as a linear regression problem with another set of basis functions $\{\psi_i \stackrel{\text{def}}{=} \phi_i - \gamma P_\pi \phi_i\}_{i=1..K}$ where it comes down to finding α that minimizes $\|\alpha \cdot \psi - r_\pi\|_\mu$. Usual supervised learning methods may thus be considered.

When μ is the stationary distribution associated with the policy π (this means that we have $\mu P_\pi = \mu$, i.e. $\mu(s) = \sum_{s'} p(s \mid s', \pi(s'))\mu(s')$ for all $s \in S$), we may deduce a bound on the approximation error $V^\pi - V_{\alpha_{QR}}$ in terms of the minimized residual or in terms of the distance between V^π and \mathcal{F} (in L^2 -norm with weight μ).

PROPOSITION 3.5. *We have*

$$\|V^\pi - V_{\alpha_{QR}}\|_\mu \leq \frac{1}{1-\gamma} \|R_{\alpha_{QR}}\|_\mu = \frac{1}{1-\gamma} \inf_{V_\alpha \in \mathcal{F}} \|L_\pi V_\alpha - V_\alpha\|_\mu \quad (3.14)$$

$$\leq \frac{1+\gamma}{1-\gamma} \inf_{V_\alpha \in \mathcal{F}} \|V^\pi - V_\alpha\|_\mu. \quad (3.15)$$

Proof. Since μ is the stationary distribution associated with π , we have the property that $\|P_\pi\|_\mu = 1$ (see, e.g. [TSI 96a]). In addition, for all α , we have

$$R_\alpha = L_\pi V_\alpha - V_\alpha = (I - \gamma P_\pi)(V^\pi - V_\alpha), \quad (3.16)$$

thus by considering α_{QR} , we deduce that $V^\pi - V_{\alpha_{\text{QR}}} = (I - \gamma P_\pi)^{-1} R_{\alpha_{\text{QR}}}$. Thus in norm,

$$\begin{aligned}\|V^\pi - V_{\alpha_{\text{QR}}}\|_\mu &\leq \|(I - \gamma P_\pi)^{-1}\|_\mu \|R_{\alpha_{\text{QR}}}\|_\mu \\ &\leq \sum_{t \geq 0} \gamma^t \|P_\pi\|_\mu^t \|R_{\alpha_{\text{QR}}}\|_\mu \leq \frac{1}{1 - \gamma} \|R_{\alpha_{\text{QR}}}\|_\mu,\end{aligned}$$

which proves (3.14). Moreover, taking the norm of (3.16), $\|R_\alpha\|_\mu \leq (1 + \gamma) \|V^\pi - V_\alpha\|_\mu$ and (3.15) follows. \square

Temporal difference solution. The TD solution, i.e. the fixed-point of the combined Bellman operator L_π followed by the projection \mathcal{A} onto \mathcal{F} (using norm $\|\cdot\|_\mu$), is obtained by solving the linear system $A\alpha = b$ with the matrix A and the vector b :

$$\begin{cases} A_{ij} \stackrel{\text{def}}{=} \langle \phi_i, \phi_j - \gamma P_\pi \phi_j \rangle_\mu, & \text{for } 1 \leq i, j \leq K, \\ b_i \stackrel{\text{def}}{=} \langle \phi_i, r_\pi \rangle_\mu, & \text{for } 1 \leq i \leq K. \end{cases} \quad (3.17)$$

We should be careful here, because the invertibility of the matrix A depends on the considered distribution μ . It is invertible when the features (ϕ_i) are linearly independent and when μ is the stationary distribution associated with the policy π [MUN 03]. In that case, the obtained solution is the same as that obtained by the TD(0) algorithm [SCH 03]. A consequence of Proposition 3.4 is the bound on the approximation error in terms of the distance between V^π and \mathcal{F} (in L^2 -norm weighted by μ):

$$\|V^\pi - V_{\alpha_{\text{TD}}}\|_\mu \leq \frac{1}{1 - \gamma} \inf_{V_\alpha \in \mathcal{F}} \|V^\pi - V_\alpha\|_\mu.$$

The generalization of least-squares methods to TD(λ) systems with $\lambda > 0$ (i.e. providing the TD(λ) solution) may be found in [BOY 99].

Let us notice that the size of the linear system is K , the number of coefficients (or number of features), which in general is much smaller than the number of states (the latter possibly being infinite). However, in order to use this method, we need to be able to calculate the result of the transition operator P_π applied to the features ϕ_k , as well as the inner products (weighted sum over all states for which $\mu > 0$). This problem is all the more troublesome when we consider the RL setting where the transition probabilities are unknown from the learning agent. In addition, when this evaluation method is used in an API algorithm, the lack of knowledge of those probabilities makes the policy improvement step (3.8) problematic. Those problems are solved by introducing, like in Chapter 2, an approximation of the state-action value function Q . We now explain this implementation.

3.3.3.3. Linear approximation of the state-action value function

Here we do not assume anymore that the transition probabilities $p(s' | s, a)$ are known from the learning agent. Rather, we now assume that the agent has access to a *generative model* [KAK 03] which enables us to sample a successor state $s' \sim p(\cdot | s, a)$ in any state s action a , and thus to generate trajectories when following a given policy.

Here we consider an approximation of the state-action value function (or Q-value function) Q^π [WAT 89, SUT 98] instead of the value function V^π . We remind that Q^π is defined, for any pair (s, a) , by the immediate reward when action a is chosen in state s plus the expected sum of discounted rewards when we use policy π afterwards, i.e. by using the definition of V^π :

$$Q^\pi(s, a) \stackrel{\text{def}}{=} r(s, a) + \gamma \sum_{s' \in S} p(s' | s, a) V^\pi(s').$$

The representations in terms of Q-value function or V-value function are equivalent: Q^π may be expressed using V^π as shown above, and symmetrically V^π is defined from Q^π according to $V^\pi(s) = Q^\pi(s, \pi(s))$. However, the benefit of the Q-value function is that the greedy policy is very easily deduced: for any $s \in S$, the greedy policy w.r.t. V^π in s is $\arg \max_{a \in A} Q^\pi(s, a)$.

In a way similar to what has been done in the previous section, we consider the linear approximation space:

$$\mathcal{F} \stackrel{\text{def}}{=} \left\{ Q_\alpha(s, a) \stackrel{\text{def}}{=} \sum_{k=1}^K \alpha_k \phi_k(x, a), \alpha \in \mathbb{R}^K \right\},$$

where the features ϕ_k are now defined over the product space $S \times A$.

The API algorithm using Q-value function representation is defined as follows: at round n , the approximate policy evaluation step calculates an approximation Q_n of Q^{π_n} ; the policy improvement step defines the next policy π_{n+1} as

$$\pi_{n+1}(s) \stackrel{\text{def}}{=} \arg \max_{a \in A} Q_n(s, a).$$

The two kinds of least-squares methods for policy evaluation apply immediately. In addition, the matrix A and vector b of the linear systems (3.13) and (3.17) may be estimated from observed data, as explained in [LAG 03]: a data base D is built from a set of transitions. At each transition (state s , action a) to state $s' \sim p(\cdot | s, a)$ with reward r , we add to D , the data (s, a, s', r) . Those data about transitions and rewards are built incrementally [BOY 99] or from the observation of trajectories induced by different policies [LAG 03], or also from data coming from prior knowledge about the state dynamics.

From this data base D , at each round n of the API algorithm, in order to approximately evaluate the policy π_n , we select in D the data $\{(s_m, a_m, s'_m, r_m)\}_{1 \leq m \leq M}$ such that the chosen action corresponds to the policy under evaluation (i.e. $a_m = \pi_n(s_m)$). From this selected set of data, we define an unbiased estimate of A and b for the TD system (3.17): for $1 \leq i, j \leq K$,

$$\begin{cases} \widehat{A}_{ij} \stackrel{\text{def}}{=} \frac{1}{M} \sum_{m=1}^M \phi_i(s_m, a_m) [\phi_j(s_m, a_m) - \gamma \phi_j(s'_m, \pi_n(s'_m))], \\ \widehat{b}_i \stackrel{\text{def}}{=} \frac{1}{M} \sum_{m=1}^M \phi_i(s_m, a_m) r_m. \end{cases}$$

Indeed, since the next states s' are generated according to $p(\cdot | s, a)$, we have the property that $\phi_j(s', \pi_n(s'))$ is an unbiased estimate of the operator P_{π_n} applied to ϕ_j in s , i.e. $\sum_{s' \in S} p(s' | s, a) \phi_j(s', \pi_n(s'))$. Moreover, from the law of large numbers, when the number of sampled data M is large, the empirical average over those M samples concentrates around the corresponding expectation, i.e. the inner products in (3.17). Thus \widehat{A} and \widehat{b} are unbiased and consistent estimates of A and b with a variance of order $O(1/M)$. When the system (3.17) is invertible, the solution $\widehat{\alpha}$ of the approximated system $\widehat{A}\widehat{\alpha} = \widehat{b}$ tends to the solution α_{TD} of the temporal difference system (3.17) when $M \rightarrow \infty$.

In a similar way, we could think that

$$\frac{1}{M} \sum_{m=1}^M [\phi_i(s_m, a_m) - \gamma \phi_i(s'_m, \pi_n(s'_m))] [\phi_j(s_m, a_m) - \gamma \phi_j(s'_m, \pi_n(s'_m))] \quad (3.18)$$

would provide an unbiased estimate of the element A_{ij} for the quadratic residual system (3.13). However this is not true (see [SUT 98, page 220] or [LAG 03, MUN 03]). This estimate would actually lead to a parameterization that tends to reduce the variance of the Q-value function of the successor states. The problem comes from the fact that the random variables $\phi_i(s', \pi_n(s'))$ and $\phi_j(s', \pi_n(s'))$ are correlated. There are several ways to recover an unbiased estimate of A and b :

– For each couple (s_m, a_m) , use two independent samples s'_m and s''_m drawn from $p(\cdot | s_m, a_m)$ by using the generative model, in order to decorrelate $\phi_i(s'_m, \pi_n(s'_m))$ and $\phi_j(s''_m, \pi_n(s''_m))$. Then, an unbiased estimate of A and b for the quadratic residual system (3.13) is

$$\begin{cases} \widehat{A}_{ij} \stackrel{\text{def}}{=} \frac{1}{M} \sum_{m=1}^M [\phi_i(s_m, a_m) - \gamma \phi_i(s'_m, \pi_n(s'_m))] [\phi_j(s_m, a_m) - \gamma \phi_j(s''_m, \pi_n(s''_m))], \\ \widehat{b}_i \stackrel{\text{def}}{=} \frac{1}{M} \sum_{m=1}^M [\phi_i(s_m, a_m) - \gamma \phi_i(s'_m, \pi_n(s'_m))] r_m. \end{cases}$$

– If we only possess a single sample s'_m per couple (s_m, a_m) (for instance because the data have been generated by following trajectories), we can consider the nearest-neighbor $s_{m'}$ of s_m such that the data $(s_{m'}, a_m, s'_{m'}, r_{m'})$ is in D . This introduces a bias (due to the distance between the samples s_m and $s_{m'}$) which depends on the smoothness of the transition probabilities and the density of points. The corresponding estimate of A and b for the system (3.13) is deduced analogously.

– A third approach, analyzed in [ANT 08], consists of modifying the criterion to be minimized by subtracting from the residual $(\|L_\pi Q_n - Q_n\|_\mu^2)$ a term $(\|\mathcal{A}L_\pi Q_n - L_\pi Q_n\|_\mu^2)$, where $\mathcal{A}L_\pi Q_n$ is the projection onto \mathcal{F} of $L_\pi Q_n$) whose estimate (from data) has the same bias as that of the residual (3.18), killing consequently the bias of the resulting estimate. We do not further describe this approach but simply mention the fact that in the specific case of linear approximation, the corresponding solution is the same as that of the temporal difference system, but this method extends naturally to non-linear approximation. See [FAR 09] for such an implementation using penalization.

The algorithms that have been presented in this section are efficient in terms of samples, since the observed transition and reward data $s, a \rightarrow s', r$ are memorized and enable to find directly the parameter α by solving a linear system. Incremental versions are of course possible.

3.4. Direct minimization of the Bellman residual

In addition to the usual value and policy iteration methods previously seen, we mention here a method which aims at directly minimizing the norm of the Bellman residual (for the operator L). The idea is very simple: since the optimal value function V^* is the fixed-point of the operator L , i.e. the norm of its residual $\|LV^* - V^*\|$ is zero, we may wish to find the minimum in a function space \mathcal{F} of the norm of the residual:

$$\inf_{V_\alpha \in \mathcal{F}} \|LV_\alpha - V_\alpha\|, \quad (3.19)$$

where $\|\cdot\|$ is a given norm.

The following result tells us that if the residual is well minimized (in L^∞ -norm), then the performance of the corresponding greedy policy is close to the optimum.

PROPOSITION 3.6 [WIL 93]. *Let V be a function defined over S and π a greedy policy w.r.t. V . The performance loss resulting from using the policy π instead of an optimal policy is bounded in terms of the Bellman residual of V as*

$$\|V^* - V^\pi\|_\infty \leq \frac{2}{1-\gamma} \|LV - V\|_\infty. \quad (3.20)$$

Proof. Since $L_\pi V = LV \geq L_{\pi^*} V$, we have

$$\begin{aligned} V^* - V^\pi &= L_{\pi^*} V^* - L_{\pi^*} V + L_{\pi^*} V - LV + LV - L_\pi V - L_\pi V^\pi \\ &\leq \gamma P_{\pi^*}(V^* - V^\pi + V^\pi - V) + \gamma P_\pi(V - V^\pi). \end{aligned}$$

Thus

$$(I - \gamma P_{\pi^*})(V^* - V^\pi) \leq \gamma(P_{\pi^*} - P_\pi)(V^\pi - V),$$

and from the property $V^\pi - V = (I - \gamma P_\pi)^{-1}(LV - V)$, it comes that

$$\begin{aligned} V^* - V^\pi &\leq \gamma(I - \gamma P_{\pi^*})^{-1}(P_{\pi^*} - P_\pi)(I - \gamma P_\pi)^{-1}(LV - V) \\ &= [(I - \gamma P_{\pi^*})^{-1} - (I - \gamma P_\pi)^{-1}](LV - V), \end{aligned} \tag{3.21}$$

thus in norm L^∞ ,

$$\begin{aligned} \|V^* - V^\pi\|_\infty &\leq \left[\|(I - \gamma P_{\pi^*})^{-1}\|_\infty + \|(I - \gamma P_\pi)^{-1}\|_\infty \right] \|LV - V\|_\infty \\ &\leq \frac{2}{1 - \gamma} \|LV - V\|_\infty. \end{aligned} \quad \square$$

Thus we have a performance guarantee for policies greedy w.r.t. functions V_α efficiently minimizing the norm of the residual. However the minimization problem (3.19) may be hard to solve, even in the case of a linear parameterization, since the operator L is not affine (in opposition to the operator L_π) because of the maximum operator over actions. There do not exist simple methods for finding the global minimum (in opposition to the previous case using the operator L_π and a linear approximation where the norm of the residual to be minimized $\alpha \rightarrow \|L_\pi V_\alpha - V_\alpha\|_\mu$ was a quadratic mapping in α). However local optimization techniques exist (such as gradient methods, where the direction opposite to the gradient $\nabla_\alpha \|LV_\alpha - V_\alpha\|_\mu^2$ is followed), for example, neural networks are commonly used in practice to minimize the L^2 -norm of the residual, although there are no global convergence guarantee.

3.5. Towards an analysis of dynamic programming in L^p -norm

We have provided several performance bounds ((3.6) and (3.9), respectively, for the AVI and API algorithms) in terms of the approximation errors in L^∞ -norm. However, as illustrated in section 3.2.1, usual supervised learning algorithms solve an optimization problem using an empirical L^p -norm (with $p = 1$ or 2). Hence, bounds on the approximation errors (such as (3.12) or (3.15)) are of L^p kind whereas those on error propagation in dynamic programming are of L^∞ kind.

The fundamental problem of the analysis of DP with function approximation lies in the different tools that are used to analyze DP and approximation theory:

– Usual DP analysis makes use of the norm L^∞ , which is very natural since the Bellman operators L and L_π are contraction operators in this norm. The value iteration, policy iteration, and their RL variants are all based on this property.

– Function approximation makes almost exclusively use of L^p -norms: for example, least-squares methods, neural networks, Support Vector Machines and other kernel methods, etc.

The different norms explain the difficulty to analyze DP combined with function approximation. For example, if one considers the AVI algorithm defined by (3.3), the Bellman operator L is a contraction in L^∞ -norm, the approximation operator \mathcal{A} is a non-expansion in L^2 -norm (in the case of an orthogonal projection onto \mathcal{F}), but one cannot say anything about the combined operator $\mathcal{A}L$. The L^∞ analysis of this algorithm, illustrated by the result (3.6), provides a performance bound in terms of the uniform approximation error $\|\varepsilon_n\|_\infty$, which is very difficult to control, especially in large-scale problems. This result is hardly useful in practice. Moreover, most of the approximation operators and supervised learning algorithms solve a minimization problem using an empirical L^1 or L^2 norm, for example, (3.4). The uniform error $\|\varepsilon_n\|_\infty$ is difficult to bound in terms of the error actually minimized by the empirical L^p problem (3.4).

An L^p analysis of the AVI algorithm which would take into account the approximation errors in L^p -norm would enable us to evaluate the performance in terms of the empirical errors and a capacity term (such as usual Vapnik-Chervonenkis dimension or covering numbers [POL 84, VAP 98]) of the considered function space \mathcal{F} . Indeed, following usual results in statistical learning, the approximation error in L^p -norm (the so-called *generalization error*) may be upper bounded by the empirical error (or *learning error*) actually minimized plus a capacity term of \mathcal{F} . In addition, since most of the approximation operators and supervised learning algorithms provide good fits by minimizing a L^p -norm, it appears essential to analyze the performance of dynamic programming using this same norm.

First steps along this direction are reported in [MUN 07, MUN 08] and briefly described in the next two sections.

3.5.1. Intuition of an L^p analysis in dynamic programming

First let us remind the definition of L^p -norm (for $p \geq 1$) weighted by a distribution μ : $\|f\|_{p,\mu} \stackrel{\text{def}}{=} [\sum_{s \in S} \mu(s)|f(s)|^p]^{1/p}$. When $p = 2$, we use the simplified notation $\|f\|_\mu$.

The underlying intuition of an L^p analysis of DP is simple and comes from componentwise bounds. Indeed, let f and g be two positive functions defined over S ,

such that $f \leq Pg$, with P being a stochastic matrix. Of course, this implies that $\|f\|_\infty \leq \|g\|_\infty$ (since $\|P\|_\infty = 1$), but in addition, if ν and μ are distributions over S such that $\nu P \leq C\mu$ (we should interpret the notation νP as the matrix product of the row vector ν by the matrix P) for some constant $C \geq 1$, then we may also deduce that $\|f\|_{p,\nu} \leq C^{1/p} \|g\|_{p,\mu}$.

Indeed, we have

$$\begin{aligned}\|f\|_{p,\nu}^p &= \sum_{s \in S} \nu(s) |f(s)|^p \leq \sum_{s \in S} \nu(s) \left| \sum_{s' \in S} P(s' | s) g(s') \right|^p \\ &\leq \sum_{s \in S} \nu(s) \sum_{s' \in S} P(s' | s) |g(s')|^p \\ &\leq C \sum_{s' \in S} \mu(s') |g(s')|^p = C \|g\|_{p,\mu}^p,\end{aligned}$$

where we used Jensen's inequality (i.e. convexity of $x \rightarrow |x|^p$) at the second line.

For instance, the componentwise bound (3.21) enables us to deduce the bound (3.20) in terms of the L^∞ -norm of the Bellman residual. From this same bound (3.21), we may also deduce a bound in terms of the L^p -norm of the Bellman residual:

$$\|V^* - V^\pi\|_{p,\nu} \leq \frac{2}{1-\gamma} C(\nu, \mu)^{1/p} \|LV - V\|_{p,\mu}, \quad (3.22)$$

where ν and μ are two distributions over S and $C(\nu, \mu)$ is a constant that measures the concentrability (relative to μ) of the discounted future state distribution (given that the initial state is sampled from ν) of the MDP (see [MUN 07, MUN 08] for a precise definition and the link with Lyapunov exponents in dynamical systems). This bound is tighter than the L^∞ bound since when $p \rightarrow \infty$ we recover the bound (3.20).

Similar results may be derived for the AVI and API algorithms. For illustration, for the AVI algorithm, we may prove the componentwise bound:

$$\begin{aligned}&\limsup_{n \rightarrow \infty} V^* - V^{\pi_n} \\ &\leq \limsup_{n \rightarrow \infty} (I - \gamma P_{\pi_n})^{-1} \left(\sum_{k=0}^{n-1} \gamma^{n-k} [(P_{\pi^*})^{n-k} + P_{\pi_n} P_{\pi_{n-1}} \cdots P_{\pi_{k+2}} P_{\pi_{k+1}}] |\varepsilon_k| \right),\end{aligned}$$

with $\varepsilon_k = LV_k - V_{k+1}$ (approximation error at round k). By taking the L^∞ -norm, this bound gives (3.6). But we may also deduce the L^p -norm bound:

$$\limsup_{n \rightarrow \infty} \|V^* - V^{\pi_n}\|_{p,\nu} \leq \frac{2\gamma}{(1-\gamma)^2} C(\nu, \mu)^{1/p} \limsup_{n \rightarrow \infty} \|\varepsilon_n\|_{p,\mu}. \quad (3.23)$$

Similarly for the API algorithm, the componentwise bound (3.11) enables us to deduce the L^∞ result (3.9) as previously shown, and also to derive the following L^p bound:

$$\begin{aligned} & \limsup_{n \rightarrow \infty} \|V^* - V^{\pi_n}\|_{p,\nu} \\ & \leq \frac{2\gamma}{(1-\gamma)^2} C(\nu, \mu)^{1/p} \limsup_{n \rightarrow \infty} \|V_n - V^{\pi_n}\|_{p,\mu}. \end{aligned} \quad (3.24)$$

This L^p analysis in DP enables us to establish a link with statistical learning and deduce PAC bounds (*Probably Approximately Correct*) [VAL 84] for RL algorithms.

3.5.2. PAC bounds for RL algorithms

We now provide a PAC bound (detailed in [MUN 08]) for an RL algorithm based on AVI. We consider a large state space, for example, continuous. At each iteration, the approximation operator consists of performing an empirical regression based on a finite number N of sampled states, where in each state, the Bellman operator is estimated by using M transition samples obtained from the generative model.

More precisely, we repeat K approximate policy iteration steps (3.3). At round $1 \leq k < K$, $V_k \in \mathcal{F}$ denotes the current value function approximation, and a new approximation V_{k+1} is obtained as follows. We sample N states $\{s_n\}_{1 \leq n \leq N} \in S$ independently from a distribution μ over S . For each state s_n and each action $a \in A$, we generate M successor states $\{s'_{n,a,m} \sim p(\cdot | s_n, a)\}_{1 \leq m \leq M}$ and we formulate an empirical estimate of the Bellman operator applied to V_k in s_n :

$$v_n \stackrel{\text{def}}{=} \max_{a \in A} [r(s_n, a) + \gamma \frac{1}{M} \sum_{m=1}^M V_k(s'_{n,a,m})].$$

Finally V_{n+1} is defined as the solution to this L^p fitting problem:

$$V_{n+1} \stackrel{\text{def}}{=} \arg \min_{f \in \mathcal{F}} \frac{1}{N} \sum_{n=1}^N |f(s_n) - v_n|^p.$$

At round K , we write π_K a greedy policy w.r.t. V_K and we wish to evaluate its performance (compared to the optimal performance) in terms of the number of iterations K , the number of sampled states N , the number of successors M , the capacity of the function space \mathcal{F} , the smoothness of the MDP (concentrability constant $C(\nu, \mu)$ in the bound (3.23)) and the Bellman residual $d(L\mathcal{F}, \mathcal{F})$ of the function space \mathcal{F} . We have the following result.

PROPOSITION 3.7 [MUN 08]. *For any $\delta > 0$, with probability at least $1 - \delta$, we have*

$$\begin{aligned} \|V^* - V^{\pi_K}\|_{p,\nu} &\leq \frac{2\gamma}{(1-\gamma)^2} C(\nu, \mu)^{1/p} d(L\mathcal{F}, \mathcal{F}) + O(\gamma^K) \\ &\quad + O\left\{\left(\frac{V_{\mathcal{F}^+} \log(1/\delta)}{N}\right)^{1/2p} + \left(\frac{\log(1/\delta)}{M}\right)^{1/2}\right\}, \end{aligned} \quad (3.25)$$

where $d(L\mathcal{F}, \mathcal{F}) \stackrel{\text{def}}{=} \sup_{g \in \mathcal{F}} \inf_{f \in \mathcal{F}} \|Lg - f\|_{p,\mu}$ is the Bellman residual of the space \mathcal{F} , and $V_{\mathcal{F}^+}$ a capacity measure (the pseudo-dimension) of \mathcal{F} [HAU 95].

The four terms of this bound, respectively, mean the following:

- The Bellman residual $d(L\mathcal{F}, \mathcal{F})$ generalizes the notion of Bellman residual to a class of functions \mathcal{F} . It measures how well the function space \mathcal{F} makes it possible to approximate functions Lg , $g \in \mathcal{F}$ (images by the Bellman operator of functions in \mathcal{F}). This term is analogous to the notion of distance between the target function and \mathcal{F} in the case of regression. However, here there is no target function to approximate since our goal is to approximate the fixed-point of the Bellman operator with functions in \mathcal{F} . When the MDP is smooth (e.g. if the transition probabilities $p(s' \cdot \cdot, a)$ and the reward function $r(\cdot, a)$ are Lipschitzian), we can show that the term $d(L\mathcal{F}, \mathcal{F})$ decreases when the function space \mathcal{F} increases (since then the Bellman operator possesses a regularizing effect and the space $L\mathcal{F}$ is a subspace of Lipschitz functions with Lipschitz coefficient independent of \mathcal{F} , see [MUN 08]).

- The term coming from the finite number K of iterations tends to 0 exponentially fast.

- Two terms of order $O((V_{\mathcal{F}^+}/N)^{1/2p}) + O(1/\sqrt{M})$ bound the estimation error in terms of the number of samples N and M .

This result states that, if we use enough samples (N, M, K) , the performance of this algorithm can be arbitrarily close (up to a constant) to the Bellman residual of the space \mathcal{F} , which itself can be made arbitrarily small by considering a sufficiently rich function approximation class. This kind of bound is analogous to the bounds obtained in supervised learning [GYÖ 02] and allows us to analyze the stability and convergence rate of AVI based on samples, especially the following:

- It enables us to understand the *bias-variance trade-off* in approximate dynamic programming. Indeed, the performance bound (3.25) contains a bias term, the Bellman residual $d(L\mathcal{F}, \mathcal{F})$, that decreases when \mathcal{F} gets richer, and a variance term, due to the capacity $V_{\mathcal{F}^+}$ of \mathcal{F} , that increases with the richness of \mathcal{F} , but which may be reduced by using a larger number of samples N (in order to avoid overfitting).

- It enables us to understand the well-known counter-examples (of divergence of the algorithm) mentioned in previous works (in particular, the Bellman residual

$d(L\mathcal{F}, \mathcal{F})$ of the function spaces used in [BAI 95, TSI 96b] is infinite) and to be able to predict the behavior of this kind of algorithms in terms of the characteristics of the MDP, the capacity and richness of the function space \mathcal{F} and the number of samples.

3.6. Conclusions

The results mentioned in the previous section are a direct consequence of statistical learning results combined with the L^p -norm analysis of DP briefly introduced. Many extensions are possible, such as RL methods based on API (like the LSPI of [LAG 03]) even when samples are obtained by the observation of a unique trajectory [ANT 08]. In parallel to these theoretical works, we should mention the important diversity and quantity of works about the use of approximate representations of the value function for the purpose of decision making. Kernel methods [SCH 01] have been applied to DP and RL [ORM 02, RAS 04], as well as decision trees [WAN 99, ERN 05], neural networks [BER 96, COU 02], Bayesian approaches [DEA 98] and factored representations [GUE 01, KOL 00, DEG 06] (see Chapter 4) only to cite a few.

This chapter focused on approximate representation of the *value function*, following the dynamic programming principle initiated by Bellman [BEL 57]. Let us finally mention that there exists a completely different approach, somehow dual to this one, which consists of searching directly an approximation of the optimal policy by considering a class of parameterized policies. Those methods, now called *direct policy search* originate from Pontryagin's principle [PON 62] which sets necessary conditions for optimality by considering sensitivity analysis of the performance measure with respect to the policy parameters. This approach is the object of Chapter 5.

3.7. Bibliography

- [ANT 08] ANTOS A., SZEPESVÁRI C. and MUNOS R., “Learning near-optimal policies with Bellman-residual minimization based fitted policy iteration and a single sample path”, *Machine Learning*, vol. 71, no. 1, pp. 89–129, 2008.
- [ATK 97] ATKESON C. G., MOORE A. W. and SCHAAL S. A., “Locally weighted learning”, *AI Review*, vol. 11, pp. 11–73, 1997.
- [BAI 95] BAIRD L. C., “Residual algorithms: reinforcement learning with function approximation”, *Proceedings of the 12th International Conference on Machine Learning (ICML'95)*, Morgan Kaufmann, San Francisco, CA, pp. 30–37, 1995.
- [BEL 57] BELLMAN R. E., *Dynamic Programming*, Princeton University Press, Princeton, NJ, 1957.
- [BEL 59] BELLMAN R. E. and DREYFUS S. E., “Functional approximation and dynamic programming”, *Math. Tables and other Aids Comp.*, vol. 13, pp. 247–251, 1959.

- [BER 96] BERTSEKAS D. P. and TSITSIKLIS J., *Neuro-Dynamic Programming*, Athena Scientific, Belmont, MA, 1996.
- [BOY 99] BOYAN J., “Least-squares temporal difference learning”, *Proceedings of the 16th International Conference on Machine Learning (ICML’99)*, pp. 49–56, 1999.
- [BRA 96] BRADTKE S. and BARTO A., “Linear least-squares algorithms for temporal difference learning”, *Machine Learning*, vol. 22, pp. 33–57, 1996.
- [COU 02] COULOM R., Reinforcement learning using neural networks, with applications to motor control, PhD thesis, Institut National Polytechnique de Grenoble, 2002.
- [CRI 96] CRITES R. H. and BARTO A. G., “Improving elevator performance using reinforcement learning”, *Advances in Neural Information Processing Systems 8 (NIPS’95)*, pp. 1017–1023, 1996.
- [DAV 97] DAVIES G. M., MALLAT S. and AVELLANEDA M., “Adaptive greedy approximations”, *Constructive Approximation*, vol. 13, pp. 57–98, 1997.
- [DEA 98] DEARDEN R., FRIEDMAN N. and RUSSELL S., “Bayesian Q-learning”, *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI’98)*, pp. 761–768, 1998.
- [DEG 06] DEGRIS T., SIGAUD O. and WUILLEMIN P.-H., “Learning the structure of factored markov decision processes in reinforcement learning problems”, *Proceedings of the 23rd International Conference on Machine Learning (ICML’06)*, Pittsburgh, PA, pp. 257–264, 2006.
- [DEV 98] DEVORE R., “Nonlinear approximation”, *Acta Numerica*, vol. 7, pp. 51–150, 1998.
- [ERN 05] ERNST D., GEURTS P. and WEHENKEL L., “Tree-based batch mode reinforcement learning”, *Journal of Machine Learning Research*, vol. 6, pp. 503–556, 2005.
- [FAR 09] FARAHMAND A. M., GHAVAMZADEH M., SZEPESVARI C. and MANNOR S., “Regularized fitted Q-iteration for planning in continuous-space markovian decision problems”, *Proceedings of the American Control Conference*, St. Louis, MO, 2009.
- [GOR 95] GORDON G. J., “Stable function approximation in dynamic programming”, PRIEDITIS A. and RUSSELL S., Eds., *Proceedings of the 12th International Conference on Machine Learning (ICML’95)*, Morgan Kaufmann, San Francisco, CA, pp. 261–268, 1995.
- [GOS 04] GOSAVI A., “A reinforcement learning algorithm based on policy iteration for average reward: Empirical results with yield management and convergence analysis”, *Machine Learning*, vol. 55, pp. 5–29, 2004.
- [GUE 01] GUESTRIN C., KOLLER D. and PARR R., “Max-norm projections for factored MDPs”, *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI’01)*, Seattle, WA, pp. 673–680, 2001.
- [GYÖ 02] GYÖRFI L., KOHLER M., KRZYŻAK A. and WALK H., *A Distribution-Free Theory of Nonparametric Regression*, Springer Series in Statistics, Springer-Verlag, New York, 2002.
- [HAS 01] HASTIE T., TIBSHIRANI R. and FRIEDMAN J., *The Elements of Statistical Learning*, Springer Series in Statistics, Springer-Verlag, New York, 2001.

- [HAU 95] HAUSSLER D., “Sphere packing numbers for subsets of the Boolean n -cube with bounded Vapnik-Chervonenkis dimension”, *J. Combin. Theory Ser. A*, vol. 69, pp. 217–232, 1995.
- [JUD 98] JUDD K., *Numerical Methods in Economics*, MIT Press, Cambridge, MA, 1998.
- [KAK 03] KAKADE S., On the sample complexity of reinforcement learning, PhD thesis, Gatsby Computational Neuroscience Unit, University College London, 2003.
- [KOL 00] KOLLER D. and PARR R., “Policy iteration for factored MDPs”, *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI'00)*, Morgan Kaufman, San Francisco, CA, pp. 326–334, 2000.
- [KUS 97] KUSHNER H. J. and YIN G. G., *Stochastic Approximation Algorithms and Applications*, Springer-Verlag, New York, 1997.
- [LAG 03] LAGOUAKIS M. G. and PARR R., “Least-squares policy iteration”, *Journal of Machine Learning Research*, vol. 4, pp. 1107–1149, 2003.
- [MAH 97] MAHADEVAN S., MARCHALLECK N., DAS T. and GOSAVI A., “Self-improving factory simulation using continuous-time average-reward reinforcement learning”, *Proceedings of the 14th International Conference on Machine Learning (ICML'97)*, Morgan Kaufmann, San Mateo, CA, pp. 202–210, 1997.
- [MAL 97] MALLAT S., *A Wavelet Tour of Signal Processing*, Academic Press, San Diego, CA, 1998.
- [MUN 03] MUNOS R., “Error bounds for approximate policy iteration”, *Proceedings of the 19th International Conference on Machine Learning (ICML'03)*, pp. 560–567, 2003.
- [MUN 06] MUNOS R., “Geometric variance reduction in Markov chains: Application to value function and gradient estimation”, *Journal of Machine Learning Research*, vol. 7, pp. 413–427, 2006.
- [MUN 07] MUNOS R., “Performance bounds in L_p -norm for approximate value iteration”, *SIAM J. Control Optim.*, vol. 46, pp. 541–561, 2007.
- [MUN 08] MUNOS R. and SZEPESVÁRI C., “Finite-time bounds for fitted value iteration”, *Journal of Machine Learning Research*, vol. 9, pp. 815–857, 2008.
- [ORM 02] ORMONEIT D. and SEN S., “Kernel-based reinforcement learning”, *Machine Learning*, vol. 49, pp. 161–178, 2002.
- [POL 84] POLLARD D., *Convergence of Stochastic Processes*, Springer Series in Statistics, Springer Verlag, New York, 1984.
- [PON 62] PONTRYAGIN L. S., BOLTYANSKII V. G., GAMKRLEDZE R. V. and MISCHENKO E. F., *The Mathematical Theory of Optimal Processes*, Interscience Publishers, New York, 1962.
- [PUT 94] PUTERMAN M. L., *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, John Wiley & Sons, New York, 1994.
- [RAS 04] RASMUSSEN C. E. and KUSS M., “Gaussian processes in reinforcement learning”, THRUN S., SAUL L. K. and SCHÖLKOPF B., Eds., *Advances in Neural Information Processing Systems 16 (NIPS'03)*, MIT Press, Cambridge, MA, pp. 751–759, 2004.

- [REE 77] REETZ D., “Approximate solutions of a discounted Markovian decision problem”, *Dynamische Optimierung*, Bonner Math. Schriften, no. 98, Inst. Angew. Math., Univ. Bonn, Bonn, pp. 77–92, 1977.
- [RUS 96] RUST J., “Numerical dynamic programming in economics”, AMMAN H., KENDRICK D. and RUST J., Eds., *Handbook of Computational Economics*, Handbooks in Econom., vol. 13, North Holland, Amsterdam, pp. 619–729, 1996.
- [SAM 59] SAMUEL A. L., “Some studies in machine learning using the game of checkers”, *IBM Journal of Research and Development*, vol. 3, no. 3, pp. 210–229, 1959.
- [SAM 67] SAMUEL A. L., “Some studies in machine learning using the game of checkers. II – Recent progress”, *IBM Journal on Research and Development*, vol. 11, no. 6, pp. 601–617, 1967.
- [SCH 01] SCHÖLKOPF B. and SMOLA A. J., *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*, MIT Press, Cambridge, MA, 2001.
- [SCH 03] SCHOKNECHT R., “Optimality of reinforcement learning algorithms with linear function approximation”, *Advances in Neural Information Processing Systems 15 (NIPS'02)*, 2003.
- [SIN 97] SINGH S. P. and BERTSEKAS D. P., “Reinforcement learning for dynamic channel allocation in cellular telephone systems”, *Advances in Neural Information Processing Systems 9 (NIPS'96)*, 1997.
- [SUT 88] SUTTON R. S., “Learning to predict by the method of temporal differences”, *Machine Learning*, vol. 3, no. 1, pp. 9–44, 1988.
- [SUT 98] SUTTON R. S. and BARTO A. G., *Reinforcement Learning: An Introduction*, Bradford Book, MIT Press, Cambridge, MA, 1998.
- [TES 95] TESAURO G. J., “Temporal difference learning and TD-Gammon”, *Communication of the ACM*, vol. 38, pp. 58–68, 1995.
- [TSI 96a] TSITSIKLIS J. N. and VAN ROY B., An analysis of temporal difference learning with function approximation, Report no. LIDS-P-2322, MIT, 1996.
- [TSI 96b] TSITSIKLIS J. N. and VAN ROY B., “Feature-based methods for large scale dynamic programming”, *Machine Learning*, vol. 22, pp. 59–94, 1996.
- [VAL 84] VALIANT L. G., “A theory of the learnable”, *Communications of the ACM*, vol. 27, no. 11, pp. 1134–1142, 1984.
- [VAP 97] VAPNIK V., GOLOWICH S. E. and SMOLA A., “Support vector method for function approximation, regression estimation and signal processing”, *Advances in Neural Information Processing Systems 9 (NIPS'96)*, pp. 281–287, 1997.
- [VAP 98] VAPNIK V., *Statistical Learning Theory*, John Wiley & Sons, New York, 1998.
- [WAN 99] WANG X. and DIETTERICH T. G., “Efficient value function approximation using regression trees”, *Proceedings of the IJCAI Workshop on Statistical Machine Learning for Large-Scale Optimization*, Stockholm, Sweden, 1999.
- [WAT 89] WATKINS C. J. C. H., Learning from delayed rewards, PhD thesis, Cambridge University, Psychology Department, Cambridge, UK, 1989.

[WIL 93] WILLIAMS R. J. and BAIRD L. C., Tight performance bounds on greedy policies based on imperfect value functions, Report no. NU-CCS-93-14, Northeastern University, College of Computer Science, Boston, MA, November 1993.

[ZHA 95] ZHANG W. and DIETTERICH T., “A reinforcement learning approach to job-shop scheduling”, *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI'95)*, Montreal, Canada, pp. 1114–1120, 1995.

Chapter 4

Factored Markov Decision Processes

4.1. Introduction

Solution methods described in the MDP framework (Chapters 1 and 2) share a common bottleneck: they are not adapted to solve large problems. Indeed, using non-structured representations requires an explicit enumeration of the possible states in the problem. The complexity of this enumeration grows exponentially with the number of variables in the problem.

EXAMPLE 4.1. In the case of the car to be maintained, the number of possible states of a car can be huge. For instance, each part of the car can have its one wearout state. The idea of factored representations is that some parts of this huge state do not depend on each other and that this structure can be exploited to derive a more compact representation of the global state and obtain more efficiently an optimal policy. For instance, changing the oil in the car should have no effect on the breaks, thus we do not need to care about the state of the breaks to determine an optimal oil changing policy.

This chapter aims to describe FMDPs (Factored Markov Decision Processes), first proposed by [BOU 95, BOU 99]. FMDPs are an extension of MDPs that makes it possible to represent the transition and the reward functions of some problems compactly (compared to an explicit enumeration of state-action pairs). First, we describe the framework and how problems are modeled (section 4.2). Then we describe different planning methods able to take advantage of the structure of the

Chapter written by Thomas DEGRIS and Olivier SIGAUD.

problem to calculate optimal or near-optimal solutions (section 4.3). Finally, we conclude in section 4.4 and present some perspectives.

4.2. Modeling a problem with an FMDP

4.2.1. Representing the state space

It is intuitive to describe a problem by a set of observations whose values describe the current status of the environment. So, the state s can be described as a multivariate random variable $\mathbf{X} = (X_1, \dots, X_n)$ where each variable X_i can take a value in its domain $\text{DOM}(X_i)$.¹ Then, a state becomes an instantiation of each random variable X_i and can be written as a vector $\mathbf{x} = (x_1, \dots, x_n)$ such that $\forall i, x_i \in \text{DOM}(X_i)$. We denote by $\text{DOM}(\mathbf{X})$ the set of possible instantiations for the multivariate variable \mathbf{X} . Consequently, the state space S of the MDP is defined by $S = \text{DOM}(\mathbf{X})$.

With such representations, states are not atomic and it becomes possible to exploit some structures of the problem. In FMDPs, such representation is mainly used to represent the problem compactly and to reduce the complexity of the computation of the solution. More precisely, FMDPs exploit *function-specific independence* to represent compactly the transition and the reward functions. Moreover, FMDPs are also appropriate to exploit two other properties related to the structure of the problem, that is, *context-specific independence* and linear approximation.

To illustrate the FMDP framework, we are going to use a well-known example in the literature named *Coffee Robot* (section 4.2.2), first proposed by [BOU 00]. Using this example, we are then going to describe the decomposition of the transition and the reward functions (section 4.2.3) with a formalization of function-specific independencies. Section 4.2.4 proposes a formalization of context-specific independencies.

4.2.2. The Coffee Robot example

A robot must go to a café to buy a cup of coffee for its owner who is located at its office. When it is raining, it must get an umbrella to stay dry when going to the café. The state of the system is composed of six binary variables² X_i , where $\text{DOM}(X_i) = \{0, 1\}$ (corresponding, respectively, to False and True). These variables are the following:

1. For convenience, we will sometimes see such multivariate random variables as sets of random variables.
 2. This example uses binary variables. However, nothing prevents from using a random variable X_i , where $|\text{DOM}(X_i)| > 2$.

- \mathcal{H} : Does the owner have a coffee?
- \mathcal{C} : Does the robot have a coffee?
- \mathcal{W} : Is the robot wet?
- \mathcal{R} : Is it raining?
- \mathcal{U} : Does the robot have an umbrella?
- \mathcal{O} : Is the robot in the office?

For instance, the vector $[\mathcal{H} = 0, \mathcal{C} = 1, \mathcal{W} = 0, \mathcal{R} = 1, \mathcal{U} = 0, \mathcal{O} = 1]$ represents the state of the *Coffee Robot* problem where the owner does not have a coffee, the robot has a coffee, it is not wet, it is raining, the robot does not have an umbrella and it is located in the office. This problem being composed of 6 binary variables, there is $2^6 = 64$ possible states.

In this problem, four actions are available to the robot:

- \mathcal{G} o: Move to the other location.
- \mathcal{B} uyC: Buy a coffee: the robot gets one only if it is in the café.
- \mathcal{D} elC: Deliver coffee: the owner will get a coffee only if the robot is located in the office and it has a coffee.
- \mathcal{G} etU: Get an umbrella: the robot will get one only if it is located in the office.

Actions can be noisy to represent stochastic problems. For instance, when the robot gives the coffee, its owner will get his coffee only with a given probability (the cup may fall). Thus, when the action \mathcal{D} elC is executed in the state $\mathbf{x} = [\mathcal{C} = 0, \mathcal{H} = 1, \mathcal{W} = 0, \mathcal{R} = 1, \mathcal{U} = 0, \mathcal{O} = 1]$ (the robot is in the office and the owner does not have a coffee), the transition function of the problem defines

- $P([\mathcal{C} = 1, \mathcal{H} = 1, \mathcal{W} = 0, \mathcal{R} = 1, \mathcal{U} = 0, \mathcal{O} = 1] | \mathbf{x}, \mathcal{D}$ elC) = 0.8,
- $P([\mathcal{C} = 0, \mathcal{H} = 1, \mathcal{W} = 0, \mathcal{R} = 1, \mathcal{U} = 0, \mathcal{O} = 1] | \mathbf{x}, \mathcal{D}$ elC) = 0.2,
- 0.0 for other transition probabilities.

Finally, for the reward function, the robot gets a reward of 0.9 when the owner has a coffee (0 when it does not) and 0.1 when it is dry (and 0 when the robot is wet). The reward the robot obtains when the owner gets a coffee is larger than the reward obtained when the robot is dry so as to specify that the task of getting a coffee has a higher priority than the constraint of staying dry.

4.2.3. Decomposition and function-specific independence

Function-specific independencies refer to the property that some functions in the problem do not depend on all the variables in the problem or on the action executed by the agent. For instance, in the *Coffee Robot* problem, the value of the variable \mathcal{R}

at the next time step, meaning if it is raining or not, only depends on its own value at the current time step. Indeed, the fact that it is going to rain at the next time step is independent of variables such as “Does the robot have a coffee ?” (variable \mathcal{H}) or the last action executed by the robot.

The FMDP framework allows us to take advantage of such independencies in the representation of the transition and reward functions. Once these independencies specified, planning algorithms can then exploit them to improve their complexity. This notion is formalized by two different operators, namely PARENTS and SCOPE, defined, respectively, in the next section (section 4.2.3.1) for the transition function and in section 4.2.3.4 for the reward function.

4.2.3.1. Transition function decomposition

A transition function in a finite MDP is defined by the finite set of probabilities $p(\mathbf{x}_{t+1} \mid \mathbf{x}_t, a_t)$. Assuming that the state can be decomposed using multiple random variables (see section 4.2.1), it is possible to decompose the probability $p(\mathbf{x}_{t+1} \mid \mathbf{x}_t, a_t)$ as a product of probabilities, then to exploit the independencies between these random variables to decrease the size of the representation of the transition function.

For instance, in the *Coffee Robot* problem, the transition function for the action DelC is defined by the set of probabilities $P_{\text{DelC}}(\mathbf{x}_{t+1} \mid \mathbf{x}_t)$. By first decomposing the state \mathbf{x}_{t+1} and using the Bayes rule, we have

$$\begin{aligned} P_{\text{DelC}}(\mathbf{x}_{t+1} \mid \mathbf{x}_t) &= P_{\text{DelC}}(c_{t+1}, h_{t+1}, w_{t+1}, r_{t+1}, u_{t+1}, o_{t+1} \mid \mathbf{x}_t) \\ &= P_{\text{DelC}}(c_{t+1} \mid \mathbf{x}_t) * P_{\text{DelC}}(h_{t+1} \mid \mathbf{x}_t, c_{t+1}) * \dots \\ &\quad * P_{\text{DelC}}(o_{t+1} \mid \mathbf{x}_t, c_{t+1}, h_{t+1}, w_{t+1}, r_{t+1}, u_{t+1}), \end{aligned}$$

where \mathbf{X}_t is a random variable representing the state at time t . In the case of the *Coffee Robot* problem, we know that the value of a variable X_{t+1} only depends on the variables at time t , so that we have

$$\begin{aligned} P_{\text{DelC}}(\mathbf{x}_{t+1} \mid \mathbf{x}_t) &= P_{\text{DelC}}(c_{t+1} \mid \mathbf{x}_t) * P_{\text{DelC}}(h_{t+1} \mid \mathbf{x}_t, c_{t+1}) * \dots \\ &\quad * P_{\text{DelC}}(o_{t+1} \mid \mathbf{x}_t, c_{t+1}, h_{t+1}, w_{t+1}, r_{t+1}, u_{t+1}) \\ &= P_{\text{DelC}}(c_{t+1} \mid \mathbf{x}_t) * \dots * P_{\text{DelC}}(o_{t+1} \mid \mathbf{x}_t). \end{aligned}$$

Similarly, it is possible to decompose the state \mathbf{x}_t :

$$\begin{aligned} P_{\text{DelC}}(\mathbf{x}_{t+1} \mid \mathbf{x}_t) &= P_{\text{DelC}}(c_{t+1} \mid \mathbf{x}_t) * \dots * P_{\text{DelC}}(o_{t+1} \mid \mathbf{x}_t) \\ &= P_{\text{DelC}}(c_{t+1} \mid c_t, h_t, w_t, r_t, u_t, o_t) * \dots \\ &\quad * P_{\text{DelC}}(o_{t+1} \mid c_t, h_t, w_t, r_t, u_t, o_t). \end{aligned}$$

Then, given the structure of the *Coffee Robot* problem, we know, for instance, that the probability $P_{\text{DelC}}(\mathcal{R}_{t+1} = 1 \mid \mathbf{x}_t)$ that it will rain at the next time step only depends on r_t , that is, if it is raining at time t . Thus, we have $P_{\text{DelC}}(\mathcal{R}_{t+1} = 1 \mid \mathbf{x}_t) = P_{\text{DelC}}(\mathcal{R}_{t+1} = 1 \mid r_t)$. By exploiting function-specific independencies, it is possible to compactly represent the transition function for the action DelC by defining each probability $P_{\text{DelC}}(x_{t+1} \mid \mathbf{x}_t)$ only by the variables it depends on at time t in the problem (rather than all the variables composing the state \mathbf{x}_t):

$$\begin{aligned} P_{\text{DelC}}(\mathbf{x}_{t+1} \mid \mathbf{x}_t) &= P_{\text{DelC}}(c_{t+1} \mid c_t, h_t, w_t, r_t, u_t, o_t) * \dots \\ &\quad * P_{\text{DelC}}(o_{t+1} \mid c_t, h_t, w_t, r_t, u_t, o_t) \\ &= P_{\text{DelC}}(c_{t+1} \mid c_t, h_t, o_t) * P_{\text{DelC}}(h_{t+1} \mid h_t, o_t) \\ &\quad * P_{\text{DelC}}(w_{t+1} \mid w_t) * P_{\text{DelC}}(u_{t+1} \mid u_t) \\ &\quad * P_{\text{DelC}}(r_{t+1} \mid r_t) * P_{\text{DelC}}(o_{t+1} \mid o_t). \end{aligned}$$

So, for instance, rather than defining the probability distribution $P_{\text{DelC}}(\mathcal{R}_{t+1} \mid \mathbf{X}_t)$, we now use only the required variables, that is, the same variable at time t : $P_{\text{DelC}}(\mathcal{R}_{t+1} \mid \mathbf{X}_t) = P_{\text{DelC}}(\mathcal{R}_{t+1} \mid \mathcal{R}_t)$. The *Coffee Robot* problem is composed of six binary variables, meaning that it needs to define $2^6 * 2^6 = 4096$ probabilities $P_{\text{DelC}}(\mathbf{x}_{t+1} \mid \mathbf{x}_t)$ for the action DelC whereas the compact representation above only needs to define $2 * 2^3 + 2 * 2^2 + 2 * 2^1 + 2 * 2^1 + 2 * 2^1 + 2 * 2^1 = 40$ probabilities.

Consequently, function-specific independence related to the structure of the problem is explicitly defined and makes it possible to aggregate regularities in the transition function. Moreover, function-specific independence refers to an often intuitive representation of a problem by describing the consequences of actions on the values of the different variables. In FMDPs, function-specific independencies are formalized with dynamic Bayesian networks [BOU 95].

4.2.3.2. Dynamic Bayesian networks in FMDPs

Bayesian networks [PEA 88] are a representational framework to represent dependencies (or independencies) between random variables. These variables are the nodes of a directed graph. Direct probabilistic dependencies between two variables are represented by an edge between the two nodes representing these two variables. Dynamic Bayesian networks (DBNs) are Bayesian networks representing temporal stochastic processes. The nodes in a DBN represent variables for a particular time slice.

Assuming that the problem is stationary (the transition function T of the MDP does not depend on the time), it is possible to represent T with DBNs using only two

successive time steps (assuming the Markov property is satisfied). In such a case, DBNs are composed of two sets of nodes:

- 1) the set of nodes representing the variables of the problem at time t ,
- 2) the set of nodes representing the variables of the problem at time $t + 1$.

Edges indicate direct dependencies between variables at time t and variables at time $t + 1$ or between variables in the same time slice at time $t + 1$ (such edges are named *synchronous arcs*). Such DBNs are sometimes named *2 Time-slice Bayesian Networks*.

Independencies between random variables to define the transition function can then be represented using one DBN per action. Similarly, even if the action is rather a decision variables, actions that were executed in the past by the agent can also be considered as a random variable at time t . In such a case, only one DBN is necessary to define independencies between variables (and the action) in the transition function [BOU 96]. Finally, the DBN is quantified by a conditional probability distribution to define the probability of each value $x \in \text{DOM}(X)$ for each variable X given the value of the random variables X directly depends on (its parent variables), as illustrated in the next section with the *Coffee Robot* example.

4.2.3.3. Factored model of the transition function in an FMDP

Figure 4.1 represents the effect of the action $\mathcal{D}\text{elC}$ on a state. The DBN $\tau_{\mathcal{D}\text{elC}}$ (Figure 4.1(a)) clearly states that, for instance, for the action $\mathcal{D}\text{elC}$, the variable \mathcal{C} only depends on the values of the variables \mathcal{O} , \mathcal{H} and \mathcal{C} at the previous time step and is independent of the other state variables.

We can define $\text{PARENTS}_\tau(X'_i)$ the set of parents of the variable X'_i in DBN τ . This set can be partitioned in two subsets $\text{PARENTS}_\tau^t(X'_i)$ and $\text{PARENTS}_\tau^{t+1}(X'_i)$ representing, respectively, the set of parents at time t and the set of parents at time $t+1$. In the *Coffee Robot* example, we assume that there is no synchronous arcs, that is, $\text{PARENTS}_\tau^{t+1}(X'_i) = \emptyset$ and $\text{PARENTS}_\tau(X'_i) = \text{PARENTS}_\tau^t(X'_i)$. Thus, in Figure 4.1, we have $\text{PARENTS}_{\mathcal{D}\text{elC}}(\mathcal{C}') = \{\mathcal{O}, \mathcal{H}, \mathcal{C}\}$.

Corresponding DBN τ is quantified by a set of conditional probability distributions, noted $P_\tau(X'_i \mid \text{PARENTS}_\tau(X'_i))$ for a variable X'_i . Then, the probability $P_\tau(\mathbf{X}' \mid \mathbf{X})$ can be defined compactly as

$$P_\tau(\mathbf{x}' \mid \mathbf{x}) = \prod_i P_\tau(x'_i \mid \text{PARENTS}(x'_i)) \quad (4.1)$$

with x'_i the value of the variable X'_i in state \mathbf{x}' and $\text{PARENTS}(x'_i)$ the values of the variables in the set $\text{PARENTS}_\tau(X'_i)$.

Figure 4.1(b) gives the conditional probability distribution $P_{\mathcal{D}\text{elC}}(\mathcal{C}' \mid \mathcal{O}, \mathcal{H}, \mathcal{C})$ in the *Coffee Robot* problem in a tabular form. The columns \mathcal{O} , \mathcal{H} and \mathcal{C} represent

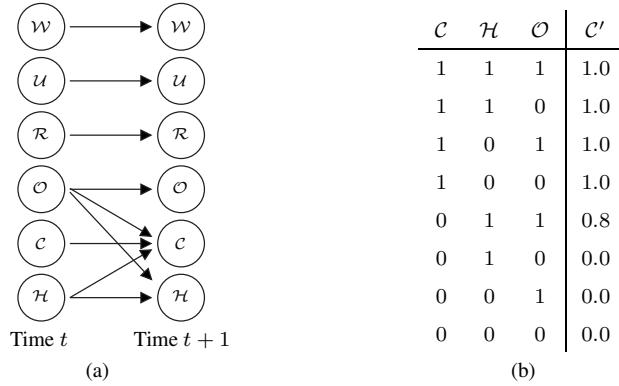


Figure 4.1. Partial representation of the transition function T for the Coffee Robot problem.

Figure (a) represents the dependencies between variables for the DelC action.

Figure (b) defines the conditional probability distribution $P_{\text{DelC}}(\mathcal{C}' | \mathcal{O}, \mathcal{H}, \mathcal{C})$ using a tabular representation

the values of these variables at time t . The column \mathcal{C}' represents the probability for variable \mathcal{C} to be true at time $t + 1$.

The multiplicative decomposition (equation (4.1)) and the specification of functional independencies in the model description of the transition function are the main contributions of FMDPs compared to MDPs. Both of these properties are exploited by the algorithms exploiting the structure of the problem specified in the FMDP.

4.2.3.4. Factored model of the reward function

A similar representation can be used to specify the reward function of the problem in FMDPs. Indeed, first, the reward function R can be decomposed additively and, second, the different terms of the decomposition do not necessarily depend on all the state variables of the problem.

For instance, in the *Coffee Robot* problem, the reward function, represented by a diamond in Figure 4.2, only depends on two variables \mathcal{C} and \mathcal{W} . It is independent of the action executed by the agent or of other variables in the problem.

The table in Figure 4.2(b) specifies that the best state for the robot is when its owner has a coffee and the robot is dry whereas the worst case is when its owner does not have a coffee and the robot is wet. We can notice that a preference is given to the state where the owner has a coffee and the robot is dry over the state where the owner does not have a coffee and the robot is dry.

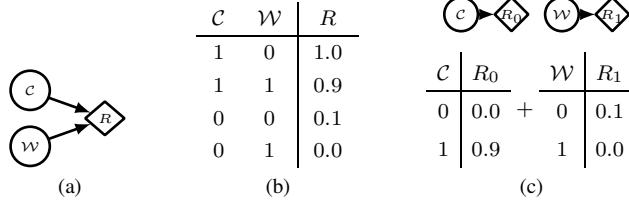


Figure 4.2. Representation of the reward function $R(s)$ in the Coffee Robot problem

[BOU 00] defines the reward function of the *Coffee Robot* problem by summing the two criteria of the problem: “the owner has a coffee” and “the robot is dry”. However, these two criteria are independent of each other. To be able to exploit this additive decomposition of the reward function, [GUE 03b] proposes to represent the reward function of an FMDP as a sum of different *localized reward functions*.

Given the *Coffee Robot* problem, we can define the reward function as being the sum of two localized reward functions depending, respectively, on the variables \mathcal{C} and \mathcal{W} , representing the criteria “the owner has a coffee” and “the robot is dry”.

[GUE 03b] formalizes such a structure by first defining the *scope* of a localized function f (noted $\text{SCOPE}(f)$). Similarly to PARENTS for DBNs, the scope of f is defined as follows.

DEFINITION 4.1 (scope). A function f has a scope $\text{SCOPE}(f) = \mathcal{C} \subseteq \mathbf{X}$ if $f : \text{DOM}(\mathcal{C}) \rightarrow \mathbb{R}$.

Let a function f such as $\text{SCOPE}(f) = \mathcal{C}$, we write $f(x)$ as a shorthand for $f(x[\mathcal{C}])$ where $x[\mathcal{C}]$ is the restriction of x to the variables in \mathcal{C} . Consequently, the SCOPE definition allows us to define the function-specific independence of f .

It is now possible to define a *localized reward function*. Let a set of localized reward functions R_1^a, \dots, R_r^a with scope $\text{SCOPE}(R_i^a)$ for each R_i^a be constrained to a subset $C_i^a \subseteq \{X_1, \dots, X_n\}$, then the reward function associated with action a is defined as

$$R^a(\mathbf{x}) = \sum_{i=1}^r R_i^a(\mathbf{x}[C_i^a]) \quad (4.2)$$

$$= \sum_{i=1}^r R_i^a(\mathbf{x}). \quad (4.3)$$

Regarding the *Coffee Robot* example, the problem can be defined by the two reward functions R_1 and R_2 given in Figure 4.2(c) and representing, respectively, the two criteria “the owner has a coffee” and “the robot is dry” with $\text{SCOPE}(R_1) = \{\mathcal{C}\}$ and $\text{SCOPE}(R_2) = \{\mathcal{W}\}$. We write $R_1(\mathbf{x})$ as a shorthand for $R_1(\mathbf{x}[\mathcal{C}])$ with $\mathbf{x}[\mathcal{C}]$ representing the value of \mathcal{C} in \mathbf{x} .

Whereas all algorithms in the FMDP framework exploit function-specific independencies of the reward function, they do not necessarily exploit its additive decomposition. Moreover, not all problems show such structure in their reward function.

4.2.4. Context-specific independence

For a given function and a given context, it is not necessarily required to test every variable on which the function depends to define the output of the function. Such property is named *context-specific independence*.

For the *Coffee Robot* problem, in the definition of the conditional probability distribution $P_{\mathcal{D}\text{el}\mathcal{C}}(\mathcal{C}' \mid \mathcal{O}, \mathcal{H}, \mathcal{C})$ (see Figure 4.1(b)), whereas $\text{PARENTS}_{\mathcal{D}\text{el}\mathcal{C}}(\mathcal{C}') = \{\mathcal{O}, \mathcal{H}, \mathcal{C}\}$, it is not necessary to test variables \mathcal{O} and \mathcal{H} if $\mathcal{C} = 1$ when $\mathcal{C}_t = 1$ to know the probability distribution of the variable \mathcal{C}' .

From [GUE 03a], a context is formalized as follows:

DEFINITION 4.2 (context). *Let a function $f : \mathbf{X} \rightarrow \mathbf{Y}$. A context $\mathbf{c} \in \text{DOM}(f)$ is an instantiation of a multivariate random variable $\mathbf{C} = (C_0, \dots, C_j)$ such that $\mathbf{C} \subseteq \mathbf{X}$. It is noted: $(C_0 = c_0) \wedge \dots \wedge (C_j = c_j)$ or $C_0 = c_0 \wedge \dots \wedge C_j = c_j$.*

Unlike function-specific independence, exploiting context-specific independence is directly related to the data structure used by the algorithms to solve the problem. Indeed, the operators PARENTS and SCOPE, representing function-specific independence, define the set of variables on which the function depends. As seen in section 4.2.3, such structures allow us to compactly represent some problems, even when the data structure used to define these functions is not structured, as it is the case for the tabular representation used in Figure 4.1.

For a given set of variables (specified with the PARENTS and SCOPE operators), context-specific independencies are used to represent a function more compactly. In this case, the main idea is to use structured representations to aggregate similar states, unlike tabular representations. Thus, [BOU 00] suggests different data structures to represent the different functions of a given FMDP, such as rules [POO 97], decision lists [RIV 87] or algebraic decision diagrams [BRY 86]. Consequently, because each algorithm proposed in the FMDP framework uses a different data structure and strongly relies on it, we have preferred focusing on the description of these data structures in the next section.

4.3. Planning with FMDPs

This section describes different planning methods to solve problems specified as FMDPs. Rather than describing the algorithms in detail, we describe the different representations and data structures they use as an outline of their main properties. However, all the required references are given for the reader interested in more in-depth descriptions.

4.3.1. Structured policy iteration and structured value iteration

Structured Value Iteration (SVI) and *Structured Policy Iteration* (SPI) [BOU 00] are adaptations to FMDPs of the *Policy Iteration* and *Value Iteration* algorithms. In addition to using function-specific independence, SVI and SPI exploit context-specific independence by using decision trees to represent the different functions of the problem.

4.3.1.1. Decision trees

Decision trees represent a function by partitioning its input space and associating the output value to each of these partitions. A decision tree is composed of:

- *internal nodes* (or decision nodes): they represent a test on a variable of the input space; they are parents of other nodes in the tree and define the partitions of the input space;
- *edges*: they connect a parent interior node to a child node and constrain the value of the variable tested at the parent node to one value to reach the child node;
- *external nodes* (or leaves): they represent the terminal nodes of the tree and define the value of the function for the partition defined by the parent (internal) nodes.

Note that, in a decision tree, one node has only one parent (except for the root which has no parent).

In SVI and SPI, decision trees are used to represent the different functions of the FMDP, such as reward functions, transition functions, policies and value functions. A function f represented by a decision tree is denoted by $\text{Tree}[f]$. Graphically, we represent decision trees with the following convention: for an internal node testing a Boolean variable X , the left and right edges correspond, respectively, to $X = 1$ and $X = 0$ (or, respectively, X being true and X being false).

4.3.1.2. Representation of the transition function

In the *Coffee Robot* problem, the tabular representation of the conditional probability distribution $P_{\text{DelC}}(\mathcal{C}' \mid \mathcal{O}, \mathcal{H}, \mathcal{C})$ (see Figure 4.3(a)) shows, depending on the context, different regularities that can be exploited to represent the function more compactly. For instance, as described above in section 4.2.4, in the context where $\mathcal{C} = 1$, the probability that \mathcal{C}' is true is equal to 1, whatever the value of the two other

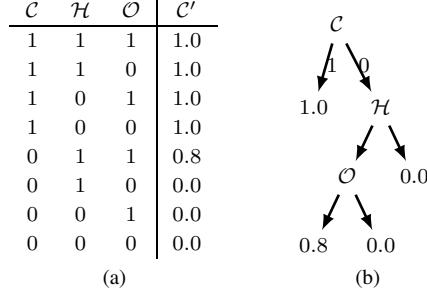


Figure 4.3. Representation of the conditional probability distribution $P_{\text{DelC}}(\mathcal{C}' \mid \mathcal{O}, \mathcal{H}, \mathcal{C})$ with (a) a tabular data structure and (b) a decision tree. The leaf 0.8 means that the probability for the variable \mathcal{C}' of being true at the next time step is $P_{\text{DelC}}(\mathcal{C}' = 1 \mid \mathcal{O} = 1, \mathcal{H} = 1, \mathcal{C} = 0) = 0.8$. In the decision tree, note that some regularities are aggregated such the probability distributions when $P_{\text{DelC}}(\mathcal{C}' = 1 \mid \mathcal{C} = 1) = 1.0$

variables $\mathcal{O}, \mathcal{H} \in \text{PARENTS}_{\text{DelC}}(\mathcal{C}')$. In the problem, this means that it is certain that an owner having a coffee will still have it at the next time step. Decision trees allow to represent such context-specific regularities more compactly than tabular representations.

A decision tree $\text{Tree}[P_{\tau}(X' \mid \text{PARENTS}_{\tau}(X'))]$ representing a conditional probability distribution $P_{\tau}(X' \mid \text{PARENTS}_{\tau}(X'))$ is made of:

- *internal nodes*: they represent a test on a variable $X_j \in \text{PARENTS}_{\tau}(X')$;
- *edges*: they represent a value $x_j \in \text{DOM}(X_j)$ of a variable X_j tested at the parent node and defining a partition represented by the child node connected to the edge;
- *external nodes*: they represent the probability distribution $P_{\tau_l}(X' \mid c_l)$, with c_l the context defined by the set of values of the variables $X_j \in \text{PARENTS}_{\tau_l}(X')$ tested in the parents node for a leaf l in the tree.

Reading such a tree is straightforward: the probability distribution of a variable X' for a given instantiation x is given by the unique leaf reached by selecting the edge at each internal node corresponding to the value of the tested variable in the instantiation x . Such a path defines the context C_l associated with the leaf l reached.

Figure 4.3(b) represents the conditional probability distribution $P_{\text{DelC}}(\mathcal{C}' \mid \mathcal{O}, \mathcal{H}, \mathcal{C})$ as a decision tree. The value at a leaf indicates the probability that the variable \mathcal{C}' will be true at the next time step. Because the decision tree representation exploits context-specific independence in this example, the representation of $P_{\text{DelC}}(\mathcal{C}' \mid \mathcal{O}, \mathcal{H}, \mathcal{C})$ is more compact compared to the tabular representation.

Whereas 8 lines are required (Figure 4.3(a)) for the tabular form, only 4 are required for the same function with a decision tree. Such factorization is one of the main principle used by SPI and SVI for planning.

4.3.1.3. Representation of the reward function

The representation of the reward function with decision trees is very similar to the representation of conditional probability distributions described above. Indeed, the semantics of the internal nodes and edges are the same. Only the values attached to the leaves of the tree are different: rather than probabilities, the leaves represent real numbers.

Figure 4.4 represents the reward function for the *Coffee Robot* problem and compares (a) the tabular representation $R(\mathbf{x})$ with (b) a decision tree representation $\text{Tree}[R(\mathbf{x})]$. Note that the number of leaves in the tree is equal to the number of lines in the table, meaning that there is no context-specific independence to exploit in the representation of this function.

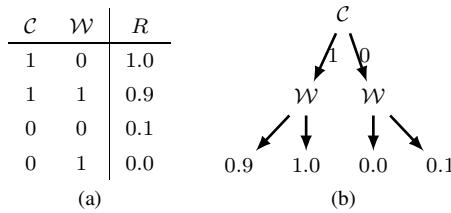


Figure 4.4. Definition of the reward function $R(\mathbf{x})$ with (a) a tabular representation and (b) a decision tree. The leaf 0.9 means $R(\mathcal{C} = 1, \mathcal{W} = 1) = 0.9$

Finally, SVI and SPI are not able to exploit the additive decomposition of the reward function as described in section 4.2.3.4.

4.3.1.4. Representation of a policy

Of course, a policy $\pi(\mathbf{x})$ can also be represented with a decision tree $\text{Tree}[\pi(\mathbf{x})]$. Figure 4.5 represents a stationary optimal policy $\text{Tree}[\pi^*(\mathbf{x})]$ in the *Coffee Robot* problem.

The state space of the problem *Coffee Robot* is composed of 6 binary variables. So, a tabular representation of a policy π requires $2^6 = 64$ entries. The tree $\text{Tree}[\pi^*]$ representing the optimal policy in the *Coffee Robot* problem requires only 8 leaves (15 nodes in total). Consequently, in this problem, the decision tree representation of the policy exploits context-specific independencies. This means, for instance, that when the robot is in the office with a coffee, it is not necessary to check the weather to define the best action to perform.

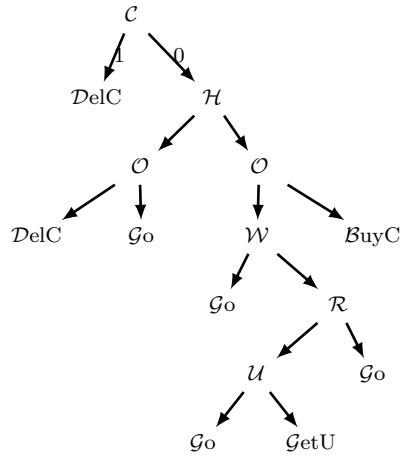


Figure 4.5. Representation of an optimal policy $\pi^*(x)$ with a decision tree $\text{Tree}[\pi^*(x)]$.
The leaf BuyC means $\pi(\mathcal{C} = 0, \mathcal{H} = 0, \mathcal{O} = 0) = \text{BuyC}$

In the worst case, note that only N tests are required to determine the action to execute for a problem with a state space made of N variables. This is not necessarily the case for all structured representations (see, e.g. section 4.3.3.4). Moreover, decision trees allow us to calculate the values of the minimum set of variables needed to define the next action to execute. Such property can be important when the policy is run in an environment where calculating the value of a variable has a cost (e.g. computation time).

4.3.1.5. Representation of the value function

Obviously, the value function V_π of a policy π can also be represented with a decision tree $\text{Tree}[V_\pi]$. The semantics of such tree is identical to a tree representing the reward function: internal nodes, edges and leaves represent, respectively, a test on a variable, a value of the tested variable at the parent internal node and the value of the function in the corresponding partition. Figure 4.6 represents the value function of the policy $\text{Tree}[\pi^*]$ represented in Figure 4.5.

$\text{Tree}[V_{\pi^*}(x)]$ contains only 18 leaves (35 nodes in total) whereas a tabular representation would have required 64 entries. Thus, in the *Coffee Robot* problem, a decision tree representation makes it possible to exploit context-specific independencies. For instance, the value $V_{\pi^*}(\mathcal{C} = 1, \mathcal{W} = 0)$ of the optimal policy π^* , that is, the owner has a coffee and the robot is dry, does not depend on the other variables in the problem. Consequently, the representation aggregates a set of states. Thus, when the value function is updated incrementally while solving the problem, it is required to calculate only the value at the leaf corresponding to the context rather than updating every state corresponding to this same context.

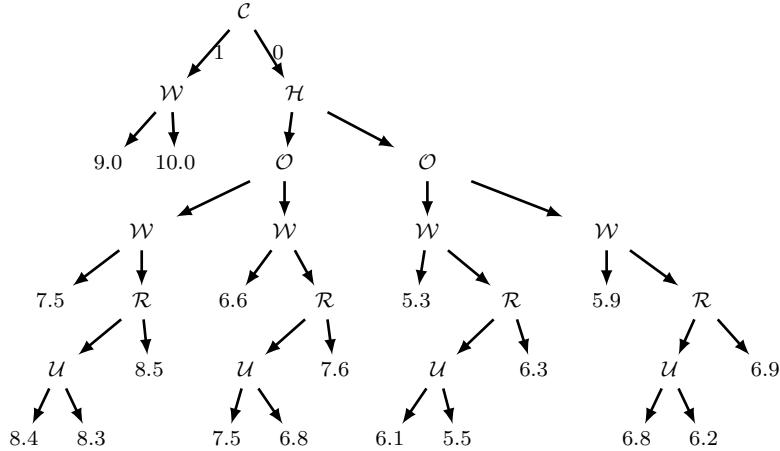


Figure 4.6. Representation of the value function $V_{\pi^*}(x)$ of the policy π^* as a decision tree $\text{Tree}[V_{\pi^*}(x)]$ for the problem Coffee Robot. The leaf 10.0 means $V_{\pi^*}(\mathcal{C} = 1, \mathcal{W} = 0) = 10.0$

However, a decision tree representation does not allow us to exploit certain regularities in the structure of the function. For instance, the sub-trees in $\text{Tree}[V_{\pi^*}]$ composed of the variables \mathcal{R} , \mathcal{W} , \mathcal{U} and \mathcal{O} share the same structure. Such structure can be exploited with an additive approximation of the value function as we will see in section 4.3.3.5.

Finally, in the worst case, that is, when the value function of the evaluated policy has a different value for each possible state, the size of the tree increases exponentially with the number of variables composing the state space, similarly to a tabular representation.

4.3.1.6. Algorithms

SVI and SPI are adaptations of, respectively, *Value Iteration* and *Policy Iteration* to decision tree representation. Consequently, rather than iterating on all the states of the problem to update the value function as *Value Iteration* and *Policy Iteration* do, SVI and SPI calculate the update only for each leaf of the decision tree, decreasing the computation when states are aggregated and represented with one leaf. We recommend reading [BOU 00] for an exhaustive description of both SVI and SPI.

4.3.2. SPUDD: stochastic planning using decision diagrams

In some problems, value functions have symmetries that are not exploited by decision trees, for instance, when the function is strictly identical in some disjoint context. SPUDD (for *Stochastic Planning Using Decision Diagrams*), proposed by

[HOE 99], uses *Algebraic Decision Diagrams* [BAH 93] (noted ADD) to represent the different functions of an FMDP. Similarly to SVI and SPI, SPUDD exploits function-specific and context-specific independencies.

Using ADDs rather than decision trees has two additional advantages. First of all, as mentioned before, ADDs can aggregate together identical substructures which have disjoint contexts.

Second, the variables used in an ADD are ordered. Whereas, finding an optimal order of tests on the variables of the problem to represent the most compact representation is a difficult problem, [HOE 00] describes different heuristics that are good enough to improve significantly the size of the representation. Moreover, such an ordering is exploited to manipulate ADDs more efficiently compared to decision trees where no ordering is assumed.

Whereas SPUDD, similarly to SVI, is an adaptation of *Value Iteration* to work with ADDs, both of the advantages described above allow SPUDD to perform significantly better than SPI or SVI on most problems proposed in the FMDP literature.

4.3.2.1. Representing the functions of an FMDP with ADDs

ADDs are a generalization of binary decision diagrams [BRY 86]. Binary decision diagrams are a compact representation of $\mathcal{B}^n \rightarrow \mathcal{B}$ functions of n binary variables to a binary variable. ADDs generalize binary decision diagrams to represent $\mathcal{B}^n \rightarrow \mathbb{R}$ functions of n binary variables to a real value in \mathbb{R} . An ADD is defined by:

- *internal nodes* (or decision nodes): they represent a test on a variable from the input space; they are the parent of two edges corresponding respectively to the values `true` and `false`;
- *edges*: they connect each parent internal node to a child node depending on its associated value `true` or `false`;
- *external nodes* (or leaves): they represent terminal nodes in the diagram and are associated with the value of the function in the subspace defined by the set of tests of the parent nodes to reach the leaf.

Unlike decision trees, a node (internal or external) in an ADD can have multiple parents. A function f represented with an ADD is denoted by $\text{ADD}[f]$. We use the following graphical convention to represent ADDs: the edges of an internal node testing a variable X are drawn with a plain or dashed line, corresponding, respectively, to X being `true` or `false` (or $X = 1$ and $X = 0$).

Compared to decision trees, ADDs have several interesting properties. First, because an order is given, each distinct function has only one representation. Moreover, the size of the representation can be compacted because identical

sub-graphs can be factored in the description. Finally, optimized algorithms have been proposed for most of the basic operators, such as the multiplication, the addition or the maximization of two ADDs.

Figure 4.7 shows an example of the same function f represented with a decision tree and an ADD. The figure illustrates that decision trees, unlike ADDs, are not adapted to represent disjunctive functions. Thus, the tree representation $\text{Tree}[f]$ is composed of 5 different leaves (and 4 internal nodes) whereas the ADD representation $\text{ADD}[f]$ contains only 2 leaves (and 3 internal nodes). Thus, an algorithm iterating on the leaves of the representation may have its complexity decreased when using ADDs rather than decision trees.

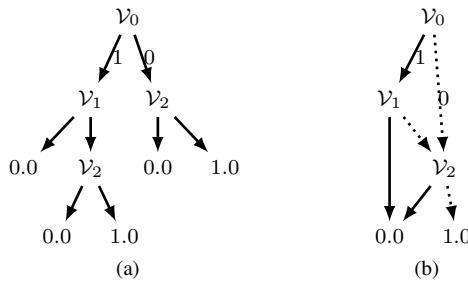


Figure 4.7. Comparison of the representation of a function f (a) as a decision tree $\text{Tree}[f]$ and (b) as an algebraic decision diagram $\text{ADD}[f]$

However, using ADDs adds two constraints on the FMDP to solve. First, all the variables in the FMDP have to be binary, ADDs representing only functions $\mathcal{B}^n \rightarrow \mathbb{R}$. For FMDPs with non-binary variables, these variables are decomposed and replaced by their corresponding additional (binary) variables. Second, as mentioned above, the algorithms manipulating ADDs assume that, in the ADDs, the tests on the variables (the internal nodes) are sorted. When both constraints are satisfied, it is possible to represent all the functions of an FMDP with ADDs.

4.3.2.2. Algorithm

Similarly to SVI, SPUDD is based on *Value Iteration* with the operators implemented to manipulate ADDs, assuming that all the variables are binary and that they are ordered. The work on SPUDD has led to APRICODD [STA 01] which contains additional improvements. First of all, the user can parameterize the algorithm to approximate the value function by limiting the maximum size of the ADD used to represent the value function. Moreover, APRICODD implements different methods for automatic variable ordering to avoid the user to have to specify it manually.

The last version of APRICODD is available on the Internet.³ Note that APRICODD can be considered as a ready-to-use solution method to solve large problems that can be modeled as FMDPs.

4.3.3. Approximate linear programming in FMDPs

An alternative to dynamic programming to solve an MDP is linear programming (see section 1.6.2.1). Using linear programming to solve FMDPs is the result of a work started by [KOL 99, KOL 00] and then continued with Guestrin [GUE 01, GUE 03a, GUE 03b].

The optimal value function of an MDP can be calculated by formulating the MDP as a linear program [MAN 60]:

$$\begin{aligned} \text{For the variables: } & V(s), \forall s \in S; \\ \text{Minimize: } & \sum_s \alpha(s)V(s); \\ \text{Under constraints: } & V(s) \geq R(s, a) + \gamma \sum_{s'} P(s' | s, a)V(s') \\ & \forall s \in S, \forall a \in A; \end{aligned} \tag{LP 1}$$

where $\alpha(s) > 0$ is the state relevance weight for the state s .

Unfortunately, solving such a linear program is not possible for large MDPs because of the complexity of the objective function, of the number of variables to solve and of the number of constraints. These problems are solved by, first, using a linear approximation of the value function and, second, exploiting function-specific independence and additive decomposition of the reward function.

More precisely, using a linear approximation of the value function (i.e. a linear combination of basis functions [SCH 85]) decreases the complexity of the objective function to optimize and the number of variables to determine. Function-specific independence and additive decomposition of the reward function are exploited by an algorithm decomposing the constraints of the original linear program into a set of constraints with a complexity depending on the structure of the problem rather than on its size.

Both of these ideas are exploited by two different algorithms proposed in [GUE 03b]. The first one is based on the *Policy Iteration* algorithm using linear programming to evaluate the current policy. The second one constructs a linear program similar to (LP 1) to directly evaluate the optimal value function of the FMDP to solve. The next section presents the representation used by both algorithms.

3. <http://www.cs.toronto.edu/~jhoey/spudd>.

4.3.3.1. Representations

Two different representations are used by the algorithms proposed by [GUE 03b]. The first representation is the tabular representation (similar to the tabular representation used in Figure 4.1, section 4.2.3.3). Algorithms using such representation exploit function-specific independencies, linear approximation of the value function and additive decomposition of the reward function (and not context-specific independencies).

The second representation is a structured representation based on rules [ZHA 99], allowing us to exploit context-specific independencies in a function. Whereas [GUE 03b] shows that, for some problems, tabular representations are faster, we have chosen to describe the rules representation mainly because the complexity of the worst case using these representations is better than the worst case of tabular representations [STA 01, GUE 03a].

[GUE 03b] prefers using rules rather than another structured representation because rules may not be exclusive, unlike decision trees or ADDs. We distinguish two types of rules: *probability rules* and *value rules*. The former are used to represent the transition function, the latter to represent value and reward functions. We describe how these rules are used to represent the functions of an FMDP in the following sections. A function f is denoted by Rule $[f]$ when represented with a set of rules.

4.3.3.2. Representation of the transition function

Probability rules describe the transition function in an FMDP. More precisely, they are used to define the conditional probability distributions quantifying the DBNs. A rule corresponds to one context defining the same probability for this context.

We first start by defining the consistency between two contexts.

DEFINITION 4.3 (consistency between two contexts). *Let $C \subseteq \{\mathbf{X}, \mathbf{X}'\}$, $c \in \text{DOM}(C)$, $B \subseteq \{\mathbf{X}, \mathbf{X}'\}$ and $b \in \text{DOM}(B)$. Two contexts b and c are consistent if they have the same assignment for the variables in the intersection $C \cap B$.*

Consequently, identical probabilities with consistent contexts are represented with probability rules.

DEFINITION 4.4 (probability rule). *A probability rule $\eta = |c : p|$ is a function $\eta : \{\mathbf{X}, \mathbf{X}'\} \rightarrow [0, 1]$ with the context $c \in \text{DOM}(C)$, $C \subseteq \{\mathbf{X}, \mathbf{X}'\}$ and $p \in [0, 1]$, and such that $\eta(\mathbf{x}, \mathbf{x}') = p$ if the instantiations \mathbf{x} and \mathbf{x}' are consistent with c , or otherwise equal to 1.*

Two rules are consistent if their context is consistent.

A set of probability rules completely defines a conditional probability distribution.

DEFINITION 4.5 (set of probability rules). A set of rules P_a of a conditional probability distribution is a function $P_a : (\{X'_i\} \cup \mathbf{X}) \rightarrow [0, 1]$ composed of the probability rules $\{\eta_1, \dots, \eta_m\}$ with their mutually exclusive and exhaustive contexts. We define $P_a(x'_i \mid \mathbf{x}) = \eta_j(\mathbf{x}, \mathbf{x}')$ with η_j the only rule in P_a with the context c_j consistent with (x'_i, \mathbf{x}) . Moreover, we necessarily have $\forall \mathbf{x} \in \mathbf{X} : \sum_{x'_i} P_a(x'_i \mid \mathbf{x}) = 1$.

Note that $\text{PARENTS}_a(X'_i)$ can be defined as the union of the variables appearing in the contexts of the rules defining the distribution.

Similarly to decision trees, the sets of rules make it possible to exploit context-specific independencies. Moreover, decision trees define a complete partition of a space. Thus, it is straightforward to define a set of mutually exclusive and exhaustive rules from a given decision tree, as shown in Figure 4.8 for the conditional probability distribution $P_{\text{DelC}}(\mathcal{C}' \mid \text{PARENTS}_{\text{DelC}}(\mathcal{C}'))$.

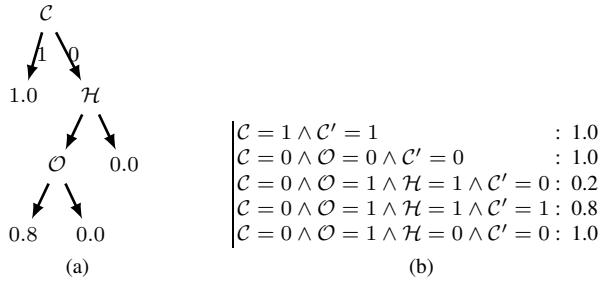


Figure 4.8. Representation of the conditional probability distribution $P_{\text{DelC}}(\mathcal{C}' \mid \text{PARENTS}_{\text{DelC}}(\mathcal{C}'))$ as a decision tree and a set of rules.

The rule $|\mathcal{C} = 0 \wedge O = 1 \wedge H = 1 \wedge C' = 1 : 0.8|$ defines $P_{\text{DelC}}(\mathcal{C}' = 1 \mid \mathcal{C} = 0, O = 1, H = 1) = 0.8$

The probability $P_{\text{DelC}}(\mathcal{C}' = 1 \mid \mathcal{C} = 0, O = 1, H = 1) = 0.8$ is represented by the corresponding rule $|\mathcal{C} = 0 \wedge O = 1 \wedge H = 1 \wedge C' = 1 : 0.8|$. We can notice that the context of a rule is split into two parts. The first part is the set of tests on the variables X at time t , corresponding to the path in the tree to reach the leaf 0.8. The second part is the value of the variable X'_i at time $t + 1$. Such representation is advantageous to solve problems with synchronous arcs. A conditional probability distribution f represented with a set of probability rules is denoted by $\text{Rule}_p[f]$.

4.3.3.3. Representation of the reward function

We define value rules to represent the reward function of an FMDP.

DEFINITION 4.6 (value rule). A value rule $\rho = |\mathbf{c} : v|$ is a function $\rho : \mathbf{X} \rightarrow \mathbb{R}$ such that $\rho(\mathbf{x}) = v$ when \mathbf{x} is consistent with the context \mathbf{c} and otherwise 0.

Note that the scope of a value rule may be defined as $\text{SCOPE}(\rho) = \mathbf{C}$ with \mathbf{C} the set of instantiated variables in the context c of the rule $\rho = |c : v|$.

It is now possible to define a function as a set of value rules.

DEFINITION 4.7 (set of a value rule). *A set of value rules representing a function $f : \mathbf{X} \rightarrow \mathbb{R}$ is composed of the set of value rules $\{\rho_1, \dots, \rho_n\}$ such that $f(\mathbf{x}) = \sum_{i=1}^n \rho_i(\mathbf{x})$ with $\forall i : \text{SCOPE}(\rho_i) \subseteq \mathbf{X}$.*

A function f represented with a set of value rules is denoted by $\text{Rule}_v [f]$.

Moreover, [GUE 03b] assumes that a reward function $R(\mathbf{x}, a)$ can be specified as the sum of reward functions with a limited scope:

$$R(\mathbf{x}, a) = \sum_j r_j^a(\mathbf{x}). \quad (4.4)$$

As shown in Figure 4.9, such representation allows us to easily define functions exploiting at the same time context-specific independence and additive decomposition.

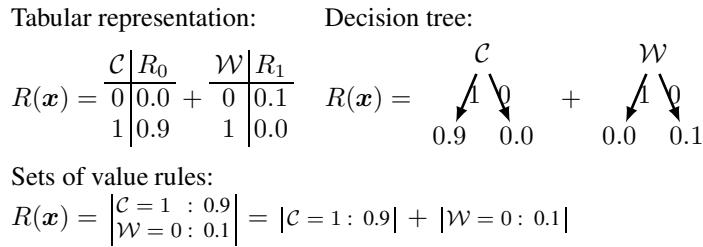


Figure 4.9. Representation of the reward function R in the Coffee Robot problem.
The reward is decomposed as a sum of reward functions with a scope limited to only one variable for each function

As described in section 4.2.3.4, the reward function in the *Coffee Robot* problem can be decomposed as a sum of two reward functions, each with a scope limited to only one variable of the problem. Different representations can be used to define these functions, in particular tables, decision trees or sets of rules. [GUE 03b] uses sets of value rules.

4.3.3.4. Policy representation

To compactly represent a policy π , [GUE 03b] uses a data structure first proposed by [KOL 00]. Rather than using $\text{Tree}[\pi]$ or $\text{ADD}[\pi]$, a default action in the FMDP is defined a priori and a policy is represented by an ordered decision list.

Every element in the list is defined by a triple containing: a context defining whether the action can be executed for a given state s , the action to execute if this decision has been taken and a bonus corresponding to the additional expected long term reward compared to the expected long-term reward if the default action were taken. The last element of a policy is always the default action with an empty context (the decision that can be taken at any time) and a bonus of 0. A policy π represented as a decision list is denoted by $\text{List}[\pi]$. Table 4.1 shows the optimal policy for the *Coffee Robot* problem where the default action is \mathcal{G}_0 .

	Context	Action	Bonus
0	$\mathcal{C} = 0 \wedge \mathcal{H} = 1 \wedge \mathcal{W} = 0 \wedge \mathcal{R} = 1 \wedge \mathcal{U} = 0 \wedge \mathcal{O} = 1$	$\mathcal{D}\mathcal{e}\mathcal{l}\mathcal{C}$	2.28
1	$\mathcal{C} = 0 \wedge \mathcal{H} = 0 \wedge \mathcal{W} = 0 \wedge \mathcal{R} = 1 \wedge \mathcal{U} = 0 \wedge \mathcal{O} = 0$	$\mathcal{B}\mathcal{u}\mathcal{y}\mathcal{C}$	1.87
2	$\mathcal{C} = 0 \wedge \mathcal{H} = 1 \wedge \mathcal{W} = 0 \wedge \mathcal{R} = 1 \wedge \mathcal{U} = 1 \wedge \mathcal{O} = 1$	$\mathcal{D}\mathcal{e}\mathcal{l}\mathcal{C}$	1.60
3	$\mathcal{C} = 0 \wedge \mathcal{H} = 1 \wedge \mathcal{W} = 1 \wedge \mathcal{O} = 1$	$\mathcal{D}\mathcal{e}\mathcal{l}\mathcal{C}$	1.45
4	$\mathcal{C} = 0 \wedge \mathcal{H} = 1 \wedge \mathcal{W} = 0 \wedge \mathcal{R} = 0 \wedge \mathcal{O} = 1$	$\mathcal{D}\mathcal{e}\mathcal{l}\mathcal{C}$	1.44
5	$\mathcal{C} = 0 \wedge \mathcal{H} = 0 \wedge \mathcal{W} = 0 \wedge \mathcal{R} = 1 \wedge \mathcal{U} = 1 \wedge \mathcal{O} = 0$	$\mathcal{B}\mathcal{u}\mathcal{y}\mathcal{C}$	1.27
6	$\mathcal{C} = 0 \wedge \mathcal{H} = 0 \wedge \mathcal{W} = 1 \wedge \mathcal{O} = 0$	$\mathcal{B}\mathcal{u}\mathcal{y}\mathcal{C}$	1.18
7	$\mathcal{C} = 0 \wedge \mathcal{H} = 0 \wedge \mathcal{W} = 0 \wedge \mathcal{R} = 0 \wedge \mathcal{O} = 0$	$\mathcal{B}\mathcal{u}\mathcal{y}\mathcal{C}$	1.18
8	$\mathcal{C} = 1 \wedge \mathcal{W} = 0 \wedge \mathcal{R} = 1 \wedge \mathcal{U} = 0$	$\mathcal{D}\mathcal{e}\mathcal{l}\mathcal{C}$	0.84
9	$\mathcal{C} = 0 \wedge \mathcal{H} = 0 \wedge \mathcal{W} = 0 \wedge \mathcal{R} = 1 \wedge \mathcal{U} = 0 \wedge \mathcal{O} = 1$	$\mathcal{G}\mathcal{e}\mathcal{t}\mathcal{U}$	0.18
10	$\mathcal{C} = 1 \wedge \mathcal{W} = 0 \wedge \mathcal{R} = 1 \wedge \mathcal{U} = 1$	$\mathcal{D}\mathcal{e}\mathcal{l}\mathcal{C}$	0.09
11	\emptyset	\mathcal{G}_0	0.00

Table 4.1. Representation of policy $\pi(s)$ as a decision list $\text{List}[\pi]$ (with the default action \mathcal{G}_0)

Note that, unlike decision trees or ADDS, the number of tests required to determine the action to execute can be higher than the number of variables in the problem.

4.3.3.5. Representation of the value function

We have seen that an MDP can be specified as the following linear program (section 1.6.2.1, (LP 1)):

$$\begin{aligned}
 & \text{For the variables: } V(s), \forall s \in S; \\
 & \text{Minimize: } \sum_s \alpha(s)V(s); \\
 & \text{Under constraints: } V(s) \geq R(s, a) + \gamma \sum_{s'} P(s' | s, a)V(s') \\
 & \quad \forall s \in S, \forall a \in A.
 \end{aligned} \tag{LP 2}$$

However, as described in section 4.3.3, because of the complexity in the number of variables to determine, the number of terms in the sum of the objective function

and the number of constraints, it is not possible to solve this linear program for large problems.

One solution, to decrease the number of terms in the sum of the objective function and the number of variables to solve, is to approximate the value function with a *linear combination* proposed by [BEL 63] (see Chapter 3). The space of approximate value functions $\tilde{V} \in \mathcal{H} \subseteq \mathbb{R}^n$ is defined by a set of *basis functions* with a scope limited to a small number of variables.

DEFINITION 4.8 (linear value function). A *linear value function* \tilde{V} with a set $H = \{h_0, \dots, h_k\}$ of basis functions is a function such that $\tilde{V}(s) = \sum_{j=1}^k w_j h_j(s)$ with $w \in \mathbb{R}^k$.

Such an approximation can directly be used to redefine the linear program by simply replacing the value function by its approximation in the objective function of the linear program [SCH 85]:

For the variables: w_1, \dots, w_k ;

$$\begin{aligned} \text{Minimize: } & \sum_{s=1}^k \alpha(s) \sum_{i=1}^k w_i h_i(s); \\ \text{Under constraints: } & \sum_{i=1}^k w_i h_i(s) \geq R(s, a) + \gamma \sum_{s'} P(s' | s, a) \sum_{i=1}^k w_i h_i(s') \\ & \forall s \in S, \forall a \in A. \end{aligned} \tag{LP 3}$$

Consequently, rather than determining the value function in the complete space of value functions, the search is reduced to the space corresponding to the set of weights w_i used in the linear approximation. Moreover, limiting the scope of the basis functions allows us to exploit function-specific independence to reduce the number of constraints.

However, whereas the number of variables to determine is not the number of possible states in the problem anymore but the number of weights w_i in the approximation, the number of terms in the sum and the number of constraints are still equal to the number of states in the problem.

For such linear programs, a solution exists only if a constant basis function is included in the set of basis functions [SCH 85]. [GUE 03b] assumes that such a function h_0 , such that $h_0(s) = 1, \forall s \in S$, is systematically included in the set of basis functions. Additionally, unlike (LP 1), the state relevance weights $\alpha(s)$ have an important effect on the quality of the approximation [FAR 01] and, thus, on the quality of the policies calculated from the value function.

Such an approximation in the value function allows us to exploit at the same time function-specific independencies and to exploit additional regularities in the structure of the value function as shown for the *Coffee Robot* problem in Figure 4.10. The additive decomposition of the approximated value function allows us to exploit regularities that neither decision trees nor ADDS are able to exploit, such as the similarities in the structure of internal nodes.

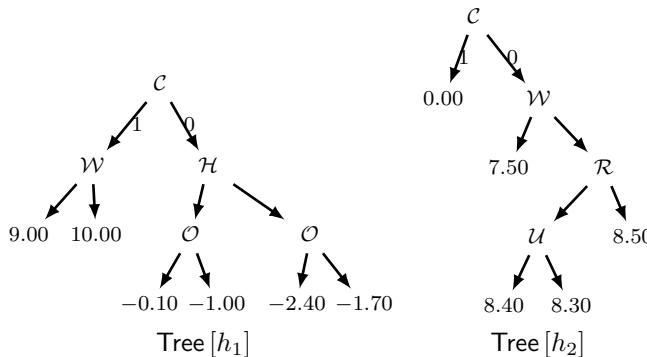


Figure 4.10. Example of a linear combination of a value function in the *Coffee Robot* problem as two decision trees representing two basis functions (with limited scopes) corresponding to the optimal policy $\pi^*(s)$ (Table 4.1). The optimal approximated value function is $\tilde{V}^*(s) = 0.63 \cdot \text{Tree}[h_0] + 0.94 \cdot \text{Tree}[h_1] + 0.96 \cdot \text{Tree}[h_2]$. The tree $\text{Tree}[h_0]$ is not shown since it defines a constant basis function and contains only one leaf equal to 1

The definition of the value function $\text{Tree}[V]$ (see Figure 4.6) is decomposed into two basis functions $\text{Tree}[h_1]$ and $\text{Tree}[h_2]$ and allows an approximation of $\text{Tree}[V_{\pi^*}]$ with an error less than 1.0.⁴ Additive decomposition of the value function is exploited because rather than containing 18 leaves, this representation requires only 11 leaves for both trees. So, the approximation of the value function in the *Coffee Robot* problem is composed of three basis functions (including the constant basis function). Thus, three weights, w_0 , w_1 and w_2 , must be determined with (LP 2).

Finally, when the reward function is compactly represented using an additive decomposition, it seems natural to expect that the value function also shows this kind properties. However, this is not necessarily the case. Indeed, a problem with a reward function with no additive decomposition may have an optimal value function well approximated with a linear approximation. On the other hand, a compact representation of the reward function or the transition function does not imply a compact representation of the value function [KOL 99, MUN 00, LIB 02].

4. The basis functions $\text{Tree}[h_1]$ and $\text{Tree}[h_2]$ have been defined manually, knowing $\text{Tree}[V_{\pi^*}]$ in the *Coffee Robot* problem.

4.3.3.6. Algorithms

Consequently, the algorithms proposed by [GUE 03b], for an FMDP, generate a linear program to calculate the value function of the problem. Additional algorithms are also described to calculate a policy (as a decision list). However, such a representation can be very expensive, even intractable in some cases. So, the authors suggest to directly estimate the approximated optimal value function of the problem. Then, approximated action value functions are calculated for each action (using the FMDP) to compare actions between each others and to estimate the best action to execute for a given state. Thus, an explicit representation of the policy is avoided. We suggest to refer to [GUE 03b] and [GUE 03a] for a complete description of these algorithms.

4.4. Perspectives and conclusion

Solving large FMDPs is still an active field of research and different extensions have been proposed in the last ten years. One extension, studied in [POU 05], is the extension to partially observable problems. Other extensions have been proposed to avoid having to specify the full structure or the values of the FMDP.

In this context, the algorithms presented in section 2.6.2 have been adapted to FMDPs, namely DBN-E³ [KEA 99], *factored* R-MAX [STR 07] and *factored* I.E. [STR 07]. These algorithms assume that function-specific independencies are known but not quantified. These algorithms propose exploring strategies to reach a policy near an optimal policy of the FMDP in a finite time.

A second approach does not assume that function-specific independencies are known beforehand and learn the structure of the problem from trials and errors of an agent in the problem [DEG 06]. However, despite interesting experimental results, no proof has been proposed yet. Research in this context is still being active [STR 07].

Another field of research in FMDPs is that of hierarchical approaches. A hierarchy of sub-problems is defined directly from a given FMDP. In this context, [JON 06] propose an algorithm named VISA with similar or better performance than SPUDD on different problems.

Finally, [SZI 08] has proposed to use dynamic programming rather than linear programming to solve FMDPs with a value function approximated by a linear combination of basis functions. Though their approach does not have necessarily better performance on all problems, their algorithms are notably simpler than the ones proposed by [GUE 03b].

4.5. Bibliography

- [BAH 93] BAHAR R., FROHM E., GAONA C., HACHTEL G., MACII E., PARDO A. and SOMENZI F., “Algebraic decision diagrams and their applications”, *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, Santa Clara, CA, pp. 188–191, 1993.
- [BEL 63] BELLMAN R., KALABA R. and KOTKIN B., “Polynomial approximation – a new computational technique in dynamic programming”, *Math. Comp.*, vol. 17, no. 8, pp. 155–161, 1963.
- [BOU 95] BOUTILIER C., DEARDEN R. and GOLDSZMIDT M., “Exploiting structure in policy construction”, *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI'95)*, Montreal, Canada, pp. 1104–1111, 1995.
- [BOU 96] BOUTILIER C. and GOLDSZMIDT M., “The frame problem and bayesian network action representations”, *Proceedings of the 11th Biennial Canadian Conference on Artificial Intelligence (AI '96)*, Toronto, Canada, pp. 69–83, 1996.
- [BOU 99] BOUTILIER C., DEAN T. and HANKS S., “Decision-theoretic planning: structural assumptions and computational leverage”, *Journal of Artificial Intelligence Research*, vol. 11, pp. 1–94, 1999.
- [BOU 00] BOUTILIER C., DEARDEN R. and GOLDSZMIDT M., “Stochastic dynamic programming with factored representations”, *Artificial Intelligence*, vol. 121, no. 1, pp. 49–107, 2000.
- [BRY 86] BRYANT R. E., “Graph-based algorithms for Boolean function manipulation”, *IEEE Transactions on Computers*, vol. C-35, no. 8, pp. 677–691, 1986.
- [DEG 06] DEGRIS T., SIGAUD O. and WUILLEMIN P.-H., “Learning the structure of factored Markov decision processes in reinforcement learning problems”, *Proceedings of the 23rd International Conference on Machine Learning (ICML'06)*, Pittsburgh, PA, pp. 257–264, 2006.
- [FAR 01] DE FARIAS D. and VAN ROY B., “The linear programming approach to approximate dynamic programming”, *Operations Research*, vol. 51, no. 6, pp. 850–856, 2001.
- [GUE 01] GUESTRIN C., KOLLER D. and PARR R., “Max-norm projections for factored MDPs”, *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI'01)*, Seattle, WA, pp. 673–680, 2001.
- [GUE 03a] GUESTRIN C., Planning under uncertainty in complex structured environments, PhD thesis, Computer Science Department, Stanford University, USA, 2003.
- [GUE 03b] GUESTRIN C., KOLLER D., PARR R. and VENKATARAMAN S., “Efficient solution algorithms for factored MDPs”, *Journal of Artificial Intelligence Research*, vol. 19, pp. 399–468, 2003.
- [HOE 99] HOEY J., ST-AUBIN R., HU A. and BOUTILIER C., “SPUDD: stochastic planning using decision diagrams”, *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI'99)*, Morgan Kaufmann, San Mateo, CA, pp. 279–288, 1999.

- [HOE 00] HOEY J., ST-AUBIN R., HU A. and BOUTILIER C., Optimal and approximate stochastic planning using decision diagrams, Report no. TR-00-05, University of British Columbia, 2000.
- [JON 06] JONSSON A. and BARTO A., “Causal graph based decomposition of factored MDPs”, *Journal of Machine Learning Research*, vol. 7, pp. 2259–2301, 2006.
- [KEA 99] KEARNS M. and KOLLER D., “Efficient reinforcement learning in factored MDPs”, *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, Stockholm, Sweden, pp. 740–747, 1999.
- [KOL 99] KOLLER D. and PARR R., “Computing factored value functions for policies in structured MDPs”, *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, Stockholm, Sweden, pp. 1332–1339, 1999.
- [KOL 00] KOLLER D. and PARR R., “Policy iteration for factored MDPs”, *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI'00)*, Morgan Kaufman, San Francisco, CA, pp. 326–334, 2000.
- [LIB 02] LIBERATORE P., “The size of MDP factored policies”, *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI'02)*, Edmonton, Alberta, Canada, pp. 267–272, 2002.
- [MAN 60] MANNE A. S., “Linear programming and sequential decisions”, *Management Science*, vol. 6, no. 3, pp. 259–267, 1960.
- [MUN 00] MUNDHENK M., GOLDSMITH J., LUSENA C. and ALLENDER E., “Complexity of finite-horizon Markov decision process problems”, *Journal of the ACM (JACM)*, vol. 47, no. 4, pp. 681–720, 2000.
- [PEA 88] PEARL J., *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann, San Mateo, CA, 1988.
- [POO 97] POOLE D., “The independent choice logic for modelling multiple agents under uncertainty”, *Artificial Intelligence*, vol. 94, no. 1-2, pp. 7–56, 1997.
- [POU 05] POUPART P., Exploiting structure to efficiently solve large scale partially observable Markov decision processes, PhD thesis, University of Toronto, 2005.
- [RIV 87] RIVEST R. L., “Learning decision lists”, *Machine Learning*, vol. 2, no. 3, pp. 229–246, 1987.
- [SCH 85] SCHWEITZER P. and SEIDMANN A., “Generalized polynomial approximations in Markovian decision processes”, *Journal of Mathematical Analysis and Applications*, vol. 110, pp. 568–582, 1985.
- [STA 01] ST-AUBIN R., HOEY J. and BOUTILIER C., “APRICODD: approximate policy construction using decision diagrams”, *Advances in Neural Information Processing Systems 13 (NIPS'00)*, pp. 1089–1095, 2001.
- [STR 07] STREHL A., DIUK C. and LITTMAN M. L., “Efficient structure learning in factored-state MDPs”, *Proceedings of the 22nd National Conference on Artificial Intelligence (AAAI'07)*, Vancouver, British Columbia, Canada, pp. 645–650, 2007.

[SZI 08] SZITA I. and LÖRINCZ A., “Factored value iteration converges”, *Acta Cybernetica*, vol. 18, no. 4, pp. 615–635, 2008.

[ZHA 99] ZHANG T. and POOLE D., “On the role of context-specific independence in probabilistic reasoning”, *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, Stockholm, Sweden, pp. 1288–1293, 1999.

Chapter 5

Policy-Gradient Algorithms

Most MDP solving methods evaluate a value function which makes it possible to identify optimal actions in each state. A first difficulty for these approaches is to handle large size problems. As shown in Chapter 3, a solution is to use methods approximating the value function. But the monotonous policy improvement is not guaranteed anymore for algorithms like value iteration or policy iteration. A second difficulty is to handle partial observations, a setting where the Markov property is not necessarily verified.

A very different approach is to perform a direct policy search, the policy taking the form of a parameterized controller. The choice of this controller makes it possible to adapt to the type of problem being considered and to make a compromise between expressing a large variety of policies and reducing the memory usage. We end up with a parameter optimization problem for which algorithms exist that guarantee a monotonous improvement of the policy towards a local optimum, even – in some cases – under partial observability.

EXAMPLE 5.1. What about the cars that need maintenance then?¹ A garage mechanic often knows the procedures (or algorithms) he has to follow to diagnose and repair a vehicle. But these procedures may vary depending on the model or the age of the vehicle. Thus, it can be more cost-effective to change the brake discs without testing them if their age is beyond an age limit. The methods we will discuss here aim at optimizing/learning parameters such as the age limit in cases where the – more or less complex – shape of the policy is fixed.

Chapter written by Olivier BUFFET.

1. See details in Chapter 1, section 1.1.

We can envision adapting various kinds of optimization algorithms. This chapter focuses on gradient algorithms, called *policy gradients*, which have been particularly studied. After a brief reminder about the notion of gradient (section 5.1), we will present two types of approaches: direct policy-gradient algorithms in section 5.2 and actor-critic policy gradients in section 5.3.

5.1. Reminder about the notion of gradient

Let us first come back to the general notions of gradient and gradient-following optimization algorithms before introducing the case of MDPs.

5.1.1. Gradient of a function

The gradient is a vector operator mapping a differentiable function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ to $\nabla f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ defined as

$$\nabla f(x_1, \dots, x_n) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(x_1, \dots, x_n) \\ \vdots \\ \frac{\partial f}{\partial x_n}(x_1, \dots, x_n) \end{bmatrix}. \quad (5.1)$$

The gradient of a function f at point x is a vector perpendicular to the level set containing $f(x)$,² its orientation being towards increasing values of the function and its norm indicating the variation “velocity” of f in that direction.

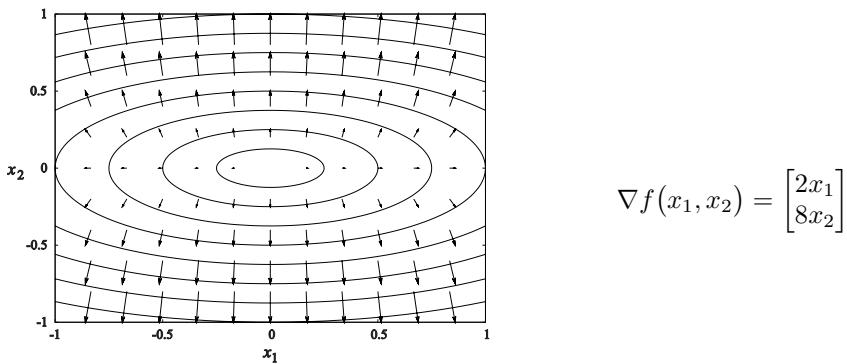


Figure 5.1. Level sets (isosurfaces) of function $f(x_1, x_2) = x_1^2 + 4x_2^2$ and the corresponding gradient vector field

2. The level set containing $f(x)$ is the set $\{y \in \mathbb{R}^n \text{ such that } f(y) = f(x)\}$.

5.1.2. Gradient descent

The gradient descent is an iterative algorithm that finds a local minimum of a function f whose gradient exists and is known (if f is convex, this minimum is global). The principle is to construct a sequence of points $(\mathbf{x}_n)_{n \in \mathbb{N}}$ by following at each point the direction opposite to the gradient so as to go “down” to the next point. Algorithm 5.1 gives the details of the process, which requires:

- a differentiable scalar function f ,
- a starting point \mathbf{x}_0 ,
- a step-size $\alpha > 0$, i.e. a multiplicative coefficient used to adjust the size of the steps of the gradient descent,
- often a threshold $\epsilon > 0$ used by a stopping criterion (here depending on the last two visited points).

Examples of stopping criteria are $\|\mathbf{x}_n - \mathbf{x}_{n-1}\| \leq \epsilon$ or $\|f(\mathbf{x}_n) - f(\mathbf{x}_{n-1})\| \leq \epsilon$. From now on we will evade this problem by considering an infinite loop.

Algorithm 5.1: Gradient descent($f, \mathbf{x}_0, \alpha, \epsilon$)

Initialization:

$n \leftarrow 0$

repeat

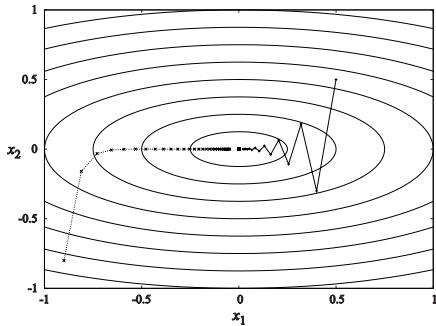
$n \leftarrow n + 1$
 $\mathbf{x}_n \leftarrow \mathbf{x}_{n-1} - \alpha \nabla f(\mathbf{x}_{n-1})$

until *stopping_criterion*($\mathbf{x}_n, \mathbf{x}_{n-1}, \epsilon$)

return x_n

Convergence proofs of the algorithms presented in this chapter depend on the evolution of the learning step-size α . It is common to require that $\sum_t \alpha_t = \infty$ and $\sum_t \alpha_t^2 < \infty$. Figure 5.2 illustrates the impact of the step-size by showing two sequences of points obtained using Algorithm 5.1 with different starting points and, in particular, employing different (constant) α step-sizes. If the step-size is too large, it provokes oscillations; if it is too small, the algorithm moves towards the minimum too slowly. In both cases, the convergence is slowed down. Finding a good step-size is a problem in itself, in particular because it can be dynamic. We will briefly come back to this topic in section 5.4.

NOTE 5.1. For a maximization problem, as will be the case in the remainder of this chapter, we speak of *gradient ascent*, the corresponding algorithm being different only in the sign preceding the step-size α .



	left	right
x_0	$[-.9, -.8]$	$[+.5, +.5]$
α	.1	.2
ϵ		.005

Both point sequences converge to the minimum $(0, 0)$ but with very different behaviors linked to their learning step-sizes.

Figure 5.2. Level sets of the function $f(x_1, x_2) = x_1^2 + 4x_2^2$, with two sequences of points obtained by a gradient descent with different starting points and different step-sizes

5.2. Optimizing a parameterized policy with a gradient algorithm

Historically, the reference algorithm for direct policy search using a gradient method is William's REINFORCE [WIL 87, WIL 92]. This work has then been generalized and improved, among others, by Baird and Moore [BAI 99a, BAI 99b], then by Baxter and Bartlett [BAX 01a, BAX 01b].

5.2.1. Application to MDPs: overview

In the case of an MDP, the objective is to maximize a performance measure depending on the (stochastic) policy π being applied. Let us, for example, denote $g(\pi)$ by a performance criterion such as the ones introduced in section 1.2.3. To be able to employ a gradient ascent, the policy is written as a function of a parameter vector θ : $\pi = h(\theta)$, so that the performance measure is a function $f(\theta) = g \circ h(\theta)$. If this function is differentiable, the conditions required to execute a gradient algorithm are met.

The choice of the parameterized form $\pi = h(\theta)$ defines a sub-space of the space of stochastic policies. Ideally, this sub-space should be as small as possible, but should still contain the best policies. In that sub-space, the optimal policy may not be deterministic anymore and a search may lead to a local optimum.

The gradient of f at point θ represents the sensitivity of the performance measure f to changes of the control parameters. With a known model of the MDP, we can envision computing the gradient exactly (see the GAMP algorithm [ABE 03]). But it is most often estimated through simulations, either because no model is available, or because an exact method would be too expensive (requiring to develop the entire state-action space). The approach is then called a stochastic gradient method.

The remainder of this section presents examples of forms that the parameterized policy can take. Then we will see in the following sections how the gradient can be estimated in different situations, which constitutes the main difficulty of the approach we are interested in.

5.2.1.1. First attempt to define a parameterized policy

A first idea to design a parameterized policy is simply to specify a scalar parameter $\theta_{s,a} (\geq 0)$ for each state-action pair, the policy being given by

$$q_\pi(a \mid s) = q(a \mid s; \boldsymbol{\theta}) = \frac{\theta_{s,a}}{\sum_{b \in A} \theta_{s,b}},$$

where $q_\pi(a \mid s)$ is the probability of executing action a in state s under stochastic policy π . The learning then has the goal of tuning the vector $\boldsymbol{\theta}$ as well as possible.

If this form has the advantage of being able to express any stochastic policy, this first solution has some major drawbacks:

- it induces a large number of parameters, hence a high-dimensional search space, what should usually be avoided;
- the gradient ascent is hampered by the non-negativity constraint on the parameters (the parameters should be prevented from becoming negative).

The following example addresses these drawbacks while presenting a rather generic form of parameterized policy.

5.2.1.2. Example of definition of a parameterized policy

Let us consider that, for each pair (s, a) , we have access to a feature vector $\phi_{s,a}$. In each state s , the policy has to provide a probability distribution over actions a as a function of the vectors $\phi_{s,a}$, what can be achieved in two steps:

- 1) calculating a scalar value $K_s(a)$ for each a (measuring the “importance” of a in the current situation),
- 2) turning these values in a probability distribution over actions.

EXAMPLE 5.2. A typical example of such a definition of a parameterized policy is:

- to take for $K_s(a)$ a linear combination of features of (s, a) – a dot product between $\phi_{s,a}$ and a parameter vector $\boldsymbol{\theta}$: $K_s(a) = \boldsymbol{\theta}^T \phi_{s,a}$,
- to take action probabilities from $K_s(a)$ using a Gibbs distribution as follows:

$$q(a \mid s; \boldsymbol{\theta}) = \frac{e^{K_s(a)}}{\sum_{b \in A} e^{K_s(b)}}. \quad (5.2)$$

Detailing the form taken by the feature vectors, Example 5.2 leads to the following two common formulations.

EXAMPLE 5.3. The state s being accessible, let us replace the vectors $\phi_{s,a}$ with matrices $\phi_{s,a}$: 1) of size $|S| * |A|$ and 2) null except for the (s, a) component: $\phi_{s,a}(s', a') = 1$ if $(s, a) = (s', a')$, 0 otherwise; we have then one parameter $\theta_{s,a}$ per pair (s, a) , so that

$$q(a \mid s; \boldsymbol{\theta}) = \frac{e^{\theta_{s,a}}}{\sum_{b \in A} e^{\theta_{s,b}}}. \quad (5.3)$$

EXAMPLE 5.4. If the state is known through a feature vector ϕ_s (e.g. observable variables), $\boldsymbol{\theta}$ can be decomposed as one vector $\boldsymbol{\theta}_a$ per action, which leads to

$$q(a \mid s; \boldsymbol{\theta}) = \frac{e^{\boldsymbol{\theta}_a^T \phi_s}}{\sum_{b \in A} e^{\boldsymbol{\theta}_b^T \phi_s}}. \quad (5.4)$$

The first case (Example 5.3) is a corrected – but still memory-consuming – version of the attempt of parameterization discussed in section 5.2.1.1. The second case (Example 5.4) is problematic if there exists some state s such that $\phi_s = \mathbf{0}$. Indeed, for all $\boldsymbol{\theta}$, this leads to $q_\pi(a \mid s) = 1/|\mathcal{A}|$ for all $a \in \mathcal{A}$: in this state s the policy cannot be modified. It is common practice to solve this problem by adding a non-zero constant feature³ (a 1 bit) to ϕ . But other parameterizations than these ones can be used: using multi-layer perceptrons, graphical models, etc.

Having given these examples of parameterized policies (a last example, with a continuous action space, is presented in section 5.2.6), let us have a look at how to estimate the gradient of such a policy.

5.2.2. Finite difference method

A first possible approach to estimate the gradient is the classical finite difference method. In the mono-dimensional case – for a differentiable function $f : \mathbb{R} \rightarrow \mathbb{R}$ – this amounts to writing the derivative of f at x as

$$f'(x) = \lim_{\epsilon \rightarrow 0} \frac{f(x + \epsilon) - f(x - \epsilon)}{2\epsilon}.$$

In our case where f can be accessed only through simulations, we have to choose a small $\epsilon > 0$ value and perform two series of simulations to estimate $f(x + \epsilon)$ and $f(x - \epsilon)$.

3. This is equivalent to the threshold parameter in a neuron following McCulloch and Pitts' model.

Using finite differences is not convenient for reinforcement learning because you have to perform $2n$ simulations to obtain an estimate of the gradient (n being the number of parameters in θ). The remainder of this chapter shows how gradient estimates of f can be calculated efficiently in an MDP setting.

5.2.3. Gradient estimation of f in an MDP, case of the finite time horizon

5.2.3.1. Time horizon 1

Let us first consider an MDP with finite time horizon of length 1 (a single decision has to be made) and whose initial state is sampled randomly according to a probability distribution d . The objective is to optimize

$$J(\pi) = E_\pi[r] = \sum_{(s,a,s') \in S \times A \times S} \underbrace{d(s)q_\pi(a | s)p(s' | s, a)}_{Pr(s,a,s')} r(s, a, s').$$

With the parameterization θ , this performance measure can be rewritten as

$$f(\theta) = \sum_{s,a,s'} d(s)q(a | s; \theta)p(s' | s, a)r(s, a, s').$$

Obviously, performing a gradient ascent requires, for all pairs (s, a) , that the function mapping θ to $q(a | s; \theta)$ be differentiable. Assuming that the likelihood ratios $\frac{\nabla q(a | s; \theta)}{q(a | s; \theta)}$ exist (and are bounded), the gradient of f is then written as⁴

$$\begin{aligned} \nabla E[r(s, a, s')] &= \nabla \left[\sum_{s,a,s'} d(s)q(a | s; \theta)p(s' | s, a)r(s, a, s') \right] \\ &= \sum_{s,a,s'} d(s)\nabla[q(a | s; \theta)]p(s' | s, a)r(s, a, s') \\ &= \sum_{s,a,s'} d(s) \left[\frac{\nabla q(a | s; \theta)}{q(a | s; \theta)} q(a | s; \theta) \right] p(s' | s, a)r(s, a, s') \\ &= \sum_{s,a,s'} \left[\frac{\nabla q(a | s; \theta)}{q(a | s; \theta)} r(s, a, s') \right] d(s)q(a | s; \theta)p(s' | s, a) \\ &= E \left[\frac{\nabla q(a | s; \theta)}{q(a | s; \theta)} r(s, a, s') \right]. \end{aligned}$$

4. Gradients are always taken with respect to vector θ .

Using this result, and with N independent and identically distributed – with distributions d , q and p – samples (s_i, a_i, s'_i) , an estimate of f 's gradient at point θ is

$$\tilde{\nabla} f(\theta) = \frac{1}{N} \sum_{i=1}^N \frac{\nabla q(a_i | s_i; \theta)}{q(a_i | s_i; \theta)} r(s_i, a_i, s'_i). \quad (5.5)$$

Using this result requires first computing $\frac{\nabla q(a | s; \theta)}{q(a | s; \theta)}$ (sometimes called the log-gradient of the policy because it can be written as $\nabla \log q(a | s; \theta)$). In the case of formulation (5.2), for example, we have

$$\begin{aligned} & \frac{1}{q(a | s; \theta)} \frac{\partial q(a | s; \theta)}{\partial \theta_i} \\ &= \frac{1}{q(a | s; \theta)} \left(\phi_{s,a,i} \frac{e^{\theta^T \phi_{s,a}}}{\sum_{a' \in \mathcal{A}} e^{\theta^T \phi_{s,a'}}} - \frac{e^{\theta^T \phi_{s,a}}}{\left[\sum_{a' \in \mathcal{A}} e^{\theta^T \phi_{s,a'}} \right]^2} \sum_{b \in \mathcal{A}} \phi_{s,b,i} e^{\theta^T \phi_{s,b}} \right) \\ &= \phi_{s,a,i} - \sum_{b \in \mathcal{A}} \phi_{s,b,i} q(b | s; \theta), \end{aligned}$$

so that we can write

$$\frac{\nabla q(a | s; \theta)}{q(a | s; \theta)} = \phi_{s,a} - \sum_{b \in \mathcal{A}} q(b | s; \theta) \phi_{s,b}. \quad (5.6)$$

The practical usage of equation (5.5) in a gradient ascent algorithm also requires adding to Algorithm 5.1 a loop generating these N samples to estimate the current parameter vector's gradient.⁵

NOTE 5.2. Such a process estimating a value by sampling is usually called a Monte Carlo method (see also section 2.3). In the present case, trajectories are being simulated along a Markov chain over the states of the MDP, the probability to transition from s to s' being $Pr(s' | s) = \sum_{a \in A} p(s' | s, a) q(a | s; \theta)$. This particular case is more precisely referred to as a Monte Carlo Markov Chain (MCMC) method [MAC 03].

5.2.3.2. Time horizon T

In the previous section, we have shown how to calculate an estimate of the gradient of f for an MDP with a time horizon of length 1. We now extend this computation to the case of an MDP of finite time horizon T . The performance

5. Do not forget to change the sign in front of α as we are considering an ascent, not a descent.

measure being considered is the expected discounted reward with $\gamma \in (0, 1]$. Let $V_t(s)$ be the expected discounted reward from t to T while starting from state s . Then, for all $s \in \mathcal{S}$ and all $t \in \{1, \dots, T-1\}$,

$$\begin{aligned} f(\boldsymbol{\theta}) &= E[V_0(s_0)] = \sum_s d(s)V_0(s), \\ V_t(s) &= E[r(s_t, a_t, s_{t+1}) + \gamma V_{t+1}(s_{t+1}) \mid s_t = s] \\ &= \sum_{a, s'} q(a \mid s; \boldsymbol{\theta}) p(s' \mid s, a) (r(s, a, s') + \gamma V_{t+1}(s')), \\ V_T(s) &= 0. \end{aligned}$$

The gradient of f can then be calculated with

$$\begin{aligned} \nabla f(\boldsymbol{\theta}) &= \sum_s d(s) \nabla V_0(s) = E[\nabla V_0(s_0)], \\ \nabla V_t(s) &= \sum_{a, s'} \nabla [q(a \mid s; \boldsymbol{\theta}) p(s' \mid s, a) (r(s, a, s') + \gamma V_{t+1}(s'))] \\ &= \sum_{a, s'} q(a \mid s; \boldsymbol{\theta}) p(s' \mid s, a) \left(\frac{\nabla q(a \mid s; \boldsymbol{\theta})}{q(a \mid s; \boldsymbol{\theta})} (r(s, a, s') \right. \\ &\quad \left. + \gamma V_{t+1}(s')) + \gamma \nabla V_{t+1}(s') \right) \\ &= E \left[\frac{\nabla q(a_t \mid s_t; \boldsymbol{\theta})}{q(a_t \mid s_t; \boldsymbol{\theta})} (r_t + \gamma V_{t+1}(s_{t+1})) + \gamma \nabla V_{t+1}(s_{t+1}) \mid s_t = s \right], \\ \text{where } r_t &= r(s_t, a_t, s_{t+1}), \\ \nabla V_T(s) &= \mathbf{0}. \end{aligned}$$

Developing the computation of $\nabla f(\boldsymbol{\theta})$ and factoring the terms associated with each reward, we obtain

$$\nabla f(\boldsymbol{\theta}) = E \left[\sum_{t=0}^{T-1} \gamma^t r_t \sum_{t'=0}^{t-1} \frac{\nabla q(a_{t'} \mid s_{t'}; \boldsymbol{\theta})}{q(a_{t'} \mid s_{t'}; \boldsymbol{\theta})} \right].$$

As with the length 1 horizon, the gradient can be estimated through sampling. While acquiring a sample s_0, a_0, \dots, s_T , we calculate a non-biased estimator \mathbf{g} of $\nabla f(\boldsymbol{\theta})$ by iterating

$$\begin{aligned} \mathbf{z}_{t+1} &= \mathbf{z}_t + \frac{\nabla q(a_t \mid s_t; \boldsymbol{\theta})}{q(a_t \mid s_t; \boldsymbol{\theta})}, \\ \mathbf{g}_{t+1} &= \mathbf{g}_t + \gamma^t r_t \mathbf{z}_t, \end{aligned}$$

where \mathbf{z} is an eligibility trace (local estimate of the policy's log-gradient) and both sequences are initialized with the null vector. A better estimator (with smaller variance) is obtained by computing the mean of N estimators \mathbf{g} calculated independently.

Algorithm 5.2 sums up the operations to execute so as to perform a gradient ascent in a finite horizon MDP. The instant reward r and next state s' are obtained through interactions with a real environment or a simulator.

Algorithm 5.2: Gradient ascent for finite horizon MDPs ($\boldsymbol{\theta}, \alpha, N$)

```

Initialization:
 $\boldsymbol{\theta}_0 \leftarrow \boldsymbol{\theta}$ 
 $i \leftarrow 0$ 
/* Gradient Ascent's Loop.
for ever do
     $i \leftarrow i + 1$ 
     $\tilde{\nabla}f \leftarrow \mathbf{0}$ 
    /* Loop acquiring  $N$  samples to estimate the gradient. */
    for  $n = 1, \dots, N$  do
         $\mathbf{z} \leftarrow \mathbf{0}$ 
         $\mathbf{g} \leftarrow \mathbf{0}$ 
        GetInitialState( $s$ ) /* Sample initial state from  $d(\cdot)$ . */
        /* Loop simulating the MDP to obtain a sample. */
        for  $t = 0, \dots, T - 1$  do
            Sample( $a, q(\cdot | s; \boldsymbol{\theta})$ )
            Execute( $a$ )
            Get( $s', r$ )
             $\mathbf{z} \leftarrow \mathbf{z} + \frac{\nabla q(a | s; \boldsymbol{\theta})}{q(a | s; \boldsymbol{\theta})}$ 
             $\mathbf{g} \leftarrow \mathbf{g} + \gamma^t r \mathbf{z}$ 
             $s \leftarrow s'$ 
        end
         $\tilde{\nabla}f \leftarrow \tilde{\nabla}f + \frac{1}{N} \mathbf{g}$ 
    end
     $\boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta}_{i-1} + \alpha \tilde{\nabla}f$ 
return  $x_n$ 

```

There are some comments about this algorithm:

- The initial vector $\boldsymbol{\theta}$ should ideally enable a good exploration.

A null vector is often a good solution, except, for example, when utilizing a multi-layer perceptron, since $\boldsymbol{\theta} = \mathbf{0}$ then corresponds to a null point of the gradient (so that the gradient ascent remains stuck since the direction to follow is the null vector). It is then preferred to use a random vector with a small norm.

- The choice of N is important: if too small, the estimate of the gradient will be poor; if too large, the estimation will be too time-consuming.

An interesting problem is to find an estimator whose variance is smaller for the same number of samples. Intuitively, this amounts to finding an estimator whose probability to give an estimate close to the true value is higher.

- An interesting property of the gradient estimation is that, assuming that a simulator is being used, it is very easy to parallelize: N processors can perform N simulations in parallel. This results in a linear speed-up.

5.2.4. Extension to the infinite time horizon: discounted criterion, average criterion

In an MCMC method, we have to ensure that the visiting frequency of states reflects the real probability of being in these states. In the finite time horizon setting, it was sufficient to start simulations in a possible initial state randomly sampled according to distribution $d(\cdot)$, then to apply the policy and to transition according to the model until the next restart. This way, each state is visited with a frequency representative of the probability to be in that state at any instant.

If the time horizon is infinite, complete trajectories cannot be used anymore – because they are endless – and states are visited with a representative frequency only after a sufficiently long simulation time. In the case of the Markov chain represented by Figure 5.3 for example, the average waiting time before arriving in state s_2 for the first time is all the longer that $\epsilon \in (0, 1]$ is small (s_0 being the initial state).

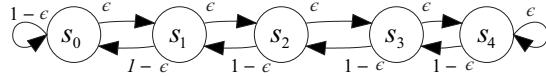


Figure 5.3. Markov Chain whose time before establishing the steady state is all the longer that ϵ is small

To extend gradient algorithms to infinite time horizon cases, it is common to ensure that such a steady state exists. This requires that the following property holds: for a given parameter vector θ , the induced Markov chain is ergodic (see Figure 5.4), which guarantees the existence of a unique distribution over states P_{stat} such that

$$\forall s \in S, \quad P_{\text{stat}}(s) = \sum_{s' \in S} \sum_{a \in A} p(s | a, s') q(a | s', \theta) P_{\text{stat}}(s').$$

For example, Figure 5.4 shows two typical non-ergodic chains.

In the remainder of this section, we only consider the case $\gamma = 1$.



Figure 5.4. A Markov chain is ergodic if 1) there exists a path from any state s_i to any state s_j (which is not the case in the left figure) and 2) it is aperiodic (which is not the case in the right figure)

5.2.4.1. Case of a regenerative process

The problem at hand is that of estimating the gradient. A first approach can be used when a recurrent state s_r exists, i.e. a state to which the system necessarily comes back (and which can be identified). The system is said to be *regenerative process*. In such a setting, the samples are sequences of states between two visits of state s_r , and the (non-biased) gradient estimate is similar to the case of the length T time horizon (section 5.2.3.2). Yet, it is difficult to make sure that a process is regenerative and to specify which recurrent state s_r to use. We will therefore search for other means to estimate the gradient.

Sampling from infinite sequences of states, it is not possible to take samples one after the other to update the gradient estimate. We need to find another way to estimate the gradient using partial samples.

5.2.4.2. Using a moving window

A second approach consists of taking as samples state sequences of fixed length n . More precisely, at time t , the eligibility trace is calculated using the last n time steps:

$$\begin{aligned} \mathbf{z}_t(n) &= \sum_{t'=t-n+1}^t \frac{\nabla q(a_{t'} | s_{t'}; \boldsymbol{\theta})}{q(a_{t'} | s_{t'}; \boldsymbol{\theta})} \\ &= \mathbf{z}_{t-1}(n) + \frac{\nabla q(a_t | s_t; \boldsymbol{\theta})}{q(a_t | s_t; \boldsymbol{\theta})} \\ &\quad - \frac{\nabla q(a_{t-n} | s_{t-n}; \boldsymbol{\theta})}{q(a_{t-n} | s_{t-n}; \boldsymbol{\theta})}. \end{aligned}$$

The resulting eligibility trace can always be calculated in an iterative manner, but this requires keeping a memory of the last $n + 1$ steps. From there, the gradient estimate after T steps is given by

$$\tilde{\nabla} f(\boldsymbol{\theta}) = \frac{1}{T-n+1} \sum_{t=n-1}^{T-1} \mathbf{z}_t(n) r_t.$$

This estimate is biased. The bias diminishes when n grows, but at the expense of variance which diverges. The choice of n therefore corresponds to a compromise between bias and variance.

5.2.4.3. Using a discount factor

A third approach enables not having to memorize n steps. It amounts to computing a discounted and non-truncated eligibility trace as follows:

$$\mathbf{z}_{t+1} = \beta \mathbf{z}_t + \frac{\nabla q(a_{t+1} | s_{t+1}; \boldsymbol{\theta})}{q(a_{t+1} | s_{t+1}; \boldsymbol{\theta})},$$

where $\mathbf{z}_0 = 0$ and $\beta \in [0, 1)$ is a discount factor. After T time steps, the (biased) gradient estimate is then given by

$$\tilde{\nabla}_\beta f(\boldsymbol{\theta}) = \frac{1}{T} \sum_{t=0}^{T-1} \mathbf{z}_t(\beta) r_t.$$

The compromise between bias and variance still exists, depending here on the β parameter: the bias tends to 0 when β tends to 1, but then the variance diverges.

On this basis, we can propose a gradient ascent algorithm that, as previously encountered algorithms, alternates between a gradient estimation phase and a gradient-following step. But an “online” gradient ascent algorithm can be proposed as well, which follows the gradient direction after each new experience, and with no computation of a gradient estimate since an *instant gradient* is used instead ($\mathbf{z}_t(\beta) r_t$). This algorithm is known as OLpomdp [BAX 01b] (“OL” for “*online*” and “*pomdp*” because it is still valid in a partially observable setting⁶) and is described in Algorithm 5.3. This algorithm has the advantage of being very simple to implement, and proved to be efficient in various applications (see Chapter 15).

5.2.5. Partially observable case

Let us focus for a moment on applying such gradient algorithms on partially observable MDPs (with a finite set of observations). Various options are possible, such as:

- coming back to an MDP over belief states, as in Chapter 7,
- optimizing a policy whose input depends on the history of past observations and actions.

6. We will come back to the partially observable setting in the next section.

Algorithm 5.3: $\text{OLpomdp}(\boldsymbol{\theta}, \alpha, \beta)$
 Online gradient ascent for finite horizon (PO)MDPs

```

Initialization:
 $\boldsymbol{\theta}_0 \leftarrow \boldsymbol{\theta}; \mathbf{z} \leftarrow \mathbf{0}; \mathbf{g} \leftarrow \mathbf{0}$ 
GetInitialState( $s$ )
for ever do
  Sample( $a, q(\cdot | s; \boldsymbol{\theta})$ )
  Execute( $a$ )
  Get( $s', r$ )
   $\mathbf{z} \leftarrow \beta \mathbf{z} + \frac{\nabla q(a | s; \boldsymbol{\theta})}{q(a | s; \boldsymbol{\theta})}$ 
   $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha r \mathbf{z}$ 
   $s \leftarrow s'$ 
return  $\boldsymbol{\theta}$ 
  
```

If the time horizon is finite, both options are conceivable by going back to a finite state space MDP. If the time horizon is infinite, the first option requires knowing the model to estimate the belief state at each time step and extending gradient ascent algorithms to continuous state spaces. We will consider the second option, limiting the input to the last perceived observation o .

To see how to cover the POMDP case, let us notice that partial observability covers two aspects:

- 1) the information received concerning the state is incomplete,
- 2) this information is noisy.

Yet the noise can be “extracted” from the observation function and put in the state: if the state s is defined as a set of random variables, the noise is just an additional (non-observed) variable N which is independent of the past and of other variables. Let us note the new state $\sigma (= (s, n))$. This being done, the observation function becomes deterministic: to each state σ corresponds a unique observation $o = O'(\sigma)$. From there, composing $O(\cdot)$ and $q(a | o; \boldsymbol{\theta})$ leads to the MDP setting where the loss of information due to the partial observability is part of the function approximator that defines the policy:

$$\begin{aligned}
 q(a | \sigma; \boldsymbol{\theta}) &= \sum_{o'} q(a | o'; \boldsymbol{\theta}) \cdot O'(o' | \sigma) \\
 &= q(a | O'(\sigma); \boldsymbol{\theta}).
 \end{aligned}$$

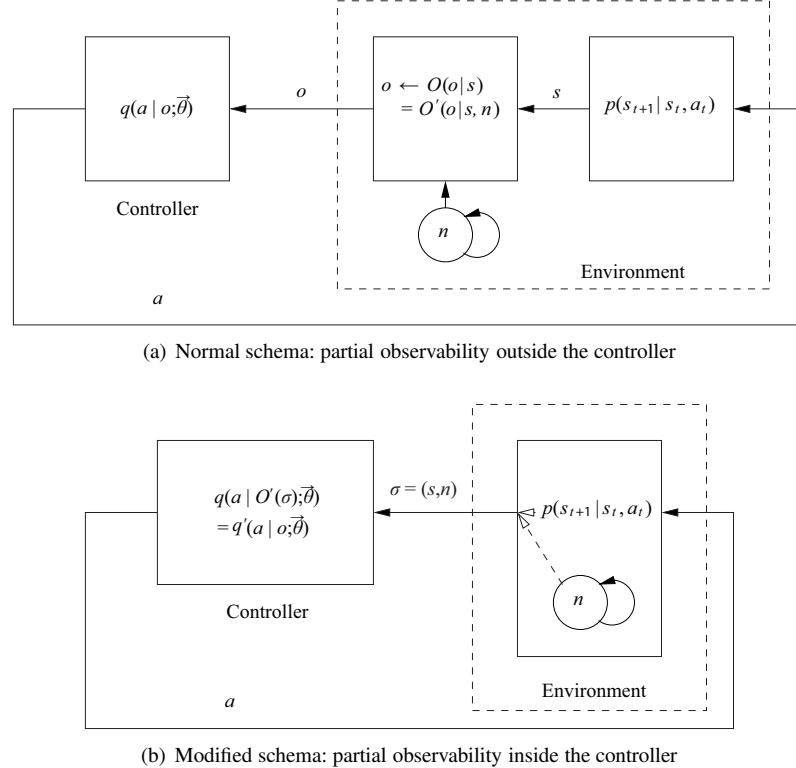


Figure 5.5. Two schemata representing the same partially observable MDP and its controller

Figure 5.5(a) presents a POMDP (and the corresponding associated controller) in a schematic manner by highlighting the “noise” (variable N) and “incomplete perception” (function O') components. Figure 5.5(b) shows the same problem as an MDP and a modified controller.

This interpretation explains how gradient ascent algorithms often extend to POMDPs. This is, in particular, the case for OL_{pomdp} (Algorithm 5.3) by replacing each occurrence of state s by observation o . But this is not the case, for example, with algorithms for regenerative processes since partial observability may prevent from identifying the recurrent state.

5.2.6. Continuous action spaces

Another interesting case is that of continuous action spaces. A lot of effort has been put into solving MDPs with large, if not continuous, state spaces. When

considering large action spaces, it seems natural to extend value-function-based approaches. However, in a given state s , finding a best action requires solving the optimization problem $\max_{a \in A} Q(s, a)$. This is a time-consuming task that needs to be performed online.

We may circumvent this problem by using an explicit policy (e.g. a controller derived using a policy-gradient) rather than a relying on a value-function [KIM 98, PET 03]. Yet, as we are considering policy-gradient algorithms, the policy should not output the assumed best action but sample actions according to a distribution centered on this action (because solution algorithms typically require that all actions be sampled during the gradient estimation).

For example, if the action space is 1-dimensional (\mathbb{R}) the controller can use a normal distribution whose mean μ is the output of a function approximator $(\mu(s; \theta) = \phi(s)^T \theta)$ and whose standard deviation σ is an extra parameter. In state s , an action a is sampled according to $\mathcal{N}(\mu(s; \theta), \sigma^2)$. Let us then detail the computation of policy derivatives for parameter θ_i :

$$\begin{aligned}\frac{\partial \mathcal{N}(\mu(s; \theta), \sigma^2)}{\partial \theta_i} &= \frac{\partial \left[\frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-(a-\mu(s; \theta))^2}{2\sigma^2}\right) \right]}{\partial \theta_i} \\ &= \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-(a-\mu(s; \theta))^2}{2\sigma^2}\right) \frac{\partial \frac{-(a-\mu(s; \theta))^2}{2\sigma^2}}{\partial \theta_i} \\ &= \mathcal{N}(\mu(s; \theta), \sigma^2) \frac{(a - \mu(s; \theta))}{\sigma^2} \phi_i(s),\end{aligned}$$

and for parameter σ :

$$\begin{aligned}\frac{\partial \mathcal{N}(\mu(s; \theta), \sigma^2)}{\partial \sigma} &= \frac{\partial \left[\frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-(a-\mu(s; \theta))^2}{2\sigma^2}\right) \right]}{\partial \sigma} \\ &= \frac{\partial \frac{1}{\sqrt{2\pi}\sigma}}{\partial \sigma} \exp\left(\frac{-(a-\mu(s; \theta))^2}{2\sigma^2}\right) + \frac{1}{\sqrt{2\pi}\sigma} \frac{\partial \exp\left(\frac{-(a-\mu(s; \theta))^2}{2\sigma^2}\right)}{\partial \sigma} \\ &= \mathcal{N}(\mu(s; \theta), \sigma^2) \frac{(a - \mu(s; \theta))^2 - \sigma^2}{\sigma^3}.\end{aligned}$$

This gives us simple expressions that can be employed in the policy-gradient algorithms presented in this chapter.

5.3. Actor-critic methods

An actor-critic method distinguishes two components: 1) the actor, i.e. the policy being learned, and 2) the critic, which evaluates the actor's efficiency to help improve its policy. The algorithms we mentioned up to now in this chapter were “actor only” (deprived of critic) while algorithms such as the Q -learning are “critic only” (because the policy is greedy with respect to the action value function). We will now see that a critic using a representation of the value function can be usefully employed in a policy-gradient algorithm.

The algorithms encountered in the previous section are based on a gradient estimate whose variance is important, which makes optimizing the parameters difficult. An essential objective is then to reduce this variance, objective that can be reached, for example, thanks to biased estimates based on the value function (or an approximation thereof). This is how algorithms such as VAPS (*Value and Policy Search*) [BAI 99a, BAI 99b] and its descendants have been introduced (let us mention, e.g. [SUT 00, KON 03]). They combine a gradient ascent and an approximation of the value function, making them part of the family of actor-critic architectures.

We now come back to the finite time horizon setting, with a discount factor γ . We first present the gradient estimate using Q -values, then explain what happens when Q -values are approximated and finally describe an actor-critic algorithm.

5.3.1. A gradient estimator using the Q -values

Before anything else, let us introduce $d^\pi(s)$ (not to be mistaken with $d(s)$), which represents the weight of state s given 1) the probability to meet this state in the future and 2) the discount factor γ : $d^\pi(s) = \sum_{t=0}^{\infty} \gamma^t \Pr(s_t = s \mid d(\cdot); \pi)$.

To calculate a gradient estimate as a function of the Q -values $Q(s, a)$, let us start with the Bellman equation:

$$V^\theta(s) = \sum_{a \in \mathcal{A}} q(a \mid s; \theta) \sum_{s' \in \mathcal{S}} p(s' \mid s, a) [r(s, a, s') + \gamma V^\theta(s)]$$

and let us derive its gradient:

$$\begin{aligned} \nabla V^\theta(s) &= \sum_{a \in \mathcal{A}} \nabla q(a \mid s; \theta) \sum_{s' \in \mathcal{S}} p(s' \mid s, a) [r(s, a, s') + \gamma V^\theta(s)] \\ &\quad + \sum_{a \in \mathcal{A}} q(a \mid s; \theta) \sum_{s' \in \mathcal{S}} p(s' \mid s, a) \nabla [r(s, a, s') + \gamma V^\theta(s)] \end{aligned}$$

$$\begin{aligned}
&= \sum_{a \in \mathcal{A}} q(a | s; \boldsymbol{\theta}) \sum_{s' \in \mathcal{S}} p(s' | s, a) \\
&\quad \times \left(\frac{\nabla q(a | s; \boldsymbol{\theta})}{q(a | s; \boldsymbol{\theta})} [r(s, a, s') + \gamma V^{\pi_{\boldsymbol{\theta}}}(s)] + \gamma \nabla V^{\boldsymbol{\theta}}(s) \right) \\
&= \sum_{a \in \mathcal{A}} q(a | s; \boldsymbol{\theta}) \sum_{s' \in \mathcal{S}} p(s' | s, a) \left(\frac{\nabla q(a | s; \boldsymbol{\theta})}{q(a | s; \boldsymbol{\theta})} Q^{\boldsymbol{\theta}}(s, a) + \gamma \nabla V^{\boldsymbol{\theta}}(s) \right).
\end{aligned}$$

We recognize in this last equation a Bellman equation defining the gradient of the value function using $\frac{\nabla q(a | s; \boldsymbol{\theta})}{q(a | s; \boldsymbol{\theta})} Q^{\boldsymbol{\theta}}(s, a)$ (instead of $r(s, a, s')$ in the definition of the value function). From there, we deduce that the gradient can also be written as

$$\nabla V^{\boldsymbol{\theta}}(s) = E \left[\sum_{t=0}^{\infty} \gamma^t \frac{\nabla q(a_t | s_t; \boldsymbol{\theta})}{q(a_t | s_t; \boldsymbol{\theta})} Q^{\boldsymbol{\theta}}(s, a) \mid s_0 = s; \pi \right],$$

which leads us to an estimator of the gradient of f :

$$\begin{aligned}
\nabla f(\boldsymbol{\theta}) &= \sum_{s \in \mathcal{S}} d(s) \nabla V^{\boldsymbol{\theta}}(s) \\
&= \sum_{s \in \mathcal{S}} d^{\pi}(s) \sum_{a \in \mathcal{A}} q(a_t | s_t; \boldsymbol{\theta}) \left[\frac{\nabla q(a_t | s_t; \boldsymbol{\theta})}{q(a_t | s_t; \boldsymbol{\theta})} Q^{\boldsymbol{\theta}}(s, a) \right] \quad (5.7) \\
&= \sum_{s \in \mathcal{S}} d^{\pi}(s) \sum_{a \in \mathcal{A}} \nabla q(a_t | s_t; \boldsymbol{\theta}) Q^{\boldsymbol{\theta}}(s, a).
\end{aligned}$$

NOTE 5.3. The same results can be obtained using as a criterion the average reward per time step [SUT 00].

5.3.2. Compatibility with an approximate value function

A motivation for employing direct policy search methods is to reduce the need for memory and computation time by limiting the search space to a sub-space of possible policies. This naturally leads to considering what happens in an actor-critic method when approximating the function $Q^{\boldsymbol{\theta}}(s, a)$ instead of making an exact computation. In particular, does the previously introduced estimator remain valid?

5.3.2.1. Approximating $Q^{\boldsymbol{\theta}}$

Let $Q^{\boldsymbol{w}} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ be this approximation of $Q^{\boldsymbol{\theta}}$. Learning the parameters \boldsymbol{w} can be achieved by going through the pairs (s, a) while updating \boldsymbol{w} by following the direction:

$$\delta \boldsymbol{w} \propto \nabla [\hat{Q}^{\pi}(s, a) - Q^{\boldsymbol{w}}(s, a)]^2 \propto [\hat{Q}^{\pi}(s, a) - Q^{\boldsymbol{w}}(s, a)] \nabla Q^{\boldsymbol{w}}(s, a),$$

where $\hat{Q}^\pi(s, a)$ is a non-biased estimate of $Q^\pi(s, a)$ and \propto is the “proportional-to” symbol. Once a local optimum has been reached, we have

$$\sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} q(a | s; \boldsymbol{\theta}) [Q^\pi(s, a) - Q^w(s, a)] \nabla Q^w(s, a) = \mathbf{0}. \quad (5.8)$$

5.3.2.2. Compatibility of the approximators

The gradient of f from equation (5.7) appears in the above equation (5.8) if it holds that

$$\nabla Q^w(s, a) = \frac{\nabla q(a | s; \boldsymbol{\theta})}{q(a | s; \boldsymbol{\theta})}. \quad (5.9)$$

THEOREM 5.1 [SUT 00]. *If this condition (5.9) – called compatibility condition – holds, then the gradient of f can be written as*

$$\nabla f(\boldsymbol{\theta}) = \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} \nabla q(a_t | s_t; \boldsymbol{\theta}) Q^w(s, a).$$

Proof. From (5.8) and (5.9), we have

$$\sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} \nabla q(a | s; \boldsymbol{\theta}) [Q^\pi(s, a) - Q^w(s, a)] = \mathbf{0}.$$

By subtracting this expression (of null value) to (5.7), we get

$$\begin{aligned} \nabla f(\boldsymbol{\theta}) &= \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} \nabla q(a_t | s_t; \boldsymbol{\theta}) Q^\pi(s, a) \\ &\quad - \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} \nabla q(a | s; \boldsymbol{\theta}) [Q^\pi(s, a) - Q^w(s, a)] \\ &= \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} \nabla q(a_t | s_t; \boldsymbol{\theta}) Q^w(s, a). \end{aligned} \quad \square$$

Theorem 5.1 allows us to find a parameterization of the value function which, given the chosen form for the policy, guarantees that the policy-gradient estimate is not biased. With the example of equation (5.2), taking the log-gradient (5.6) we immediately have a possible parameterization:

$$Q^w(s, a) = \mathbf{w}^T \left[\phi_{s,a} - \sum_{b \in \mathcal{A}} q(b | s; \boldsymbol{\theta}) \phi_{s,b} \right]. \quad (5.10)$$

Q^w must therefore be linear in the same features as the policy, with a normalization to a null mean for each state. Observing that

$$\sum_{a \in \mathcal{A}} q(a | s; \theta) Q^w(s, a) = 0,$$

it appears that the function being approximated by $Q^w(s, a)$ is the *advantage* function $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$ [BAI 93] rather than the Q -function. This is not surprising since, given a state s , the relative interest of actions is given by their respective relative values.

5.3.3. An actor-critic algorithm

As can be observed in Algorithm 5.4, the principle of the gradient ascent algorithm using an approximation of the value function is simple. Indeed, each iteration consists of:

- 1) acquiring experience (samples);
- 2) using these samples to calculate a vector w_k such that Q^{w_k} approximates Q^{π_k} as well as possible (using one of the algorithms presented in Chapter 3);
- 3) following the gradient direction calculated thanks to Q^{w_k} to obtain a new vector θ_{k+1} and therefore a new policy π_{k+1} .

If the procedure computing w_k is iterative, it can be initialized with the solution w_{k-1} since the function to approximate usually does not change much with a gradient step (between π_{k-1} and π_k).

Algorithm 5.4: Gradient ascent with value function approximation (actor-critic algorithm) $(\theta, (\alpha_k)_{k=0}^\infty)$

Initialization:
 $i \leftarrow 0; \theta_0 \leftarrow \theta$
for ever do

$k \leftarrow k + 1$
 Acquire experience (new samples)
 $w_k \leftarrow \text{solutionOf}($

$$\sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} q(a | s; \theta_k) [Q^{\pi_k}(s, a) - Q^w(s, a)] \nabla Q^w(s, a) = \mathbf{0}$$
 $)$
 $\theta_{k+1} \leftarrow \theta_k + \alpha_k \sum_{s \in \mathcal{S}} d^{\pi_k}(s) \sum_{a \in \mathcal{A}} \nabla q(a | s; \theta_k) Q^{w_k}(s, a)$

return θ_k

THEOREM 5.2 [SUT 00]. *q and Q^w are two differentiable function approximators satisfying the compatibility condition (5.9) such that*

$$\max_{\theta, s, a, i, j} \left| \frac{\partial^2 q(a | s; \theta)}{\partial \theta_i \partial \theta_j} \right| < B < \infty.$$

Let $(\alpha_k)_{k=0}^\infty$ be a sequence of step sizes such that $\lim_{k \rightarrow \infty} \alpha_k = 0$ and $\sum_{k=0}^\infty \alpha_k = \infty$. Then, for all MDP with bounded rewards, the sequence $(\theta_k)_{k=0}^\infty$ calculated by Algorithm 5.4 converges such that $\lim_{k \rightarrow \infty} \nabla f(\theta_k) = \mathbf{0}$.

5.4. Complements

These policy-gradient ascents (with or without approximation of the value function) raise various problems whose resolution would make it possible to improve existing algorithms. We now briefly discuss some of these problems.

Variance reduction

In a stochastic gradient method, a first problem is to find a good gradient estimator, which means finding an estimator:

- with a small bias, i.e. which, when looking for x , estimates $x + \delta x$ with a small $|\delta x|$;
- with a small variance, i.e. very stable (returning a value close to x with a high probability).

The existence of a bias does not prevent the algorithm from behaving properly: the main point is to follow an ascent direction (in our maximization setting), not necessarily the gradient direction. It is even sometimes better to introduce a bias in an estimator to reduce the variance. This may reduce the need for samples.

We have already mentioned the fact that the use of an approximation of the value function aims at reducing the variance of the gradient estimate. For the same reasons, it is also common in direct algorithms to subtract to all rewards a *baseline* reward, usually the estimated average reward. Greensmith *et al.* [GRE 02] have yet proved that the average reward is not always the best choice. [MUN 06] discusses in more details the problem of variance reduction in actor-critic algorithms.

Natural gradient

Let us now turn to a specific difficulty: the fact that the gradient does not necessarily provides us with the steepest ascent direction. We intuitively describe this difficulty and an approach to get around it. More details can be found in [AMA 98, KAK 02, BAG 03, PET 05, BHA 09], a currently popular algorithm being NAC (*Natural Actor-Critic*).

By choosing a parameterization of the policy, we in fact define 1) a sub-set of the set of stochastic policies and 2) how to cover it. From a mathematical point of view, such an object is called a *manifold*. Typical 3D manifolds include the sphere, the torus or the Klein bottle. We can see on these examples that manifolds are not necessarily Euclidean spaces. This is precisely the reason why the gradient ∇f at point $\boldsymbol{\theta}$ is not necessarily the steepest ascent direction. A different parameterization of the same policy space could give a different direction.

In a manifold equipped with a Riemannian metric structure, the steepest ascent direction is the *natural gradient*, which is obtained by correcting the gradient using a matrix that represents the “local distortion” of the policy space at point $\boldsymbol{\theta}$. This matrix is the *Fischer information matrix*, given in our case by

$$F_s(\boldsymbol{\theta}) = E_{q(a|s;\boldsymbol{\theta})} \left[\frac{\partial \log q(a | s; \boldsymbol{\theta})}{\partial \theta_i} \frac{\partial \log q(a | s; \boldsymbol{\theta})}{\partial \theta_j} \right]. \quad (5.11)$$

The steepest ascent direction is then

$$\tilde{\nabla} f(\boldsymbol{\theta}) = F(\boldsymbol{\theta})^{-1} \nabla f(\boldsymbol{\theta}). \quad (5.12)$$

In practice, using the natural gradient or an approximation of the value function (or both at the same time) can make the computations much more costly for each sample. In a problem where 1) the cost of each new sample is significant or 2) the computation time remains negligible compared to the time taken to get a sample, these more complex methods should be preferred in order to find a good solution with few samples. If, on the contrary, samples can be obtained quickly and at a low cost – for example, when a fast simulator of the system is available – it is then likely that a simpler algorithm (such as `OLpomdp`) will prove to find a good solution faster (see Chapter 15, section 15.3).

Step-size adaptation, stopping criterion

Any improvement of generic gradient algorithms is a candidate to also improve policy-gradient algorithms used for MDPs. We can mention the problem of adapting the learning step-size. The step-size can be adapted while learning, for example, by performing a line search to find a local optimum along the direction of the current gradient. The step-size can also be different for each dimension of the parameter space (we can speed up the search in one dimension while slowing down in another dimension). To have a good overview of this topic, please refer to [SCH 06].

More generally, the questions raised when employing an optimization algorithm are raised here as well. Here are two examples we will not go further into:

- how to define a stopping criterion, two classical options being described in section 5.1.2;

- what is the algorithm's *anytime* behavior, i.e. what is its capacity to provide a good solution "early" and to improve it progressively.

Using a memory

We have seen that some algorithms make it possible to optimize a controller making decisions based on the current observation, which can be naturally extended to any finite horizon history. When the system's model is known, the belief state – the probability distribution over possible current states – can be calculated and used to make a decision, with sometimes the need to work with a continuous state space [COQ 09].

Another possible approach is to add a memory to the controller. It can be an internal state variable, a recurrent loop in a neural network or a state automaton. How to use this memory (memorize a precise event, etc) is not pre-defined: this decision will be made as part of the learning process. In the context of policy-gradient algorithms, the following two works use finite state automata as their memory:

- in [MEU 99], this automaton is a control automaton, the same structure containing the memory and the controller, this structure (and therefore the memory size) being fixed in advance;
- in [ABE 02, ABE 03], an independent automaton serves as a memory controller (managing an internal state), a second controller using the current internal state and the current observation to decide which action to perform.

If using a memory may be essential to efficiently solve some problems, these approaches are still uncommon and difficult to use. Indeed, not only does adding a memory contribute to the search space explosion, but also there is initially no hint telling how to use this memory (which information should be retained).

Guidance

In some cases, a good controller is known that can serve as a guide while learning. We can distinguish at least three ways to use this guide:

- imitation: only the guide's decision is observed; we cannot therefore experiment and evaluate other decisions; the only possible kind of learning is a simple imitation of the guide: a supervised learning;
- imitation + exploration: the guide's decisions are mixed with an exploratory policy, so that all possible decisions will be experimented; the reinforcement learning algorithm may have to be adapted to take into account the fact that the learned policy is not the exploration policy, which is achieved in policy-gradients through *importance sampling* methods [GLY 89, MEU 01, SHE 01, UCH 04];

- bias: the parameterized policy can “embed” the guide – which serves as a bias
- so that the optimization process learns how to correct this bias when the decisions made are not the best ones.

Another related idea is to use heuristics either to facilitate the computation of a value function approximator, or to shape the reward (about reward shaping, see [NG 99]).

5.5. Conclusion

This chapter has presented gradient algorithms (policy gradients) used for solving MDPs. This approach is possible whether a model is available or not, and permits us to control the shape and size of the search space since a first step is to choose the shape of the parameterized policy to be optimized. We can also reduce the size of the search space by computing an approximation of the value function, but the quality of the resulting deterministic greedy policy is not guaranteed. Direct policy search approaches have here the advantage of working in a space of stochastic policies.

An important point in this chapter is to remind us that solving MDPs is an optimization problem which can be addressed through various techniques. Computing a value function (e.g. using dynamic programming) is the most common approach, but gradient ascent algorithms, linear programming (Chapter 4, section 4.3.3), evolutionary algorithms [SCH 94, MAR 07] or cross-entropy methods [SZI 06] may be more appropriate in certain situations.

Readers interested in putting policy-gradient algorithms into practice may, for example, refer to Chapter 15, section 15.3, of this book for an application to planning (with discrete variables), or to [PET 03] for applications to robotics (with continuous variables).

5.6. Bibliography

- [ABE 02] ABERDEEN D. and BAXTER J., “Scaling internal-state policy-gradient methods for POMDPs”, *Proceedings of the 19th International Conference on Machine Learning (ICML'02)*, pp. 3–10, July 2002.
- [ABE 03] ABERDEEN D., Policy-gradient algorithms for partially observable Markov decision processes, PhD thesis, The Australian National University, Canberra, Australia, March 2003.
- [AMA 98] AMARI S., “Natural gradient works efficiently in learning”, *Neural Computation*, vol. 10, no. 2, pp. 251–276, 1998.
- [BAG 03] BAGNELL J. and SCHNEIDER J., “Covariant policy search”, *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*, Acapulco, Mexico, pp. 1019–1024, 2003.

- [BAI 93] BAIRD L., Advantage updating, Report no. WL-TR-93-1146, Wright-Patterson Air Force Base Ohio: Wright Laboratory, 1993.
- [BAI 99a] BAIRD L., Reinforcement learning through gradient descent, PhD thesis, Carnegie Mellon University, Pittsburgh, PA 15213, 1999.
- [BAI 99b] BAIRD L. and MOORE A., “Gradient descent for general reinforcement learning”, *Advances in Neural Information Processing Systems 11 (NIPS'98)*, MIT Press, Cambridge, MA, pp. 968–974, 1999.
- [BAX 01a] BAXTER J. and BARTLETT P., “Infinite-horizon policy-gradient estimation”, *Journal of Artificial Intelligence Research*, vol. 15, pp. 319–350, 2001.
- [BAX 01b] BAXTER J., BARTLETT P. and WEAVER L., “Experiments with infinite-horizon, policy-gradient estimation”, *Journal of Artificial Intelligence Research*, vol. 15, pp. 351–381, 2001.
- [BHA 09] BHATNAGAR S., SUTTON R. S., GHAVAMZADEH M. and LEE M., Incremental natural actor-critic algorithms, Report no. TR09-10, University of Alberta, Department of Computing Science, 2009.
- [COQ 09] COQUELIN P.-A., DEGUEST R. and MUNOS R., “Particle filter-based policy gradient in POMDPs”, *Advances in Neural Information Processing Systems 21 (NIPS'08)*, pp. 337–344, 2009.
- [GLY 89] GLYNN P. and IGLEHART D., “Importance sampling for stochastic simulations”, *Management Science*, vol. 35, no. 11, pp. 1367–1392, 1989.
- [GRE 02] GREENSMITH E., BARTLETT P. and BAXTER J., “Variance reduction techniques for gradient estimates in reinforcement learning”, DIETTERICH T. G., BECKER S. and GHAHRAMANI Z., Eds., *Advances in Neural Information Processing Systems 14 (NIPS'01)*, MIT Press, Cambridge, MA, 2002.
- [KAK 02] KAKADE S., “A natural policy gradient”, *Advances in Neural Information Processing Systems 14 (NIPS'01)*, pp. 1531–1538, 2002.
- [KIM 98] KIMURA H. and KOBAYASHI S., “Reinforcement learning for continuous action using stochastic gradient ascent”, *Proceedings of the 5th International Conference on Intelligent Autonomous Systems (IAS'98)*, pp. 288–295, 1998.
- [KON 03] KONDA V. and TSITSIKLIS J., “On actor-critic algorithms”, *SIAM Journal on Control and Optimization*, vol. 42, no. 4, pp. 1143–1166, 2003.
- [MAC 03] MACKAY D., *Information Theory, Inference, and Learning Algorithms*, Cambridge University Press, New York, 2003.
- [MAR 07] DE MARGERIE E., MOURET J.-B., DONCIEUX S. and MEYER J.-A., “Artificial evolution of the morphology and kinematics in a flapping-wing mini-UAV”, *Bioinspir. Biomim.*, vol. 2, pp. 65–82, 2007.
- [MEU 99] MEULEAU N., PESHKIN L., KIM K.-E. and KAELBLING L., “Learning finite-state controllers for partially observable environments”, *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI'99)*, Morgan Kaufmann, San Mateo, CA, pp. 427–436, 1999.

- [MEU 01] MEULEAU N., PESHKIN L. and KIM K., Exploration in gradient-based reinforcement learning, Report no. AIM-2001-003, MIT Artificial Intelligence Laboratory, 2001.
- [MUN 06] MUNOS R., “Geometric variance reduction in Markov chains: application to value function and gradient estimation”, *Journal of Machine Learning Research*, vol. 7, pp. 413–427, 2006.
- [NG 99] NG A., HARADA D. and RUSSELL S., “Policy invariance under reward transformations: theory and application to reward shaping”, *Proceedings of the 16th International Conference on Machine Learning (ICML'99)*, Morgan Kaufmann, San Francisco, CA, pp. 278–287, 1999.
- [PET 03] PETERS J., VIJAYAKUMAR S. and SCHAAAL S., Policy gradient methods for robot control, Report no. CS-03-787, University of Southern California, 2003.
- [PET 05] PETERS J., VIJAYAKUMAR S. and SCHAAAL S., “Natural actor-critic”, GAMA J., CAMACHO R., BRAZDIL P., JORGE A. and TORGÓ L., Eds., *Proceedings of the 16th European Conference on Machine Learning (ECML'05)*, vol. 3720 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 280–291, 2005.
- [SCH 94] SCHOENAUER M. and RONALD E., “Neuro-genetic truck backer-upper controller”, *Proceedings of the 1st International Conference on Evolutionary Computation (ICEC'94)*, Orlando, FL, pp. 720–723, 1994.
- [SCH 06] SCHRAUDOLPH N., YU J. and ABERDEEN D., “Fast online policy-gradient learning with SMD gain vector adaptation”, WEISS Y., SCHÖLKOPF B. and PLATT J. C., Eds., *Advances in Neural Information Processing Systems 18 (NIPS'05)*, MIT Press, Cambridge, MA, pp. 1185–1192, 2006.
- [SHE 01] SHELTON C., Importance sampling for reinforcement learning with multiple objectives, Report no. AIM-2001-003, MIT Artificial Intelligence Laboratory, 2001.
- [SUT 00] SUTTON R., MCALLESTER D., SINGH S. and MANSOUR Y., “Policy gradient methods for reinforcement learning with function approximation”, *Advances in Neural Information Processing Systems 12 (NIPS'99)*, Cambridge, MA, MIT Press, pp. 1057–1063, 2000.
- [SZI 06] SZITA I. and LÖRINCZ A., “Learning tetris using the noisy cross-entropy method”, *Neural Computation*, vol. 18, pp. 2936–2941, 2006.
- [UCH 04] UCHIBE E. and DOYA K., “Competitive-cooperative-concurrent reinforcement learning with importance sampling”, *From Animals to Animats 8: Proceedings of the Eighth International Conference on Simulation of Adaptive Behavior (SAB'04)*, pp. 287–296, 2004.
- [WIL 87] WILLIAMS R., “A class of gradient-estimating algorithms for reinforcement learning in neural networks”, *Proceedings of the 1st International Conference on Neural Networks (ICNN'87)*, San Diego, CA, pp. 601–608, 1987.
- [WIL 92] WILLIAMS R., “Simple statistical gradient-following algorithms for connectionist reinforcement learning”, *Machine Learning*, vol. 8, no. 3, pp. 229–256, 1992.

Chapter 6

Online Resolution Techniques

6.1. Introduction

We have seen in previous chapters how to approximately solve large MDPs using various techniques based on a parameterized or structured representation of policies and/or value functions, and on the use of simulation for reinforcement learning (RL) techniques. The optimization process returns an approximate optimal policy $\tilde{\pi}$ that is valid for the whole state space. For very large MDPs, obtaining a good approximation is often difficult, and is all the more difficult that, in general, we do not know how to precisely quantify the policy's sub-optimality *a priori*. A possible improvement then consists of considering these optimization methods as an offline pre-computation. During a second phase, online, the *a priori* policy is improved by a non-elementary computation for each encountered state.

6.1.1. Exploiting time online

In the framework of MDPs, the algorithm used to determine the current action online is generally very simple. For example, when $\tilde{\pi}$ is defined through a value function \tilde{V} , this algorithm is a simple comparison of the actions' values "at one step" (see Algorithm 1.3 in Chapter 1). Similarly, in the case of a parameterized policy, the algorithm determining the current action is usually very basic. This resolution scheme allows for a very reactive control and is well adapted to embedded systems that are under hard real-time constraints. On the contrary, it can be challenged to improve a policy online when there is enough time to make a decision.

Chapter written by Laurent PÉRET and Frédéric GARCIA.

This is a key to the success of the best game programs, which intensively combine offline and online computations. For example, the famous chess program Deep Blue [CAM 02] relies on a complex value function defined by more than 8,000 feature functions. The weights of the value function have been subject to numerous offline optimizations, both manually and automatically. The resulting value function provides an excellent evaluation of a position's strength, integrating the judgment of human experts. Yet, it remains synthetic and imperfect: the elementary algorithm consisting in choosing the move leading to the position with the best evaluation from the current position does not always lead to an efficient move. Deep Blue is a program able to win against the best human players because it couples the value function with a tree search algorithm that expands millions of positions obtained after several moves starting from each position encountered during the game.

6.1.2. *Online search by simulation*

For large MDPs, performing a tree search can be a delicate task, in particular when transition probabilities are unknown or when the number of successor states is large – in other words, in terms of tree search, when the branching factor is high. As for offline algorithms, the use of a simulation makes it possible to circumvent these difficulties by performing computations based on samples of simulated transitions. In this context, Kearns, Mansour and Ng have proposed an algorithm [KEA 02] laying the theoretical foundations to combine tree search and simulation online. This algorithm builds a tree by stochastic simulation over a *reasoning horizon* H and defines a stochastic policy. This approach is very attractive since it allows for treating arbitrarily large MDPs using just a simulator of the system.¹ The price to pay for this generality is the huge number of calls to the simulator required by this algorithm, which makes it ineffective in many cases.

An alternative to the tree search techniques to improve a policy online has been proposed by Tesauro and Galperin [TES 97]. It requires the existence of a simulator and an *a priori* policy $\tilde{\pi}$ – defined by a value function or parameterized. It can be seen as a focused iteration of approximate policy iteration. This algorithm sometimes proves to be computationally very intensive but, on the other hand, and contrary to Kearns *et al.*'s algorithm, it has proved itself experimentally, in particular by improving the game level of the TD-Gammon backgammon program [TES 02].

For large problems, it also appears necessary to control the search online so as to quickly improve the choice of the action to perform while additional simulations are allocated to its computation. This is the objective of the *focused reinforcement*

1. Integrating a value function \tilde{V} to evaluate the leaves of the tree constitutes a natural improvement of the algorithm.

learning and controlled rollout methods proposed in [PER 04a, PER 04b] and which we present below.

6.2. Online algorithms for solving an MDP

We formalize here the problem of the online search for MDPs: this problem can be envisioned as the local resolution of an MDP over a given *reasoning horizon*. We then list some online approaches that solve this problem. Compared to offline algorithms, the main idea is to focus computations around the current state.

6.2.1. Offline algorithms, online algorithms

Our use of the terms “online” and “offline” deserves some precision. To distinguish the classical resolution methods presented in previous chapters from the methods presented here, we could have used the terms “non-focused” and “focused”, or “global” and “local”. We have chosen “offline” and “online” because these terms reflect well the succession of two resolution phases that characterizes this approach.

We should, however, specify that numerous RL algorithms which we classify in the category of offline algorithms often claim to be online approaches [SUT 93, RUM 94]. These algorithms, although pertaining to a global resolution, only involve a single optimization phase then generally called learning phase. This single phase is qualified as online since it can be implemented directly in a real-world setting, for example, to control the actions of a robot that adapts to its environment [BAG 02]. Indeed, RL algorithms typically perform a sequence of elementary updates which are compatible with hard real-time constraints or with limited computational resources.

The purpose of an online phase is to perform a non-trivial computation for each encountered state. We are therefore interested in this chapter in algorithms that focus on the computation of an optimal action for a given current state.

6.2.2. Problem formalization

In this section, we formalize the local problem of determining the best action for a current state over a given *reasoning horizon* H . We present a simple forward search algorithm that examines all possible action sequences over horizon H . This basic, but expensive, algorithm allows us to quantify the gain acquired for a given reasoning horizon.

In the following sections we will develop algorithms allowing us to efficiently solve this local problem.

Note that, in the whole chapter, analogously to shortest-path problems, we try to minimize a sum of costs rather than maximize a sum of rewards.

6.2.2.1. Forward search over a reasoning horizon

The problem of determining the best action for the current state s_t can be formulated as the resolution of a *local MDP* restricted to the set $S_H(s_t) \subset S$ of states reachable from s_t in at most H transitions. This local MDP is associated with an AND/OR graph $G_H(s_t)$ such that:

- the states are OR *nodes* with $|A|$ outgoing edges; each edge is connected to an action node; a state node is associated with a minimization operator;
- actions are AND *nodes* with $|S(s, a)|$ outgoing edges; each edge is connected to a state node s' and valued with a transition probability $p(s' | s, a)$; each action node is associated with an expectation operator.

We assume that a value function \tilde{V} approximating V^* for each state s is available. This value function is used here to evaluate the states s_{t+H} located at the fringe of $S_H(s_t)$ (see Figure 6.1).

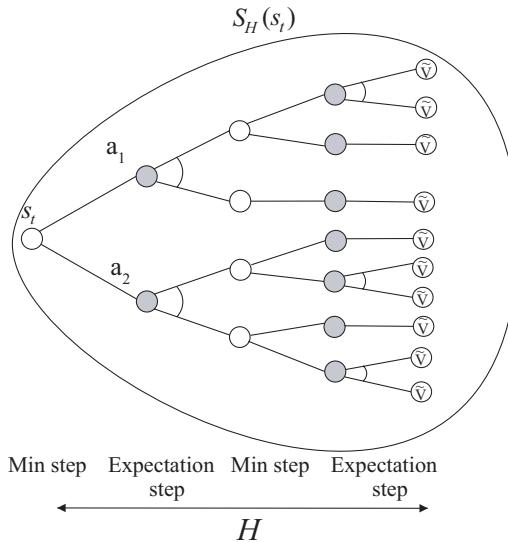


Figure 6.1. Forward search over a horizon H . White circles represent state nodes and gray circles represent action nodes

We assume that the approximation error over \tilde{V} is bounded by a real number ϵ :

$$\exists \epsilon > 0 \text{ such that } \|\tilde{V} - V^*\|_\infty \leq \epsilon. \quad (6.1)$$

The most direct way to solve the local MDP associated with s_t consists of performing an exhaustive forward search over the considered reasoning horizon H ,

i.e. a traversal of the previous AND/OR graph starting from state s_t . A new value function V_H is then calculated online. The set $S_H(s_t)$ is expanded recursively from s_t and V_H is calculated according to Bellman's optimality principle with equation (6.2) in Algorithm 6.1.

Algorithm 6.1: Forward search algorithm over a reasoning horizon

Input: current state s_t, p, c , horizon H

$$V_H(s_t) = \begin{cases} \tilde{V}(s_t) & \text{if } H = 0 \\ \min_{a \in A} Q_H(s_t, a) & \text{otherwise} \end{cases}$$

where $Q_H(s_t, a) = c(s_t, a) + \gamma \sum_{s' \in S(s_t, a)} p(s' | s_t, a) V_{H-1}(s')$

(6.2)

return $a_t = \operatorname{argmin}_{a \in A} Q_H(s_t, a)$

This is a recursive algorithm computing $V_H(s_t)$ as a function of $V_{H-1}(s')$ for all states s' reachable from s_t . During the forward search through reachable states, this algorithm requires repeating the same computations twice.

6.2.2.2. Tree or graph?

As for classical graph search algorithms, it is possible to simplify the Markovian graph as a *tree*. In this case, when a new state is expanded, it is simply linked to the state-action pair it is coming from, without testing for its existence.

This simplification – commonly employed in game programs – has the advantage of velocity. The counterpart is an increased cost in terms of memory when states are duplicated. The tree representation is preferred when the risk of encountering duplicate states is low.

We will see that both representations are used in the setting of MDPs: the heuristic online approaches described in section 6.2.3, such as the *LAO** algorithm, use a graph representation while Kearns *et al.*'s algorithm, described in section 6.2.4, relies on a tree representation.

6.2.2.3. Complexity and efficiency of the forward search

The complexity of forward search Algorithm 6.1, exponential in H , is in $O((S_{\max}|A|)^H)$, where $S_{\max} = \max_{s \in S_H(s_t)} \max_{a \in A} |S(s, a)|$ is the maximum number of successor states for a state-action pair.

As for the value iteration algorithm, the following proposition can be proved.

PROPOSITION 6.1 [HER 90]. *Error for the forward search:*

$$\|V_H - V^*\|_\infty \leq \gamma^H \epsilon.$$

This proposition states that V_H converges geometrically towards V^* . From this, if the reasoning horizon is long enough, the error due to \tilde{V} 's imprecision will be significantly reduced.

In the case of a stochastic shortest path problem with $\gamma = 1$, it is necessary for $\epsilon = \epsilon_H$ to decrease when H increases in order for the forward search to be beneficial. In other words, the error must be smaller for leaf states than for the root state. As for two-player games, this property holds if the approximate value function satisfies a property of “increased visibility”: \tilde{V} 's precision increases when getting closer to a goal state.

This way, this simple algorithm justifies the online resolution of MDPs: expliciting the Markovian graph over a horizon H makes it possible to obtain an online value function improving \tilde{V} and, consequently, the online policy π_{tree} deduced from it.

In practice, this algorithm is unwieldy due to its exponential complexity and because it requires manipulating probability distributions p extensively. We will see in the next section how heuristic search algorithms for MDPs offer an alternative to this exhaustive resolution, even if they also manipulate probability distributions p extensively.

6.2.3. Online heuristic search algorithms for MDPs

In the middle of the 90s, some researchers, in particular from the planning community, have proposed various online algorithms to efficiently exploit the information brought by the current state, the main ones being RTDP (Real-Time Dynamic Programming) [BAR 95], the envelope algorithm [DEA 95], and more recently LAO* [HAN 01].

6.2.3.1. General principles

These algorithms progressively expand a sub-set of states containing the states that can be reached from the current state without specifying the reasoning horizon. This sub-set called envelope in [DEA 95] initially contains only the current state. The states located on the fringe of the envelope can be evaluated with an offline pre-calculated value function \tilde{V} . The general working scheme of these algorithms is an alternation of expansion phases and update phases:

1) expansion phase: a state s is chosen at the fringe of the envelope and its new value is calculated:

$$V(s) \leftarrow \min_{a \in A} \left[c(s, a) + \gamma \cdot \sum_{s' \in S} p(s' | s, a) \cdot \tilde{V}(s') \right]$$

(assuming that reachable states s' are met for the first time);

2) parent states update phase: the value of parent states of the newly expanded state is updated using Bellman's optimality principle.

We should, however, note that the problem is not exactly that of improving an *a priori* policy for a large MDP, and that most of them do not rely on simulations. Their objective is rather to offer an efficient alternative to algorithms such as value iteration for reasonably complex MDPs by working on a sub-set of the state space – typically navigation problems such as stochastic shortest-paths. Thus, they make it possible to calculate an optimal action for the current state much faster than an offline algorithm that converges only when an optimal policy over all S has been calculated.

This general process that progressively expands the Markovian graph associated with the MDP from the current state also appears in the RTDP – proposed by Barto, Bradtke and Singh [BAR 95] – and *LAO** – proposed by Hansen and Zilberstein [HAN 01] – algorithms presented below.

6.2.3.2. The RTDP algorithm

For example, RTDP (Real-Time Dynamic Programming) [BAR 95] performs an expansion phase through successive trajectories generated from the current state down to a goal state. These trajectories are determined by choosing for each encountered state s the action a with the best value (current greedy policy). The state s' following s is defined as the successor of s via the simulated stochastic transition (s, a, s') . The value of a state is updated each time it is encountered. We obtain an asynchronous dynamic programming algorithm “directed” by the controlled process. This algorithm converges under certain conditions, for example, if:

- the initial value function \tilde{V} underestimates V^* – i.e. $\tilde{V} \leq V^*$ – (\tilde{V} is said to be an admissible heuristic),
- all costs are non-negative.

Algorithm 6.2 describes RTDP more formally in the case of an MDP with a unique initial state s_0 .

RTDP’s expansion strategy tends to favor most probable trajectories, makes it possible to quickly obtain a good quality policy and only refines the exploration on states reachable under an optimal policy. The counterpart is a slow convergence, unlikely states being rarely encountered (see [BON 03a]).

Algorithm 6.2: RTDP algorithm

Input: A, p, c , initial state s_0 , value function \tilde{V}

while non-satisfied stopping condition **do**

$$\begin{aligned}
 & s \leftarrow s_0 \\
 & \textbf{repeat} \\
 & | \quad S' \leftarrow \text{reachableStates}(s) \\
 & | \quad \textbf{for } s' \in S' \textbf{ do} \\
 & | | \quad \textbf{if } s' \text{ encountered for the first time then} \\
 & | | | \quad /* \text{ Initialize } V(s'). */ \\
 & | | | \quad V(s') \leftarrow \tilde{V}(s') \\
 & | | \quad \textbf{for } a \in A(s) \textbf{ do} \\
 & | | | \quad Q(s, a) \leftarrow c(s, a) + \gamma \cdot \sum_{s' \in S} p(s' | s, a)V(s') \\
 & | | | \quad a \leftarrow \text{argmin}_{a \in A(s)} Q(s, a) \\
 & | | | \quad V(s) \leftarrow Q(s, a) \\
 & | | | \quad s \leftarrow \text{sample}(p(\cdot | s, a)) \\
 & | \quad \textbf{until } s \text{ terminal state} \\
 & \textbf{return} \text{ value function } V
 \end{aligned}$$

Labeled RTDP

Labeled RTDP (LRTDP) [BON 03a] is a variant of RTDP where states whose value has converged are identified thanks to a label. This procedure makes it possible to better control of the algorithm's convergence and even to guarantee that it terminates in finite time.

This general approach that progressively expands the Markovian graph associated with the MDP from the current state has inspired later works on solving MDPs online and we present below one of these online algorithms in details, the *LAO** algorithm proposed by Hansen and Zilberstein [HAN 01].

6.2.3.3. *The LAO** algorithm

We have seen that an MDP can be seen as an “AND/OR” graph with two types of nodes: minimization nodes and expectation nodes. The *AO** algorithm [PEA 84] is a heuristic search algorithm commonly used for this kind of problems when the graph at hand is acyclic. Hansen and Zilberstein’s *LAO** algorithm generalizes *AO** to MDPs which are graphs containing cycles or loops,² *AO** being itself an extension of *A** [NIL 80] for shortest-path problems in a graph. The algorithm progressively expands

2. The ‘L’ in *LAO** stands for *loop*.

the Markovian graph – initially reduced to the current state s_t . The original version of the algorithm has been developed in the context of stochastic shortest path problems, assuming the existence of goal states.

Analogously with heuristic search algorithms, Hansen and Zilberstein distinguish three graphs:

- the implicit graph G associated with the whole MDP;
- the explicit graph $G' \subset G$: it contains all states expanded from current state s_t ;
- the solution graph $\tilde{G} \subset G'$: this is the restriction of the explicit graph to the current best *partial policy*: in s_t , one picks the action \tilde{a} whose current evaluation is the best; one then selects the successor states of s_t obtained by executing \tilde{a} ; \tilde{G} is obtained by recursively iterating this process on successor states.

Non-expanded states are those located at the fringe of the explicit graph and are evaluated by \tilde{V} . If s is a goal state, then $\tilde{V}(s) = 0$.

We give here (Algorithm 6.3, Figures 6.2, 6.3 and 6.4) the version of *LAO** based on policy iteration.³

Algorithm 6.3: LAO* algorithm

```

Input:  $A, p, c$ , current state  $s_t$ , value function  $\tilde{V}$ 
/*  $G'$  is the explicit graph */  

 $G' \leftarrow \{s_t\}$   

while  $G'$  contains a non-expanded state that is not a goal state do  

    | (1) Expansion of the solution graph  $\tilde{G}$   

    |   (a) Choose  $s$  in the solution graph that is not a goal state  

    |   (b)  $V(s) \leftarrow \min_{a \in A} [c(s, a) + \sum_{s' \in S(s, a)} p(s'|s, a) \cdot \hat{V}(s')]$ ,  

    |       where  $\hat{V}(s') = \begin{cases} \tilde{V}(s') & \text{if } s' \text{ is encountered for the first time,} \\ V(s') & \text{otherwise.} \end{cases}$   

    |   (c)  $G' \leftarrow G' \cup_{a \in A(s)} S(s, a)$   

    | (2) Update the actions' values and the solution graph  $\tilde{G}$   

    |   - Determine the set  $Z$  containing  $s$  and all its ancestors in the  

    |     explicit graph  $G'$  according to the vertices corresponding to the current  

    |     best actions, i.e. only the ancestors allowing to reach  $s$  with the current best  

    |     policy.  

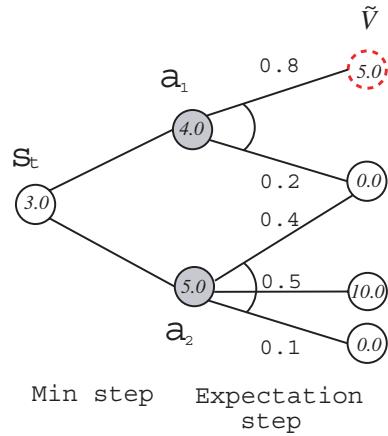
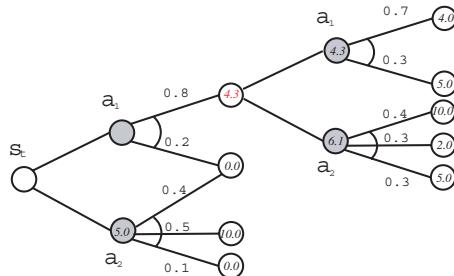
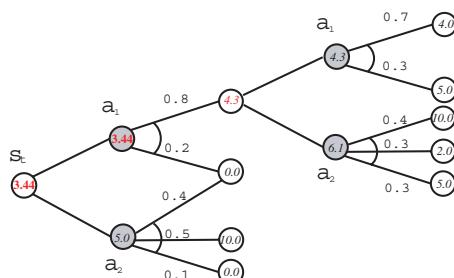
    |   - Execute policy iteration Algorithm 1.5 on set  $Z$  and determine the  

    |     new current best actions.  

return solution graph  $\tilde{G}$ 

```

3. A similar version based on value iteration has been developed.

**Figure 6.2.** LAO* algorithm - step (1 a): choice of a node to expand**Figure 6.3.** LAO* algorithm - steps (1 b) and (1 c): expansion phase**Figure 6.4.** LAO* algorithm - step (2): cost revision phase

*Convergence and efficiency of LAO**

If \tilde{V} is *admissible* – i.e. if $\tilde{V} \leq V^*$, then:

- $V(s) \leq V^*(s)$ for each state s expanded when step (2) is achieved;
- $V(s) = V^*(s)$ when LAO^* terminates;
- LAO^* terminates after a finite number of iterations.

As in related algorithms such as RTDP and the envelope algorithm, LAO^* turns out to be particularly efficient when the optimal policy only visits a small fraction of the state space [HAN 01]. The computation of an optimal policy from the current state is then much faster than with value iteration or policy iteration.

One of the major advantages of LAO^* and related algorithms is their anytime profile: they produce a possibly sub-optimal partial policy but which gets quickly close to the optimal policy as additional states are expanded [BON 03a]. Moreover, Thiébaux *et al.* [THI 02] rightfully observe that they can be used offline as well as online.

If these algorithms allow for an efficient computation of a partial policy, they are not directly applicable to large MDPs. Thus, Dean *et al.* [DEA 95] assume that the number of successor states of a state-action pair remains reasonable for the MDP treated by their algorithm. These results can be improved by using domain-specific admissible heuristics to calculate \tilde{V} . For example, GPT – a probabilistic planner based on LRTDP – has solved in a few hours POMDPs with about 900,000 encountered states [BON 03b] (see also section 15.2.3). On the other hand, when \tilde{V} comes from an algorithm like approximate policy iteration, it generally does not satisfy desired properties for heuristic functions like admissibility. We will now see how simulations can be used online to overcome these limitations.

6.2.4. Kearns, Mansour and Ng’s simulation-based algorithm

The previous approaches can be exploited only if the probability distributions $p(s' | s, a)$ are known and if the number of successor states for a state-action pair remains reasonable. As for offline methods, the simulation-based approach allows us to overcome these difficulties. Thus, for each state-action pair (s, a) , the probability distribution over successor states is implicitly⁴ approximated by generating a sample of these states.

A simple algorithm, proposed by Kearns *et al.* [KEA 02], consists of trying each action N times from each state: N times from the current state s_t , and recursively

4. As for “direct” methods like Q -learning, transition probabilities are not explicitly estimated.

N times from each state generated from s_t over a reasoning horizon H (see Algorithm 6.4 and Figure 6.5). A new value function is thus defined online by the parameters N and H .

Algorithm 6.4: Kearns, Ng and Mansour's simulation-based algorithm

Input: current state s_t , Markovian system simulator, reasoning horizon H , width N

$$V_{H,N}(s_t) = \begin{cases} 0 & \text{if } H = 0, \\ \min_{a \in A} Q_{H,N}(s_t, a) & \text{otherwise,} \end{cases}$$

where $Q_{H,N}(s_t, a) = \left[c(s_t, a) + \gamma \frac{1}{N} \sum_{s' \in S(s_t, a, N)} V_{H-1,N}(s') \right]$

(6.3)

where $S(s, a, N)$ is the set of N states sampled from the pair (s, a)

return $a_t = \operatorname{argmin}_{a \in A} Q_H(s_t, a)$

Let us note that, when $N \rightarrow +\infty$, we obtain Algorithm 6.1.

Once this tree has been developed, the current action a_t is directly deduced by applying the argmin to the actions' values at the root of the tree. This algorithm thus defines a stochastic policy π_{tree} .

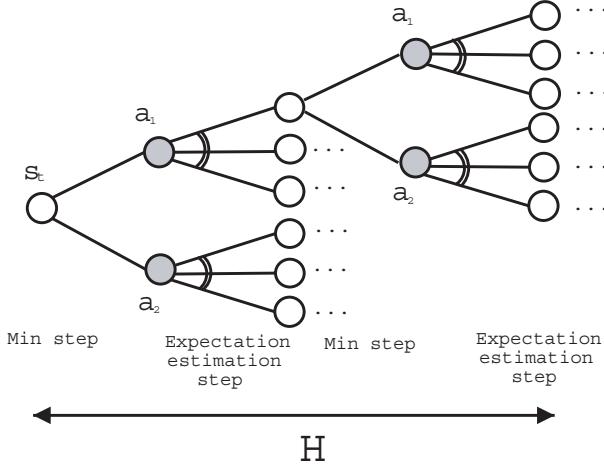


Figure 6.5. Kearns, Mansour et al.'s algorithm with two actions and $N = 3$

6.2.4.1. Complexity and convergence of Kearns et al.'s algorithm

The online complexity of $V_{H,N}(s_t)$'s computation is in $O((|A| \cdot N)^H)$. The theoretical result stated by Kearns *et al.* quantifies the difference $|V_{\pi_{\text{tree}}}(s_t) - V^*(s_t)|$ as a function of N and H : this difference can be made arbitrarily small for values of N and H which are large enough and *independent of the size of the state space*.

In other words, the online search by stochastic simulation makes it possible, in theory, to approximate an optimal policy as precisely as wanted using just a Markovian simulator.

Unfortunately, the values of N and H required in order to obtain a convergence guarantee are extremely large and unusable in practice given the $O((|A| \cdot N)^H)$ complexity of the algorithm.

6.2.4.2. Efficiency and practical considerations

Kearns *et al.* mention some ideas that could improve the efficiency of their algorithm:

- a first improvement consists of using a value function \tilde{V} to evaluate the leaves of the tree;
- a second improvement consists of privileging nodes close to the root by allocating them more simulations than to nodes close to the leaves: the width N then varies with the level h ($1 \leq h \leq H$) of the tree at hand;
- as for classical tree search algorithms, pruning techniques can be envisioned [RUS 95].

Recently, Chang *et al.* [CHA 05] have proposed another improvement of Kearns *et al.*'s algorithm. This is a two-phase variant, based on the theory of multi-armed bandits [BER 85]. After an initial estimation phase, additional simulations are distributed depending on the initial estimate of the mean and variance of the action and state values. The algorithm complexity still remains in $O((|A| \cdot N)^H)$.

Kearns *et al.*'s result proves the theoretical soundness of the simulation-based online search. It establishes that, for all MDP, a policy arbitrarily close to the optimal policy can be deduced by constructing a stochastic tree of width N over a reasoning horizon H . However, the theoretical algorithm is not applicable because of the very large values required for N and H , and because of its exponential complexity in H . We will see in the next section how to efficiently expand a search tree with reasonable computational resources.

6.2.5. Tesauro and Galperin's rollout algorithm

The *rollout* algorithm proposed by Tesauro and Galperin [TES 97] makes it possible to improve any policy $\tilde{\pi}$ using a simulator online.

The principle of the algorithm is simple: it consists of estimating through simulations over N trajectories the values $Q^{\tilde{\pi}}(s_t, a)$, i.e. the expected value of the action a followed by the application of policy $\tilde{\pi}$ over a horizon of H steps. The action a_t with the best value is then chosen and executed online. As for Kearns *et al.*'s algorithm, a stochastic policy π_{RO} is thus defined based on $\tilde{\pi}$.

Algorithm 6.5: *Rollout algorithm*

Input: current state s_t , Markovian system simulator, policy to improve $\tilde{\pi}$, reasoning horizon H , width N

$$\forall a \in A \quad Q_{H,N}^{\tilde{\pi}}(s_t, a) \leftarrow c(s_t, a) + \frac{1}{N} \sum_{i=1}^N \sum_{l=1}^{H-1} \gamma^l c_i^l \quad (6.4)$$

where the costs c_i^l are generated by applying $\tilde{\pi}$ after the successor states obtained when executing a in s_t

return $a_t = \operatorname{argmin}_{a \in A} Q^{\tilde{\pi}}(s_t, a)$

The rollout algorithm can be envisioned as a step focused on s_t of approximate policy iteration.

6.2.5.1. Complexity and convergence of the rollout algorithm

The algorithm complexity, linear in N and H , is in $O(|A|N \cdot H)$.

Chang [CHA 02] shows that a sufficient number of trajectories N guarantees that $V_{\pi_{RO}} \leq V^{\tilde{\pi}}$ with a high probability. Even if this number can be very large, the linear complexity of the algorithm makes it viable in most applications.

6.2.5.2. Efficiency of the rollout algorithm

Let us observe that the rollout algorithm only modifies the policy $\tilde{\pi}$ on the first of the H online simulated transitions. For complex problems, it is then of course preferable that the base policy $\tilde{\pi}$ be of good quality so that the policy π_{RO} may also give good results.

To restrict the number of simulations, Tesauro and Galperin suggest to continuously control the allocation of simulations by eliminating actions whose estimated values are far enough from the value of the current best action. By maintaining a confidence interval for each action, probably suboptimal actions can be pruned – the probability being specified by the user when defining the confidence interval.

Bertsekas [BER 97] proposes another interesting improvement consisting in estimating the differences between actions rather than the action values themselves. Chang [CHA 02] proposes a version called *parallel rollout*, which relies on multiple policies to define an online policy. Under certain conditions, this policy is not worse than the best one of these policies for each state encountered online.

Contrary to Kearns *et al.*'s algorithm, the rollout algorithm has been used successfully on various large MDPs: scheduling of lines of source code for a compiler simulator [MCG 02], improving the game level of the backgammon program TD-Gammon – although it has already been compared with the best human players – [TES 97, TES 02].

Tesauro and Galperin's rollout algorithm therefore offers a simple alternative to the tree search to improve online an *a priori* policy based on simulated trajectories. This approach is very generic but requires a good *a priori* policy. Moreover, the rollout algorithm often proves to be computationally very expensive. The next section describes a strategy optimizing the allocation of simulations for the rollout algorithm.

Let us note finally that some other online approaches have been developed, but in the framework of deterministic MDPs. For example, the TD-Leaf algorithm combines the TD(λ) principle with a tree search designed for the chess game [BAX 98]. Let us also mention the work by Davies *et al.* [DAV 98] who develop in a deterministic setting an online approach relying on a search algorithm similar to A^* .

6.3. Controlling the search

The simulation-based online approaches we have just described prove to be time-consuming in practice. In particular, the computation time for evaluating a policy improved online are generally very long. Indeed, gathering reliable statistics about a policy requires simulating hundreds of trajectories, which corresponds typically to tens of thousands of transitions. When, for each simulated transition, the computation of the action is non-trivial, the computation time of a policy can then be of the order of a day or a week.

In simulation-based online approaches, the control of the search consists of determining an expansion order for the nodes of the Markovian graph, the objective being to quickly select a good quality action for the current state. Compared to the heuristic search algorithms for MDPs presented above, the problem of controlling the search is much more complex because of the use of simulations. Indeed, each already expanded state can be chosen once again in order to improve the precision of the estimate of its value. Because of this, new simulations can be allocated not only to states at the fringe of the graph but also to any already expanded state.

In this section, we first present an analysis of the problem of controlling the simulation-based search in the case of Kearns *et al.*'s algorithm where the tree is expanded uniformly over a horizon H and in the case where the search is performed over a horizon of 1. This analysis then makes it possible to naturally introduce the two algorithms *focused reinforcement learning* and *controlled rollout* proposed in [PER 04a].

6.3.1. Error bounds and pathology of the forward search

The practical use of Kearns *et al.*'s algorithm makes it necessary to choose reasonable values for the horizon H and the width N , sacrificing optimality guarantees. Then comes the question of the principles that should lead the choice of H and N to obtain good performances while keeping a reasonable computing budget – let us recall that Kearns *et al.*'s algorithm is of complexity exponential in H .

Let \tilde{V} be a value function verifying equation (6.1) that will be used to evaluate the leaves of the tree. From Kearns *et al.*'s result we can derive a *probably approximately correct* bound on $|V_{H,N}(s_t) - V^*(s_t)|$ that links the probability Δ to make an error to the amplitude of this error:

PROPOSITION 6.2 [PER 04a]. *Error for Kearns et al.'s algorithm:*

$$|V_{H,N}(s_t) - V^*(s_t)| \leq \frac{c_{\max}}{(1-\gamma)^2} \sqrt{\frac{1}{N} \cdot \log \frac{(|A| \cdot N)^H}{\Delta}} + \gamma^H \epsilon$$

with a probability smaller than $1 - \Delta$, where $c_{\max} = \max_{(s,a) \in S \times A} |c(s,a)|$ is the maximum instant cost.

For a fixed horizon H , if $N \rightarrow \infty$, the first error term tends to 0, and we obtain the result established by Hernandez and Lasserre [HER 90] for Algorithm 6.1 in Proposition 6.1 and an error in $\gamma^H \epsilon$.

6.3.1.1. Pathology of the simulation-based forward search

A natural tree expansion strategy consists of progressively increasing the length of the reasoning horizon H as far as the computational resources allow it. This is the principle of *iterative deepening* approaches [KOR 85] often employed in game programs to efficiently exploit time online: the reasoning horizon is incremented progressively as far as there is available time.

Rather surprisingly, increasing the horizon H with a fixed width N leads after some point to an *increasing* error on $V_{H,N}$ for Kearns *et al.*'s algorithm: the first error term linked to the sampling-based approximation indeed increases with H . For a given width N , there exists an optimal value H^* beyond which the error increases, deteriorating the quality of the chosen action a_t (see Figure 6.6).

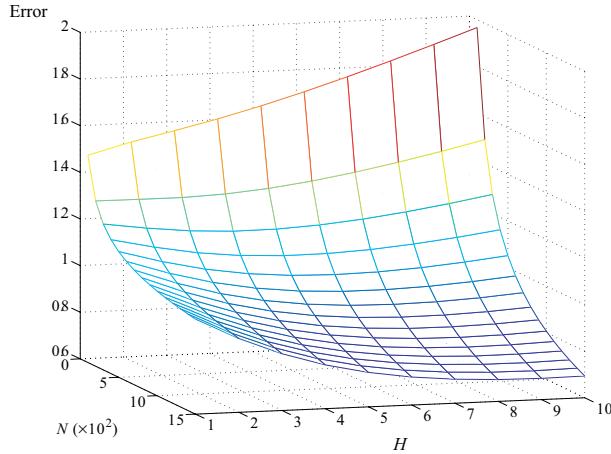


Figure 6.6. Error bound $|V_{H,C}(s_t) - V^*(s_t)|$ as a function of the horizon H and of the width $N - \Delta = 0.1$, $\epsilon = 1.0$, $c_{\max} = 0.1$, $|A| = 2$, $\gamma = 0.8$

This phenomenon is known under the name of *search pathology*, in the field of two-player games, for minimax-based tree search algorithms [KAI 90]. The selected move is that which guarantees the best position against any defense of the opponent, the strength of a position being estimated by a heuristic value function. This scheme is repeated over a certain reasoning horizon, the value function then estimating the leaf nodes of the expanded tree. Although it is generally accepted that a deeper search – that is, over a longer reasoning horizon – improves the quality of the selected move, several theoretical results [BEA 82, BRA 83, NAU 83, PEA 83] have highlighted, on the contrary, that a deeper search may *deteriorate* the quality of the selected move. For example, in the simple model developed by Pearl [PEA 83], a value function predicts whether a position will be winning with some error rate. This function satisfies an “increased visibility” property that decreases the error rate when searching deeper. Pearl then shows that, even with a high increase of the visibility, the quality of the decision taken at the root of the tree quickly deteriorates when the depth of the tree increases. The main reasons explaining this search pathology in these two-player game models are the independence of the heuristic values and a large branching factor.

In practice, this phenomenon has never been observed for classical games like chess. Various features of real games have been put forward to explain this divergence between theory and practice [KAI 88]: the existence of “trap” states corresponding to terminal positions, the dependence of the heuristic values at the leaves of the tree.

More recently, Bulitko *et al.* [BUL 03] have also demonstrated the existence of pathologies in the single-agent case. The classical problem being considered – comparable to a deterministic MDP – is that of searching for a shortest path. Bulitko *et al.* prove that the search pathology can occur even if the heuristic value function

is admissible. Their proof remains, however, restricted to very particular elementary problems for which the heuristic values at the leaves of the tree are very close. On the other hand, Péret and Garcia [PER 04b] have shown the search pathology for a medium-sized stochastic shortest-path MDP.

The search pathology thus demonstrated in the framework of MDPs is due to the existence of a sampling error at each node of the tree. In the case of two-player games, it is inherent to the properties of the heuristic function being used. The nature of the approximation error is thus different but, in both cases, it is amplified by a deeper search.

6.3.1.2. *Searching for a good compromise between depth and width*

The search pathology tells us that an “iterative deepening” search with a fixed width is not reliable since, beyond some horizon, the new expanded nodes will deteriorate the quality of action a_t . The error bound established by Proposition 6.2 can also be interpreted in terms of a compromise between bias and variance. The bias comes from the error on \tilde{V} and is mitigated by γ^H , whereas the variance comes from the first term and decreases according to $\sqrt{\log N/N}$.

Reasoning with the number of simulations, this compromise can be written in terms of choice for the length of the reasoning horizon H and the width N . Figure 6.7 draws the error bound given by Proposition 6.2 as a function of H for various computing budgets. The computing budget corresponds to the number of simulations allocated to expand the tree for given H and N and has the value

$$\sum_{h=1}^H (|A|N)^h = \frac{(|A|N)^{H+1} - |A|N}{|A|N - 1} \simeq (|A|N)^H.$$

This curves show how the optimal horizon depends on the allocated budget.

Knowing the available online budget, the values of H and N can be specified in advance. However, in practice, as it is the case for the focused reinforcement learning and controlled rollout algorithms, it can be more judicious to define a local width for each state-action pair and to progressively increase the horizon and the widths while preserving a good compromise to avoid pathological phenomena.

6.3.2. *Iterative allocation of simulations*

We have just seen that expansion requires us to establish a good global compromise between depth and width. We will now focus on the local problem of allocating simulations between various actions for a given state-action pair.

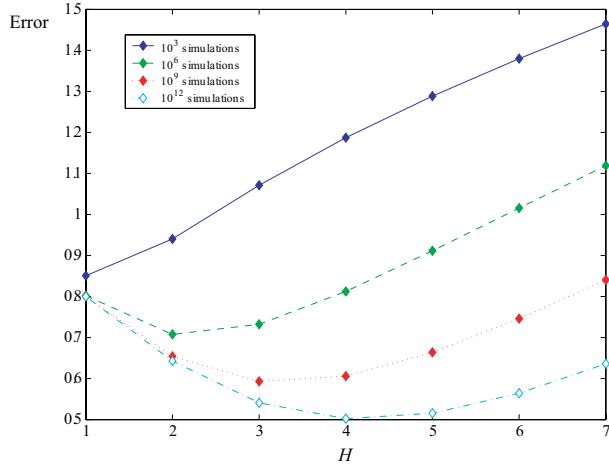


Figure 6.7. Error bound $|V_{H,C}(s_t) - V^*(s_t)|$ as a function of the horizon H for various computing budgets. $\Delta = 0.1$, $\epsilon = 1.0$, $c_{\max} = 0.1$, $|A| = 3$, $\gamma = 0.8$

It is possible to improve the uniform allocation as defined by Kearns *et al.*'s algorithm with an iterative approach accounting for the information available in already expanded nodes – mean and variance, for example. The objective is to privilege the most promising regions during the expansion of the tree. This problem is related to the classical dilemma between exploration and exploitation, in particular as it appears in RL.

6.3.2.1. Multi-armed bandits and exploration in MDPs

We have seen previously that RL algorithms only consider a single optimization phase, usually called online. To converge, these various algorithms require that each state-action pair be executed infinitely often during this phase.

With finite time, one has both to explore sufficiently to guarantee the convergence and exploit the policy considered to be optimal to minimize the incurred costs. Exploration policies defining the choice of action to perform during the optimization have to face this exploration-exploitation dilemma.

The theory of multi-armed bandits [BER 85] gives a framework for solving this dilemma. A k -armed bandit is a gambling machine giving a choice among k arms, each trial requiring the introduction of a coin in the machine. Each arm delivers a random revenue; the revenues delivered by the various arms are assumed to be independent random variables whose expectation is unknown. The problem is then to quickly select the arm whose expected revenue is the largest. More precisely, we have to determine a policy specifying which arm to pull down next given the accumulated

statistics for all arms. Various optimization criteria can be defined to determine this policy. A classical criterion is the following:

$$\max E \left[\sum_{t=1}^{\mathcal{T}} \gamma^t r_t \right], \quad (6.5)$$

where r_t is the revenue received during trial t , \mathcal{T} is the number of trials and γ is a discount factor. Exploiting consists of choosing the arm whose current estimate is the best while exploring consists of choosing another arm which may be underestimated.

An exploration policy that is optimal with respect to a criterion such as expressed by equation (6.5) can be calculated under various hypotheses about the probability distributions over revenues and in various formal frameworks – Bayesian, non-Bayesian. The book by Berry and Fristedt [BER 85] compiles classical results on this subject.

Some works [KAE 93, MEU 99a] have studied the possibility of transposing the theory of multi-armed bandits to MDPs to design exploration policies that are efficient for RL. Chang *et al.* [CHA 05] have recently proposed to apply it to Kearns *et al.*'s algorithm (see section 6.2.4). The idea is to consider an MDP as a set of $|S| |A|$ -armed bandits, the delivered revenue being the long-term criterion for each state considered.

Meuleau and Bourgine [MEU 99a] show that certain hypotheses required to obtain an optimal exploration – independence of each arm for a given bandit, stationarity of each bandit and independence of each bandit – are not satisfied by the $|S|$ interdependent bandits associated with an MDP. Nevertheless efficient heuristic exploration policies are deduced from the framework of multi-armed bandits. These policies are based on the propagation of the uncertainty associated with each state-action pair.

However, the exploration problem for the online expansion of a tree is different from the problem usually encountered in RL. Indeed, only action a_t induces a real cost. This cost is incurred once the tree has been expanded and action a_t has been executed. The costs incurred during the expansion of the tree are just simulated and a criterion such as formulated by equation (6.5) is therefore not appropriate to define an online exploration policy.

Coming from the field of stochastic optimization, or simulation-based optimization [GOS 03], the framework of ordinal optimization proposes an optimization criterion more adapted to our online exploration problem. Ordinal optimization is defined by k candidates $\theta_1, \dots, \theta_k$, with $E[J(\theta_i, \omega)]$ the average performance of each candidate. The problem is then to find

$$\theta_{i^*} = \operatorname{argmin}_{\theta_i \in \Theta} E[J(\theta_i, \omega)].$$

The performance $J(\theta_i, \omega)$ results from a simulation and $E[J(\theta_i, \omega)]$ is typically estimated as the mean

$$\bar{J}_i = \frac{1}{N_i} \sum_{j=1}^{N_i} J(\theta_i, \omega_{ij})$$

over N_i outcomes for candidate θ_i . For a total number $N = N_1 + \dots + N_k$ of simulations allocated to the problem resolution, the best system is then estimated by $\hat{\theta}_i$, with

$$\hat{i} = \operatorname{argmin}_i \bar{J}_i.$$

A classically considered criterion to evaluate allocation rules for simulations is then defined by $\Pr(SC)$, the probability to *correctly select* the optimal system θ^* among the k candidates:

$$\Pr(SC) = \Pr(\theta_b = \theta_{i^*}).$$

The main result established to date for ordinal optimization is that it is possible to design simple control rules to distribute N over N_1, N_2, \dots, N_k , which guarantees that the convergence of $\Pr(SC)$ towards 1 is exponentially fast in N , even though the convergence of a sequence \bar{J}_i towards $J(\theta_i)$ is at most in $1/\sqrt{N}$. The principle of ordinal optimization thus relies on this result, namely that it is much easier to rank candidates than to precisely estimate their value. Various control rules have been proposed in the literature, from the uniform allocation rule $N_i = N/k$ to the optimal allocation rules proposed by Chen *et al.*, among which OCBA (Optimal Computing Budget Allocation) [CHE 00a, CHE 00b], which rely on an approximation of the criterion $\Pr(SC)$ in a Bayesian setting and under hypotheses of normality of the simulation outputs $J(\theta_i, \xi)$.

The exploration problem for the online expansion of a tree can thus be addressed as an ordinal exploration problem, a candidate being an action. In the particular case where the tree only consists of a unique state – in other words when one considers a tree of depth 1 – both problem types are equivalent. Numeric experiments conducted in [PER 04a] show in particular on some simple problems the superiority of the OCBA method over exploration methods coming from RL. We can also mention Wang *et al.*'s work, which propose a Bayesian method for action selection when expanding a tree. This method turns out to be experimentally very efficient on some simple problems.

6.3.3. Focused reinforcement learning

As for the heuristic search algorithms described in section 6.2.3, the general scheme of the *Focused Reinforcement Learning* algorithm (FRL) [PER 04b] is an alternation of tree expansion phases and update phases.

The control strategy consists of repeatedly following trajectories of length H from the current state s_t to a leaf state s_{t+H} . Contrary to Kearns *et al.*'s algorithm, the global width N is not specified anymore but depends on the considered state-action pair. Contrary to the Rollout algorithm, the exploration is not limited to the first move but to the whole horizon H , since trajectories are guided with a given exploration policy.

Moreover, in order to maintain a good global compromise between depth and width, the horizon H is controlled dynamically based on the estimate of the simulation error. This estimate is a heuristic indicating whether certain state-action pairs require more simulations. The idea is to increment H when this estimate of the error is stabilized.

6.3.3.1. Global sampling error estimation

We have seen that the forward search could be pathological because of the amplification of the error due to the simulation. In order to control the increase of H , FRL estimates this error with a decentralized heuristic approach. Such an approach consisting in propagating an error through a Markovian graph has been proposed by Meuleau and Bourgine [MEU 99b] so as to design exploration policies in RL.

For a given state-action pair, the sampling error can be estimated using a *confidence interval*. Confidence intervals are an elementary statistical tool commonly used to quantify uncertainty, in particular when Monte Carlo simulation methods are used. For each pair (s, a) of local width N , we thus define a *local error* $e(s, a)$ as

$$e(s, a) = \sigma \frac{t_{\theta/2}^{N-1}}{\sqrt{N}},$$

where $\sigma^2 = \frac{1}{N-1} \sum_{s' \in S(s, a, N)} (Q(s, a) - [c(s, a) + \gamma V(s')])^2$ is the empirical variance of $\gamma V(s')$, which is the estimate of the value of the successor state s' . $t_{\theta/2}^{N-1}$ is the Student function with $N - 1$ degrees of freedom for an asymptotic confidence level $\frac{\theta}{2}$ (e.g. $\theta = 0.05$).

This error can be interpreted as follows: if $V(s')$ is a stationary random variable normally distributed with expectation v , then

$$Q(s, a) \in [c(s, a) + \gamma v - e(s, a), c(s, a) + \gamma v + e(s, a)]$$

with probability $1 - \theta$.

The variable $V(s')$ is not assumed to be normal. Moreover, as $V(s')$ is an estimate, it is not stationary – its expectation evolves when additional nodes are developed. FRL uses, however, $e(s, a)$ to estimate this local error. Let us note that this expression only

quantifies the local sampling error and accounts neither for the error due to \tilde{V} nor for the sampling errors associated with the successor states of the pair (s, a) .

This error is then propagated through the tree in the same way as state and action values are propagated in equation (6.6). Thus, a *global* sampling error E is defined by

$$E_{H,C}(s_t) = \begin{cases} \sigma_{\text{init}} & \text{if } H = 0, \\ \min_{a \in A} F_H(s_t, a) & \text{otherwise,} \end{cases} \quad (6.6)$$

where $F_H(s, a) = \left[e(s_t, a) + \gamma \frac{1}{C} \sum_{s' \in S(s_t, a, C)} E_{H-1,C}(s') \right]$,

where σ_{init} is a constant estimating the error $\|V^* - \tilde{V}\|_\infty$.

6.3.3.2. Organizing the search in successive trajectories

The FRL algorithm follows successive trajectories of length H starting from s_t , where H is incremented progressively. The policy guiding these trajectories is some exploration policy, in accordance with the scheme adopted by RL methods. The choice of the next expanded or re-estimated state is thus determined by the exploration policy and the proper dynamics of the system.

Compared with classical RL algorithms, the FRL algorithm differs on three points:

- the Markovian graph is expanded progressively; the idea is to consider a sample of states of sufficient size but still reasonable since it is memorized extensively, i.e. with no parametric structure;⁵
- the trajectories have a length specified by a reasoning horizon, which is incremented progressively;
- the theoretically better-founded exploration policies are different from those usually considered.

As for the heuristic search algorithms described in section 6.2.3, the FRL algorithm alternates graph expansion phases and phases updating the state values. As suggested by the authors of the AO^* and LAO^* algorithms, it is generally more efficient to perform multiple expansions before making an update. Similarly, FRL performs H trajectories between two updates.

5. The value function \tilde{V} used at the fringe of the graph can of course be defined by a parametric structure. The objective of the online search is precisely to reduce the approximation induced by this parametric structure by “postponing” it H transitions away.

Algorithm 6.6: Focused reinforcement learning algorithm

Inputs: current state s_t , Markovian system simulator, exploration policy π_{exp} , tolerance δ , size of the sample of trajectories M

```

 $H \leftarrow 1$ 
 $OldE \leftarrow +\infty$ 
repeat
  while  $|E_H(s_t) - OldE| > \delta$  do
    GeneratedTrajectories( $s_t, \pi_{\text{exp}}, H, M$ )
     $OldE \leftarrow E_H(s_t)$ 
    UpdateStateValues()
     $H \leftarrow H + 1$ 
  until stopping condition
return  $a_t = \operatorname{argmin}_{a \in A} Q_H(s_t, a)$ 

```

The procedure $\text{GenerateTrajectories}(s_t, \pi_{\text{exp}}, H, M)$ generates M trajectories of length H by following the policy π_{exp} from s_t . The procedure $\text{UpdateStateValues}()$ updates the values and the errors of the developed states and their ancestors according to equations (6.3) and (6.6).

The exploratory policy π_{exp} can be chosen among ordinal optimization methods, which specify for each action a a ratio $0 \leq \rho_a \leq 1$ to distribute N simulations among the $|A|$ actions. A natural way to define an exploration policy then consists of considering the stochastic policy where each action a is chosen with a probability ρ_a . This process allows us to progressively distribute simulations according to the ratios ρ_a while new trajectories are generated. However, this exploration policy will not benefit from the same convergence guarantees established for ordinal optimization algorithms: for any MDP, the action values are not distributed normally. Moreover, they are not stationary, the estimated values evolving while additional simulations are allocated among the successor states of the considered state.

6.3.3.3. Convergence and practical considerations

The convergence of the FRL algorithm can be established for a *fixed* horizon H . The mechanism incrementing the horizon of FRL is heuristic, the idea being to have enough simulations for the current horizon before incrementing it. It is, however, possible to deduce from Kearns *et al.*'s result the number of trajectories necessary to obtain a given error bound on the value function with a given horizon H (this bound is necessarily bounded from below by $\gamma^H \epsilon$).

PROPOSITION 6.3 [PER 04a]. *Convergence of FRL with a fixed horizon H : let V_H^{frl} be the value function deduced from the graph developed by M trajectories of length H by*

following a stochastic exploration policy π_{exp} . If, for all pair (s, a) , $\pi_{\text{exp}}(s, a) > 0$, then

$$V_H^{\text{frl}}(s_t) \longrightarrow V_H(s_t) \text{ almost surely when } M \longrightarrow +\infty,$$

where V_H is the value function deduced from forward search Algorithm 6.1.

The representation used by Kearns *et al.*'s algorithm is a tree, no occurrence test being performed – see Figure 6.8 (a). If it allows us to quickly develop the tree, this simplification usually proves to be of little efficiency to estimate the transition probabilities because of redundancies. It is thus preferable to perform an occurrence test for the states stemming from a given state-action pair in a view to estimate the corresponding transition probabilities. In accordance with the principle generally used in RL, $p(s' | s, a)$ is estimated by $\tilde{p}(s' | s, a) = \frac{N_{s,a}^{s'}}{N_{s,a}}$, which is the ratio between the number of times when the state s' has been observed as resulting from the pair (s, a) and the number of times when the pair (s, a) has been executed. Each vertex coming from an action node associated with a is valued with $\tilde{p}(s' | s, a)$, the structure of the tree being then preserved – see Figure 6.8 (b). Occurrence tests can also be performed among all states coming from a given state s . The tree then becomes a graph, a state node possibly having multiple fathers – see Figure 6.8 (c). Following this approach to its end, occurrence tests can also be performed among all already developed states. It is advisable to choose the most appropriate representation depending on the characteristics of the application at hand: this choice has to be guided by the probability for a state to be duplicated, by the cost of an occurrence test and by the size of the state samples being manipulated online.

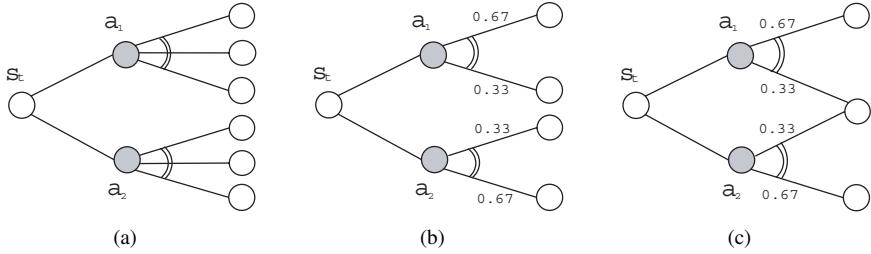


Figure 6.8. Several representations of the Markovian graph expanded online

As noted by Kearns *et al.*, the errors occurring in nodes located at a shallow depth of the tree have a stronger effect than those located deeper. As a consequence, they suggest that the distribution of the simulations decrease with the depth, for example, proportionally with γ^{2i} , where i , $1 \leq i \leq H$, is the depth of the node at hand. The trajectory-based strategy followed by FRL tends to distribute simulations proportionally with $\frac{1}{(\|A\| \|S(s, a)\|)^i}$. This tends to strongly favor the nodes closer to the

root. This phenomenon can be made up for by choosing as initial state of a trajectory a state located at depth $i > 1$. A pertinent way to choose such a state is to traverse a trajectory of length i from s_t : the role of the first i simulated transitions is then not to improve the precision of the estimate of nodes close to the root, but to select promising nodes located deeper. Finally, continuously following trajectories from s_t makes it possible to efficiently distribute simulations in the tree.

As we have seen, the value of a state of the graph is estimated by $V_n(s) = \min_{a \in A} Q_{N_{s,a}}(s, a)$, where $n = \sum_{a \in A} N_{s,a}$. When the values of n considered are low, the variance of the estimate is high. It can therefore be beneficial to introduce a weighting with the function \tilde{V} and to define an update rule including a learning rate α_n . The previous equation is then replaced by $V_n(s) = \alpha_n \cdot \tilde{V} + (1 - \alpha_n) \cdot \min_{a \in A} Q_{N_{s,a}}(s, a)$, where $0 < \alpha_n < 1$, decreases with n . The resulting update rule is similar to those defining Q-learning-like algorithms.

The FRL algorithm does not intend to improve the value function \tilde{V} for all states S through a parametric structure as RL algorithms usually do: the computations are performed with the only goal of selecting a good action for the current state s_t . It is, however, possible to reuse the already developed graph $G_H(s_t)$ to select a good action for s_{t+1} . After the real execution of transition s_t, a_t, s_{t+1} , if s_{t+1} belongs to the set $S_H(s_t)$ of the states explicitated in $G_H(s_t)$, then the graph $G_H(s_{t+1})$ is not initially reduced to s_{t+1} and contains all the successor states of s_{t+1} in G_t . This principle has been applied to various A^* -like search algorithms, in particular for real-time replanning problems – see, for example, the work of Koenig *et al.* [KOE 04a, KOE 04b].

6.3.4. Controlled rollout

We conclude this chapter by presenting an improved version of the rollout algorithm called *controlled rollout* [PER 04a].

6.3.4.1. Choice of the horizon

As for Kearns *et al.*'s algorithm, choosing the horizon of the rollout algorithm aims at finding a good compromise between bias and variance. The problem at hand is, however, different, since a single policy $\tilde{\pi}$ is evaluated beyond the first simulated transition (let us recall that the rollout algorithm has a complexity linear in H).

The following general result proposes a probably approximately correct bound on the evaluation error of a policy π by N simulated trajectories over a horizon H . This result is written for the γ -discounted criterion over an infinite horizon.

PROPOSITION 6.4. *Let $V_{H,N}^\pi(s) = \frac{1}{N} \sum_{n=1}^N \sum_{t=0}^{H-1} \gamma^t c_t^n$ be the estimation of $V^\pi(s)$ obtained by simulating a policy π over N trajectories of length H from s . With*

probability at least $1 - \Delta$:

$$|V^\pi(s) - V_{H,N}^\pi(s)| \leq \frac{c_{\max}}{1-\gamma} \left[\gamma^H + (1-\gamma^H) \sqrt{\frac{\ln(\frac{2}{\Delta})}{2N}} \right],$$

where c_{\max} is the maximum instant cost.

It is therefore sufficient, in order to obtain the optimal horizon H , to numerically minimize (the analytical resolution ends up with a fifth degree equation) this quantity for a given width N . The tolerance Δ can be chosen, for example, as equal to 0.05.

This result remains true beyond the rollout algorithm to evaluate a policy through simulations from a given state. An analogous result is established by Kearns and Singh [KEA 00] for the TD- λ algorithm. It is based on the theory of the analysis of large deviations. Nevertheless, their proof requires that the policy being considered be evaluated for each state in S , which limits its impact to MDPs of reasonable size.

6.3.4.2. Iterative allocation of simulations

For each current state, the rollout algorithm has the objective of determining the best of the $|A|$ actions through simulations. Ordinal optimization methods are directly applicable to perform an informed and incremental allocation of simulations with an algorithm like OCBA. Compared to a uniform allocation of N simulations for each action, they allow for improving the choice of the current action, at the price of additional computations at each iteration (see Algorithm 6.7 below).

Algorithm 6.7: Controlled rollout algorithm

Input: current state s_t , Markovian system simulator, policy to improve $\tilde{\pi}$, total number of simulations C , number of simulations allocated to each iteration n , tolerance Δ .

Calculate H^* minimizing the error of Proposition 6.4 by setting $N = \frac{C}{|A|H}$ for a given Δ

$l \leftarrow 1$

repeat

Calculate the ratios ρ_a for each action a with the OCBA method
For each action a , perform $n \cdot \rho_a$ additional trajectories of length H^* to calculate $Q^{\tilde{\pi}}(s_t, a)$

$l \leftarrow l + 1$

until $nl > C$

return $a_t = \operatorname{argmin}_{a \in A} Q^{\tilde{\pi}}(s_t, a)$

The controlled rollout algorithm thus proposes an improvement over the rollout algorithm by optimizing on the one hand the length of simulated trajectories and, on the other hand, by controlling the allocation of simulations using an ordinal optimization method (OCBA).

6.4. Conclusion

We have seen in this chapter the possibilities offered by online search methods to address large MDPs, in particular by employing stochastic simulation. Thus, an online search allows us to quickly determine a current action of good quality and to improve this choice while additional simulations are allocated. The key to an efficient online search lies in the simulation allocation strategy, which must realize a good compromise between depth and width on one side, and between exploration and exploitation on the other side.

Despite their advantages and although they are naturally employed in almost all two-player game programs, online search techniques are still relatively little exploited in the context of MDPs. Thus, the large majority of works on solving large MDPs concentrates on computing offline an approximate policy of value function. Yet, an online search can significantly improve the performance of a policy, although at the cost of an important computational effort.

If Tesauro' [TES 97] work has opened the way, few researchers have followed it. In addition to the works presented above, we can observe a recent gain in interest for online search methods for MDPs and, in particular, for search methods based on stochastic simulation – for example, the work of Chang [CHA 02, CHA 05] or Bent and Van Hentenryck [BEN 04] applied to packet routing in telecommunication networks. Let us finally mention the UCT algorithm by Kocsis and Szepesvari [KOC 06], based on the bandit theory and which proved experimentally to be efficient on various problems including the game of Go [GEL 06].

6.5. Bibliography

- [BAG 02] BAGNALL B., “Building a light-seeking robot with Q-learning”, *InformIT*, April 2002.
- [BAR 95] BARTO A. G., BRADTKE S. J. and SINGH S. P., “Learning to act using real-time dynamic programming”, *Artificial Intelligence*, vol. 72, pp. 81–138, 1995.
- [BAX 98] BAXTER J., TRIGDELL A. and WEAVER L., “KnightCap: a chess program that learns by combining TD(λ) with game-tree search”, *Proceedings of the 15th International Conference on Machine Learning (ICML'98)*, Morgan Kaufmann, San Francisco, CA, pp. 28–36, 1998.
- [BEA 82] BEAL D. F., “Benefits of minimax search”, CLARKE M. R. B., Ed., *Advances in Computer Chess 3*, Pergamon Press, Oxford, pp. 17–24, 1982.

- [BEN 04] BENT R. and VAN HENTENRYCK P., “The value of consensus in online stochastic scheduling”, *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS’04)*, Whistler, Canada, pp. 219–226, 2004.
- [BER 85] BERRY D. A. and FRISTEDT B., *Bandit Problems: Sequential Allocation of Experiments*, Chapman & Hall, London, 1985.
- [BER 97] BERTSEKAS D., “Differential training of rollout policies”, *Proceedings of the 35th Allerton Conference on Communication, Control, and Computing*, 1997.
- [BON 03a] BONET B. and GEFFNER H., “Labeled RTDP: improving the convergence of real-time dynamic programming”, *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS’03)*, Trento, Italy, pp. 12–21, 2003.
- [BON 03b] BONET B. and THIÉBAUX S., “GPT meets PSR”, *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS’03)*, Trento, Italy, pp. 102–111, 2003.
- [BRA 83] BRATKO I. and GAMS M., “Error analysis of the minimax principle”, *Advances in Computer Chess 3*, Pergamon Press, Oxford, pp. 1–15, 1982.
- [BUL 03] BULITKO V., LI L., GREINER R. and LEVNER I., “Lookahead pathologies for single agent search”, *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI’03)*, Acapulco, Mexico, pp. 1531–1533, 2003.
- [CAM 02] CAMPBELL M., HOANE JR. A. and HSU F., “Deep blue”, *Artificial Intelligence*, vol. 134, no. 1-2, pp. 57–83, 2002.
- [CHA 02] CHANG H. S., On-line sampling-based control for network queueing problems, PhD thesis, Purdue University, West Lafayette, IN, 2002.
- [CHA 05] CHANG H. S., FU M. C. and MARCUS S. I., “An adaptive sampling algorithm for solving Markov decision processes”, *Operations Research*, vol. 53, no. 1, pp. 126–139, 2005.
- [CHE 00a] CHEN C. H., LIN J., YUCESAN E. and CHICK S. E., “Simulation budget allocation for further enhancing the efficiency of ordinal optimization”, *Journal of Discrete Event Dynamic Systems: Theory and Applications*, vol. 10, no. 3, pp. 251–270, 2000.
- [CHE 00b] CHEN H. C., CHEN C. H. and YUCESAN E., “Computing efforts allocation for ordinal optimization and discrete event simulation”, *Journal of IEEE Transactions on Automatic Control*, vol. 45, no. 5, pp. 960–964, 2000.
- [DAV 98] DAVIES S., NG A. Y. and MOORE A., “Applying online search techniques to continuous-state reinforcement learning”, *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI’98)*, pp. 753–760, 1998.
- [DEA 95] DEAN T., Kaelbling L., KIRMAN J. and NICHOLSON A., “Planning under time constraints in stochastic domains”, *Artificial Intelligence*, vol. 76, no. 1-2, pp. 35–74, 1995.
- [GEL 06] GELLY S. and WANG Y., “Exploration exploitation in go: UCT for Monte-Carlo go”, *NIPS’06 Workshop on On-line Trading of Exploration and Exploitation*, 2006.
- [GOS 03] GOSAVI A., *Simulation-Based Optimization: Parametric Optimization Techniques and Reinforcement Learning*, Kluwer Academic Publishers, Boston, MA, 2003.

- [HAN 01] HANSEN E. and ZILBERSTEIN S., “LAO*: a heuristic search algorithm that finds solutions with loops”, *Artificial Intelligence*, vol. 129, pp. 35–62, 2001.
- [HER 90] HERNANDEZ O. and LASERRE J. B., “Error bounds for rolling horizon policies in discrete-time Markov control processes”, *IEEE Transactions on Automatic Control*, vol. 35, no. 10, pp. 1118–1124, 1990.
- [KAE 93] Kaelbling L. P., *Learning in Embedded Systems*, MIT Press, Cambridge, MA, 1993.
- [KAI 88] KAINDL H., “Minimaxing: theory and practice”, *AI Magazine*, vol. 9, no. 3, pp. 69–76, 1988.
- [KAI 90] KAINDL H., “Tree searching algorithms”, MARSLAND T. and SCHAEFFER J., Eds., *Computers, Chess, and Cognition*, Springer-Verlag, NY, pp. 133–158, 1990.
- [KEA 00] KEARNS M. and SINGH S., ““Bias-Variance” error bounds for temporal difference updates”, *Proceedings of the 13th Annual Conference on Computational Learning Theory (COLT’00)*, pp. 142–147, 2000.
- [KEA 02] KEARNS M. J., MANSOUR Y. and NG A. Y., “A sparse sampling algorithm for near-optimal planning in large Markov decision processes”, *Machine Learning*, vol. 49, no. 2-3, pp. 193–208, 2002.
- [KOC 06] KOCSIS L. and SZEPESVÁRI C., “Bandit based Monte-Carlo planning”, *Proceedings of the European Conference on Machine Learning (ECML’06)*, pp. 282–293, 2006.
- [KOE 04a] KOENIG S., LIKHACHEV M. and FURCY D., “Lifelong Planning A*”, *Artificial Intelligence*, vol. 155, no. 1-2, pp. 93–146, 2004.
- [KOE 04b] KOENIG S., LIKHACHEV M., LIU Y. and FURCY D., “Incremental heuristic search in AI”, *Artificial Intelligence Magazine*, vol. 25, no. 2, pp. 99–112, 2004.
- [KOR 85] KORF R. E., “Depth-first iterative-deepening: an optimal admissible tree search”, *Artificial Intelligence*, vol. 27, no. 1, pp. 97–109, 1985.
- [MCG 02] McGOVERN A., MOSS E. and BARTO A. G., “Building a basic block instruction scheduler with reinforcement learning and rollouts”, *Machine Learning*, vol. 49, no. 2-3, pp. 141–160, 2002.
- [MEU 99a] MEULEAU N. and BOURGINE P., “Exploration of multi-state environments: local measures and back-propagation of uncertainty”, *Machine Learning*, vol. 35, no. 2, pp. 117–154, 1999.
- [MEU 99b] MEULEAU N., PESHKIN L., KIM K.-E. and Kaelbling L., “Learning finite-state controllers for partially observable environments”, *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI’99)*, Morgan Kaufmann, San Mateo, CA, pp. 427–436, 1999.
- [NAU 83] NAU D. S., “Pathology on game trees revisited, and an alternative to minimaxing”, *Artificial Intelligence*, vol. 23, pp. 221–244, 1983.
- [NIL 80] NILSSON N., *Principles of Artificial Intelligence*, Morgan Kaufmann, San Francisco, CA, 1980.

- [PEA 83] PEARL J., “On the nature of pathology in game searching”, *Artificial Intelligence*, vol. 20, no. 4, pp. 427–453, 1983.
- [PEA 84] PEARL J., *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley, Reading, MA, 1984.
- [PER 04a] PERET L., Recherche en ligne pour les Processus Décisionnels de Markov: application à la maintenance d'une constellation de satellites, PhD thesis, Institut National Polytechnique de Toulouse, Toulouse, November 2004.
- [PER 04b] PERET L. and GARCIA F., “On-line search for solving Markov decision processes via heuristic sampling”, *Proceedings of the European Conference on Artificial Intelligence (ECAI'04)*, pp. 530–534, 2004.
- [RUM 94] RUMMERY G. A. and NIRANJAN M., On-line Q-learning using connectionist systems, Report, Cambridge University Engineering Department, Cambridge, UK, 1994.
- [RUS 95] RUSSELL S. and NORVIG P., *Artificial Intelligence: A Modern Approach*, Prentice Hall Series in Artificial Intelligence, Prentice Hall, Englewood Cliffs, NJ, 1995.
- [SUT 93] SUTTON R. S. and WHITEHEAD S. D., “Online learning with random representations”, *Proceedings of the 10th International Conference on Machine Learning*, Amherst, MA, pp. 314–321, 1993.
- [TES 97] TESAURO G. and GALPERIN G. R., “On-line policy improvement using Monte-Carlo search”, *Advances in Neural Information Processing Systems 9 (NIPS'96)*, MIT Press, Cambridge, MA, pp. 1068–1074, 1997.
- [TES 02] TESAURO G., “Programming backgammon using self-teaching neural nets”, *Artificial Intelligence*, vol. 134, no. 1-2, pp. 181–199, 2002.
- [THI 02] THIEBAUX S., KABANZA F. and SLANLEY J., “Anytime state-based solution methods for decision processes with non-Markovian rewards”, *Proceedings of the 18th Annual Conference on Uncertainty in Artificial Intelligence (UAI'02)*, Morgan Kaufmann, San Francisco, CA, pp. 501–510, 2002.

Part 2
Beyond MDPs

Chapter 7

Partially Observable Markov Decision Processes

Although most real-life systems can be modeled as Markov processes, it is often the case that the agent trying to control or to learn to control these systems has not enough information to infer the real state of the process. The agent observes the process but does not know its state. The framework of Partially Observable Markov Decision Processes (POMDPs) has been especially designed to deal with this kind of situation where an agent only has a partial knowledge of the process to control.

EXAMPLE 7.1. The car maintenance example developed in previous chapters (see sections 1.1 and 2.1) implicitly relied on the fact that the state of the car was known. More than often, this is not the case as no one is constantly checking the waterproofness of the cylinder head gasket or the wear-out of the brake lining. A quick look over does not give us the exact state of the car but only an observation with only partial and unreliable information. The framework of POMDPs makes it possible to model this kind of problem and the solutions obtained using this framework show how to choose optimal decisions despite partial information. But, once again, the dynamics of the process must be known, that is to say the consequences of the possible actions (transitions), the cost of actions (rewards) and the probabilities of each observation when in a given state (observation function).

Difficulties raised by the application of dynamic programming to real problems have been detected and discussed quite early. In 1965, Aström laid the theoretical foundations of POMDPs and showed how “belief states” could be used to solve them

Chapter written by Alain DUTECH and Bruno SCHERRER.

[AST 65]. The first real algorithms, although highly inefficient, are found in the work of Smallwood and Sondik [SMA 73, SON 71, SON 78]. The field really expanded thanks to the WITNESS algorithm [CAS 94] and to the ITERATIVE PRUNING method [ZAN 96]. Since then, numerous algorithms that give exact or approximate solutions to POMDPs have been designed.

This chapter deals with POMDPs. The first part (section 7.1) details the POMDP framework and introduces the notion of information state. Then we show that the direct application of MDP methods seen in previous chapters to POMDPs is doomed to fail (section 7.2). As shown in section 7.3, there exist some generic concepts that help design exact algorithms, based on Value Iteration (section 7.4) or Policy Iteration (section 7.5).

7.1. Formal definitions for POMDPs

7.1.1. Definition of a POMDP

A partially observable Markov decision process is an MDP where the agent does not know the real state of the process: the agent can only access an incomplete and partial observation of this state [CAS 98]. Similarly to a MDP (see section 1.2.1), a POMDP is defined by a tuple $(\mathcal{S}, \mathcal{A}, \Omega, T, p, O, r, b_0)$ where:

- \mathcal{S} is a state space;
- \mathcal{A} is an action space;
- Ω is an observation space;
- T is the time axis;
- $p()$ are transition probabilities between states;
- $O()$ are observation probabilities;
- $r()$ is a reward function defined on the transitions;
- b_0 is an initial probability distribution over states.

The only additions to MDPs are the initial probability distribution over states b_0 , the observation space Ω and the associated observation function $O()$.

As illustrated in Figure 7.1, the POMDP model states that, at each instant t from T , the agent does not know the current state $s_t \in \mathcal{S}$ but can only have a partial view of it as an observation $o_t \in \Omega$. This observation is given by the observation function O . When the agent performs an action $a_t \in \mathcal{A}$, the state of the process changes stochastically according to $p()$ to reach the new state s_{t+1} . The agent only receives the observation o_{t+1} , generated according to $O()$. Then, as for the MDP, the agent receives a reward $r \in \mathbb{R}$.

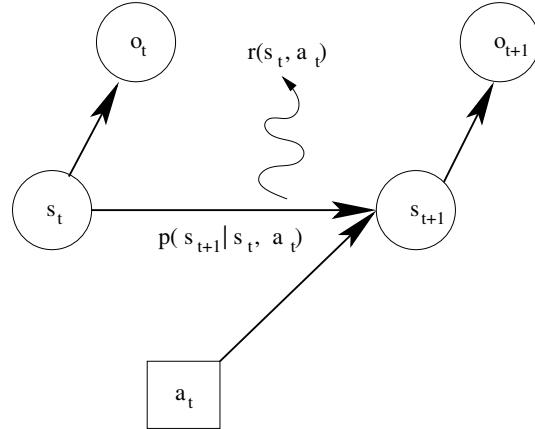


Figure 7.1. Generic view of a POMDP as an influence diagram

Like \mathcal{S} and \mathcal{A} , the observation space Ω is assumed to be finite. In the most general setting, the current observation depends on the last transition but, more often, the observation only depends on the current state of the process. At time t , the probability for the agent to observe o while the process is in state s is given by $\Pr(o | s) = O_t(o | s)$. Of course, we have that $\forall t, \forall s, \sum_o O_t(o | s) = 1$.

Even when the observation function $O()$ is deterministic, meaning that only one observation is associated with a state, it is still possible that the agent does not know the state of the process: two different states can be mapped to the same observation. Indeed, POMDPs are only to be considered in settings where there is an ambiguity on the state of the process, otherwise the problem to solve is in fact a MDP.

For the rest of this chapter, we will only consider settings with ambiguous observations. Furthermore, we will only work with stationary processes (i.e. $O()$ and $p()$ are not time-dependent).

7.1.2. Performance criteria

Like for MDPs, solving a POMDP aims at maximizing a given performance criterion. The family of criteria is the same as the one used for MDPs (see section 1.2.3).

Whereas the existence of performance criteria is not a problem for POMDPs, the very existence of the various value functions that were detailed for MDPs is problematic. In fact, value functions can be defined on the state space \mathcal{S} but this space is not accessible to the agent. For POMDPs, we must consider what are called

“information states” and their associated space because, as we will see, their existence is a necessary condition for defining proper value functions and thus solving POMDPs.

7.1.3. Information state

The agent, which does not know the state of the process, must choose its actions according to the only available information. This is why the term *information state* is used to speak about the spaces on which it is possible to define policies (or value functions) allowing the agent to control a POMDP in an optimal way.

7.1.3.1. Definition

Let I_t be the information available to an agent at time t . When the action a_t is executed, the agent perceives the process as an observation o_{t+1} , allowing it to update its available information. This process is described by the following equation:

$$I_{t+1} = \tau(I_0, I_1, \dots, I_t, o_{t+1}, a_t). \quad (7.1)$$

DEFINITION 7.1 (information state of a POMDP). *I is an information state if the sequence $(I)_t$ defines a controlled Markov chain (it is controlled by the actions taken by the agent), meaning that*

$$\forall t, \quad \Pr(I_{t+1} | I_0, I_1, \dots, I_t, o_{t+1}, a_t) = \Pr(I_{t+1} | I_t, o_{t+1}, a_t).$$

A special warning is needed here. Some spaces for defining policies may seem “natural” (like, e.g. the observation space Ω) *but*, in most cases, these spaces are not information spaces. Section 7.2 gives more details on this very crucial characteristic of POMDPs.

Given a proper space of information states, it is possible to define a transition function for this process by

$$\phi(I, a, o, I') = \Pr(I_t = I' | I_{t-1} = I, a, o). \quad (7.2)$$

The associated reward function is then

$$\rho(I, a) = \sum_{s \in \mathcal{S}} r(s, a) \Pr(s | I). \quad (7.3)$$

In theory, a POMDP can always be written as an MDP defined on a proper information state space. The biggest and central problem is to find a proper information state space such that it allows for a tractable representation of the information states.

7.1.3.2. Complete information state

The simplest way, although somewhat naive, to build information states is to represent them using the complete past of the process since time $t = 0$. In that case, an information state is made of the complete history of past observations and actions.

Formally, the *complete information state* at time t (denoted I_t^C) is made of:

- the initial probability distribution over states b_0 ,
- the history composed of all past observations and of the current observation (o_0, \dots, o_t) ,
- the history of past actions (a_0, \dots, a_{t-1}) .

Such complete information states clearly satisfy the Markov property described above. The main caveat of this approach is that the size of an information state grows at each time step. This representation can quickly become intractable and is clearly inappropriate for infinite horizon processes.

7.1.3.3. Sufficient statistics

A more efficient representation for information states can be based on sufficient or exhaustive statistics relative to the control of the process [BER 95].

DEFINITION 7.2 (sufficient information). A sequence of information states $(I)_t$ defines a sufficient information process when:

- $I_t = \tau(I_{t-1}, o_t, a_{t-1})$,
- $\Pr(s_t | I_t) = \Pr(s_t | I_t^C)$,
- $\Pr(o_t | I_{t-1}, a_{t-1}) = \Pr(o_t | I_{t-1}^C, a_{t-1})$,

where I_{t-1}^C and I_t^C are complete information states.

A sufficient information state is a state that allows predicting the behavior of the process, and therefore allows us to control it. These “meta”-information on the process obey the Markov property and preserve enough information from complete information states to control the process. The main advantage of sufficient information states is their more compact representation. Their size does not grow with time. The drawbacks are their update process that is a bit more complex and the fact that they are defined over a space that may be continuous, thus infinite.

7.1.3.4. Belief states

Widely used sufficient states are belief states.

DEFINITION 7.3 (belief state). A belief state b_t at time t is defined by

$$b_t(s) = \Pr(s_t = s \mid I_t^C).$$

A belief state is a probability distribution on the state space. The set of belief states is denoted by \mathcal{B} ; it is the space of all probability distributions on \mathcal{S} .

As previously mentioned, a sufficient information state must be updated after the execution of an action. Thus, if b is a belief state, after a transition (executing action a and receiving observation o) of the POMDP process, it becomes b_o^a where

$$\begin{aligned} b_o^a(s') &= \Pr(s' \mid b, a, o) \\ &= \frac{\Pr(s', b, a, o)}{\Pr(b, a, o)} \\ &= \frac{\Pr(o \mid s', b, a) \Pr(s' \mid b, a) \Pr(b, a)}{\Pr(o \mid b, a) \Pr(b, a)} \\ &= \frac{\Pr(o \mid s', b, a) \Pr(s' \mid b, a)}{\sum_{s'' \in \mathcal{S}} \Pr(s'' \mid b, a) \Pr(o \mid s'', a)} \\ &= \frac{O(o \mid s') \sum_{s \in \mathcal{S}} \Pr(s' \mid a, s) \Pr(s)}{\sum_{s \in \mathcal{S}} \sum_{s'' \in \mathcal{S}} O(o \mid s'') \Pr(s'' \mid a, s) \Pr(s)} \\ &= \frac{O(o \mid s') \sum_{s \in \mathcal{S}} p(s' \mid s, a) b(s)}{\sum_{s \in \mathcal{S}} \sum_{s'' \in \mathcal{S}} O(o \mid s'') p(s'' \mid s, a) b(s)}. \end{aligned}$$

This leads to

$$b_o^a(s') = \frac{O(o \mid s') \sum_{s \in \mathcal{S}} p(s' \mid s, a) b(s)}{\sum_{s \in \mathcal{S}} \sum_{s'' \in \mathcal{S}} O(o \mid s'') p(s'' \mid s, a) b(s)}. \quad (7.4)$$

A sequence of belief states forms a Markov process that can be explicated, in particular its transition and reward functions. For this, the conditional probability of an observation is defined as

$$\begin{aligned} \omega(b, a, o) &= \Pr(o \mid b, a) \\ &= \sum_{s \in \mathcal{S}} \sum_{s'' \in \mathcal{S}} O(o \mid s'', a) p(s'' \mid s, a) b(s). \end{aligned}$$

Then, the transition function between two belief states can be calculated. Given a belief state b and an action a , each observation o leads to a different belief state b_o^a .

The transition probability to a belief state b' is the sum of all transition probabilities leading to the belief states b_o^a equal to b' . This is written as

$$\phi(b, a, b') = \Pr(b' | b, a) = \sum_{o \in \Omega} \omega(b, a, o) \delta(b', b_o^a),$$

with

$$\delta(x, y) = \begin{cases} 0 & \text{if } x = y, \\ 1 & \text{otherwise.} \end{cases}$$

The reward associated with a belief state can be calculated by

$$\rho(b, a) = \sum_{s \in \mathcal{S}} r(s, a) b(s).$$

The Markov decision process defined using belief states is quite difficult to solve as its state space, the space of probability distributions on \mathcal{S} , is continuous. Furthermore, except for a small sub-family of POMDPs called “transient” (see section 7.3.2.2), the sequence of belief states generated by a given policy is made of an infinite number of different belief states. Nevertheless, section 7.3 shows that it is possible to take advantage of the value function defined on this Markov decision process to define exact or nearly exact algorithms.

7.1.4. Policy

In the case of POMDPs, there are more possibilities to define policies than there were for MDPs. It is not possible to define a policy on the state space or on the space of state histories as they are not within the reach of the agent. A simple solution is to base the policies on the observation space or on the histories of observations. As explained in section 7.2, these solutions do not guarantee that the policies will be optimal but, because of their simplicity, they may sometimes prove to be valuable alternatives.

It is more pertinent to work with the information spaces that were introduced in the previous section. There is a large choice of possible complete information states but they are more or less equivalent to histories of actions and observations. We will now detail a possible representation of policies using histories of actions and observations, a method that can easily be ported to other spaces like, for example, histories of observations only.

Starting from an initial complete information state I , the optimal policy of finite horizon N is a kind of conditional plan that can be represented as a tree. In fact, for a horizon of 1, the optimal policy for the information state I is the optimal action from

this information state, denoted by $a^1(I)$. Suppose that, for a horizon of 2, the optimal policy from I begins with action a . The agent then receives observation o and is now in the new information state I_o^a . The next optimal action is then $a^1(I_o^a)$, according to the optimal policy of horizon 1. The second action chosen depends on the observation o and, as such, the optimal policy for horizon 2 is a kind of conditional plan.

As a consequence, a policy of horizon N for an information state I can be represented as a policy tree like the one depicted in Figure 7.2. Executing a conditional plan or an N -order policy from an information state I can be seen as descending this policy tree by starting from the root node and going alternatively through “action” nodes and “observation” branches. When in a node, the associated action is applied and then, according to the observation received, the next node is reached by following the branch labeled by the observation.

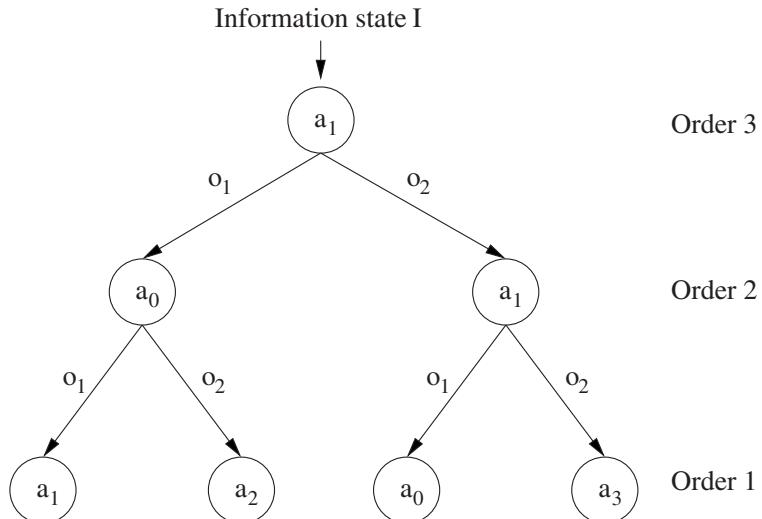


Figure 7.2. Conditional plan: This tree represents a policy of horizon 3 for an information state I . It is also a kind of conditional plan. At each step of the plan, a node has an action to apply and, according to the observation received, following the appropriate branch leads to the next node and the next action

Policy trees are adequate for policies with a finite horizon. They are more difficult to use for problems with infinite horizon as the trees grow exponentially. Finite state automata are a possible way out but only for policies that are cyclic. Figure 7.3 offers an example of such an automaton that is run like a policy tree was: the action in the current node is applied then, according to the received observation, a transition is made to the next node by following the appropriate branch. The starting node for the policy is a function of the initial information state.

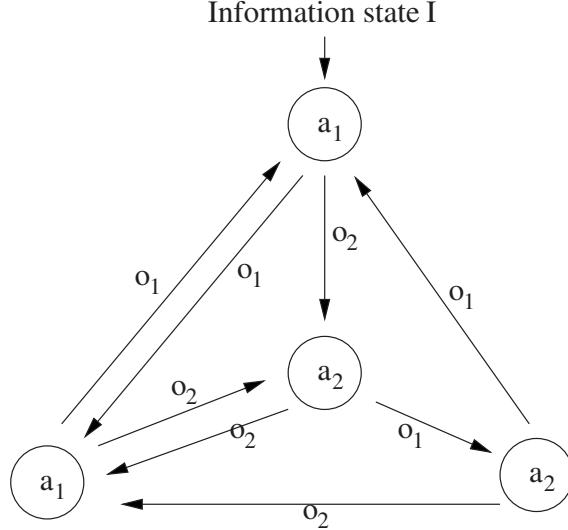


Figure 7.3. Finite state automaton for an infinite cyclic policy: Finite state automata are a way to encode infinite policies with cycles. Similarly to trees, running a policy means applying the action of the current node and branching to the next node according to the received observation

7.1.5. Value function

Similarly to MDPs, a value function exists for POMDPs defined on information states. For ease of understanding, only the discounted criterion will be presented in this section but all the development and computation presented in section 1.5 with the various criteria for MDPs could be used here instead.

The iteration operator for the value function of a policy π_t defined on the information space \mathcal{I} is written as

$$V_n^{\pi_t}(I) = \rho(I, \pi_t(I)) + \gamma \sum_{I' \in \mathcal{I}} \phi(I, \pi_t(I), I') V_{n-1}^{\pi_t}(I'). \quad (7.5)$$

Then, the Bellman equation that describes a fundamental property of the optimal value function for the MDP defined on information states is

$$\mathcal{V}^*(I) = \max_{a \in \mathcal{A}} \left[\rho(I, a) + \gamma \sum_{I' \in \mathcal{I}(I, a)} \phi(I, a, I') \mathcal{V}^*(I') \right], \quad (7.6)$$

where $\mathcal{I}(I, a)$ is the set of successors to the information state I . Previously, we showed how to calculate the successors of I by saying that $I' = \tau(I, o, a)$, meaning that

there are, at most, $|\Omega|$ successors to an information state. Summing over the set of observations leads to

$$\mathcal{V}^*(I) = \max_{a \in \mathcal{A}} \left[\rho(I, a) + \gamma \sum_{o \in \Omega} \Pr(o | I, a) \mathcal{V}^*(\tau(I, o, a)) \right]. \quad (7.7)$$

By directly using the parameters of the POMDP, we obtain

$$\begin{aligned} \mathcal{V}^*(I) = \max_{a \in \mathcal{A}} & \left[\sum_{s \in \mathcal{S}} r(s, a) \Pr(s | I) \right. \\ & \left. + \gamma \sum_{s \in \mathcal{S}} \sum_{s' \in \mathcal{S}} \sum_{o \in \Omega} \Pr(s | I) p(s' | s, a) O(o | s') \mathcal{V}^*(\tau(I, o, a)) \right]. \end{aligned} \quad (7.8)$$

As in the case of MDPs, the operator associated with this equation is a contraction for the max norm when $0 \leq \gamma < 1$. The same mathematical demonstrations show the existence and the unicity of the solution (see Theorems 1.4, 1.5 and 1.6).

7.2. Non-Markovian problems: incomplete information

As written before, generally, observations are not information states for POMDPs. In other words, an agent cannot optimally control a POMDP when relying on the sole current observation. In this section, we will study in more details the properties of policies defined on observations so as to better understand the differences between POMDPs and MDPs. This understanding will also be useful later on when dealing with the exact resolution of POMDPs.

7.2.1. Adapted policies

In this section, only the following form of policy π will be considered:

$$\pi : (\Omega \times T) \longrightarrow \Pi(\mathcal{A}),$$

where $\Pi(\mathcal{A})$ is the set of probability distributions on \mathcal{A} .

The basic and somehow naive adaptation to POMDPs of the algorithms designed for MDPs leads to algorithms that look for policies of the form just stated before. We say that these policies are “adapted” from MDPs. In fact, we will see that they *are not adapted* to optimal control even if some practical implementations can sometimes bring interesting results.

7.2.2. Discounted reward

7.2.2.1. Adapted stochastic policies

For this section, let us consider only the case of discounted rewards. It is quite straightforward to prove that there is not always an optimal deterministic policy for POMDPs, contrarily to MDPs. The example of Figure 7.4 clearly demonstrates the following.

PROPOSITION 7.1. *There exist POMDPs for which the best stochastic adapted policy can be arbitrary better than the best deterministic adapted policy.*

Proof. Figure 7.4 shows a POMDP with two states ($1a$ and $1b$) and only one observation (1). Only two different deterministic adapted policies exist (either always pick action “A” or always pick action “B”). In the best case, these policies bring a reward of $+R$ followed by an infinite sequence of $-R$, that is, a discounted value of $\frac{(1-2\gamma)R}{1-\gamma}$. The stochastic adapted policy that chooses either action with a probability of 0.5 brings, on the average, a reward of 0. Then, the difference between the value of any deterministic policy and the value of the stochastic policy can be arbitrary large provided $\gamma > 0.5$ and R is large enough. \square

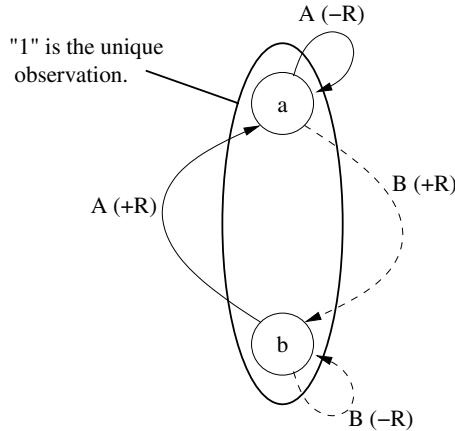


Figure 7.4. A need for stochastic adapted policies: The POMDP of this figure is made of two states ($1a$ and $1b$) that both generate the same observation “1”, that is represented with an ellipsis. Facing this unique observation, the agent can chose between action “A” and “B” that, according to the underlying state, will bring a positive ($+R$) or negative ($-R$) reward. No optimal deterministic adapted policy can be found

Furthermore, it is also possible to prove, still using simple examples (see [JAA 95]), the following.

PROPOSITION 7.2. *There exist POMDPs for which the best stochastic adapted policy can be arbitrary worse than the optimal policy of the underlying MDP.*

PROPOSITION 7.3. *There exist POMDPs for which the best adapted policy can be non-stationary.*

7.2.2.2. Adapted value function

The fact we just demonstrated questions the very existence of an optimal adapted policy. A way to answer is to study the existence of an optimal value function.

From an adapted policy $\pi : \Omega \rightarrow \Pi(\mathcal{A})$, it is possible to build a policy π' defined on the state of the MDP by

$$\Pr^{\pi'(a|s)} = \sum_{o \in \Omega} \Pr(o | s) \Pr^{\pi}(a | o) = \sum_{o \in \Omega} O(o | s) \pi(a | s). \quad (7.9)$$

Then, knowing the underlying states of the POMDP, this policy π' can be applied to the POMDP and its value function $\mathcal{V}_{\pi'}$ can be calculated, obeying

$$\mathcal{V}_{\pi'}(s) = \sum_{a \in \mathcal{A}} \Pr(a | \pi', s) \left[r(s) + \gamma \sum_{s' \in \mathcal{S}} p(s' | s, a) \mathcal{V}_{\pi'}(s') \right].$$

This property uses the fact that, for the policy π' , the states come from a Markov process. This is clearly not the case when π is run on the observations as they are not information states. As a consequence, it is not possible to define a value function over states for an adapted policy, that is, a policy defined on the observations.

But, $\mathcal{V}_{\pi'}$ can be used to define the value function of an observation by saying that it is the expected value function of the states underlying this observation when π' is run. We call this new value function an *adapted value function*.

DEFINITION 7.4 (adapted value function). *For an adapted policy π , there exists a policy π' defined on the states by equation (7.9) that can be used to define an adapted value function of π in the following way:*

$$\vartheta^{\pi}(o) = \sum_{s \in \mathcal{S}} \Pr^{\pi}(s | o) \mathcal{V}_{\pi'}(s), \quad (7.10)$$

where $\Pr^{\pi}(s | o)$ is the asymptotic probability distribution over the states, i.e. the probability that the state of the underlying MDP is s when one observes o as t tends to infinity and $\mathcal{V}_{\pi'}$ is the value function of π' .

This is only a definition and it does not say how an agent can calculate the adapted value function of an observation as it does not know s . Nevertheless, with this

definition it is possible to show that, contrary to MDPs, a POMDP does not necessarily admit a stationary policy that maximizes the value of all observations simultaneously. For a MDP, the optimal policy simultaneously maximizes the value of all the states.

This statement can be proved using the POMDP of Figure 7.5. This POMDP has four states (1 , $2a$, $2b$ and 3) and three observations (1 , 2 and 3), the only possible choice is the action for observation “ 1 ”. If the probability of choosing action “ A ” is increased, the value of observation “ 1 ” is also increased but the value of “ 2 ” is decreased. The opposite effect is obtained when the probability of choosing action “ B ” is increased. As a consequence, the value of both observations “ 1 ” and “ 2 ” cannot be increased simultaneously.

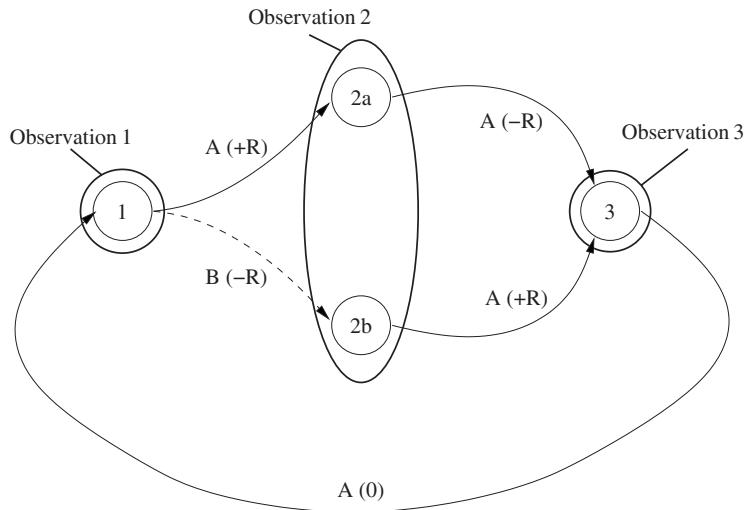


Figure 7.5. No optimal adapted policy: The POMDP of this figure is made of four states (1 , $2a$, $2b$ and 3) and three observations (1 , 2 and 3). The only possible choice for the policy is the action chosen for observation “ 1 ”. Increasing the probability of action “ A ” increases the value for “ 1 ” but decreases the value of “ 2 ”. This shows that it is not possible to find an adapted policy that maximizes the value of all observations

Thus, the adapted value function is quite different from the value function of an MDP. In particular, it is not possible to define an *optimal* adapted value function and, consequently, there are no optimal adapted policies, at least, as far as the γ discounted reward criterion is concerned.

7.2.2.3. Convergence of adapted algorithms

Considering that neither optimal adapted value functions nor optimal adapted policies exist for POMDPs, it seems legitimate to question the utility of trying to

adapt to POMDPs the algorithms developed for MDPs. In other words, we would like to understand what kind of results can be expected when classical algorithms for MDPs are applied to POMDPs by assimilating observations to states. In fact, even the convergence of these adapted algorithms cannot be guaranteed as the existence of an optimal value function was required to prove their convergence for MDPs.

When looking in more detail, two main factors are to be considered. First, the value of an observation depends on the occupation probability of its underlying states. So, algorithms that do not alter this probability distribution should work quite well with POMDPs. Second, in general, we have seen that it is not possible to maximize the value of all observations at the same time. As a consequence, algorithms that do try to find this dominant value function are quite likely to behave strangely (e.g. value iteration or policy iteration as the non-linear step of maximization is a potential source of instability). Nevertheless, algorithms that have a stationary exploration strategy of the environment and that are based on a stochastic approximation of the value function might be able to converge, mathematically speaking. Results found in the literature seem to support this intuitive affirmation as the convergence of two algorithms of this kind has been proven (see [SIN 94]).

TD(0): for a POMDP, with classical hypotheses for convergence, the TD(0) algorithm converges with probability 1 to the solution of the following system of equations:

$$\forall o \in \Omega, \quad \vartheta^\pi(o) = \sum_{s \in \mathcal{S}} \Pr^\pi(s | o) \left[r(s) + \gamma \sum_{o' \in \Omega} \Pr^\pi(s, o') \vartheta^\pi(o') \right], \quad (7.11)$$

where $\Pr^\pi(s, o') = \sum_{s' \in \mathcal{S}} \Pr^\pi(s' | s) O(o' | s')$.

We must note that the value function to which TD(0) converges is not necessarily the value function defined by equation (7.10). In fact, it is rather easy to find simple examples where this is not the case.

Q-Learning: for a POMDP, an adapted Q-learning with a stationary exploration policy π_{exp} converges with probability 1 to the solution of the following set of equations (with classical convergence hypotheses):

$$Q(o, a) = \sum_{s \in \mathcal{S}} \Pr^{\pi_{\text{exp}}}(s | o, a) \left[r(s, a) + \gamma \sum_{o' \in \Omega} \Pr^a(s, o') \max_{a' \in \mathcal{A}} Q(o', a') \right],$$

where $\Pr^{\pi_{\text{exp}}}(s | o, a)$ is the asymptotic occupation probability of s under the exploration policy π_{exp} when we observe o after having applied action a and where $\Pr^a(s, o') = \sum_{s' \in \mathcal{S}} p(s' | s, a) O(o' | s')$.

The convergence is guaranteed only if the environment is explored using a stationary policy (this is not the case with an ϵ -greedy algorithm, for example). Besides, the Q-Learning is limited by the fact that it can only converge to a deterministic policy which, as explained in section 7.2.2.1, can be largely sub-optimal.

These caveats are more of a problem for learning than for planning because, as shown later on, searching for optimal policies when the model is known is partially solved.

7.2.3. Adapted algorithms and adapted average reward criterion

As explained, it is not possible to define a value function that is optimal for all observations simultaneously. In order to compare two adapted policies so as to try to find an optimal one, the idea is to “map” the value function to a scalar value, for example, by saying that an adapted policy will maximize $\hat{\vartheta} = \sum_{o \in \Omega} P_\Omega(o) \vartheta^\pi(o)$ where P_Ω is a probability distribution on Ω . Numerous choices are possible for this probability distribution but a natural one could be the initial probability distribution over observations or the asymptotic probability distribution over the observations. This last option is in fact equivalent to using the *average criterion* and its associated utility U^π (see section 1.5.4), as proven by [SIN 94].

To show this, let us assume that $\Pr^\pi(o)$ is the asymptotic probability of the observation o when the policy π is run. Then, the scalar criterion can be written as

$$\begin{aligned} \sum_{o \in \Omega} \Pr^\pi(o) \vartheta^\pi(o) &= \sum_{o \in \Omega} \Pr^\pi(o) \sum_{s \in \mathcal{S}} \Pr^\pi(s | o) V^\pi(s) \\ &= \sum_{s \in \mathcal{S}} \sum_{o \in \Omega} \Pr^\pi(o) \Pr^\pi(s | o) V^\pi(s) \\ &= \sum_{s \in \mathcal{S}} \Pr^\pi(s) V^\pi(s) \\ &= \sum_{s \in \mathcal{S}} \Pr^\pi(s) r(s, \pi(s)) + \gamma \sum_{s \in \mathcal{S}} \Pr^\pi(s) \sum_{s' \in \mathcal{S}} p(s' | s, \pi(s)) V^\pi(s') \\ &= U^\pi + \gamma \sum_{s' \in \mathcal{S}} \Pr^\pi(s') V^\pi(s') \\ &= U^\pi + \gamma \sum_{o \in \Omega} \Pr^\pi(o) \vartheta^\pi(o) \end{aligned}$$

and thus $\sum_{o \in \Omega} \Pr^\pi(o) \vartheta^\pi(o) = \frac{U^\pi}{1-\gamma}$.

It is then tempting to look for an adapted stochastic policy that is optimal for this criterion, i.e. the average reward. To our knowledge, the only existing algorithm comes from the work of Jaakkola, Singh and Littman [JAA 95]. This algorithm belongs to the family of Policy Iteration and is grounded on a Monte Carlo evaluation of the average reward of a policy that allows the computation of *Q-values* in order to derive a “better” policy. The main problem with this algorithm is that it has nearly never been experimentally tested and that it can only converge to a local optimum of the scalar value function.

Chapter 5 on gradient methods for the search for optimal policies lists other methods that look for approximate solutions to POMDPs, especially in section 5.2.5.

7.3. Computation of an exact policy on information states

Using information states, it is possible to map a POMDP into an MDP. In theory, resolution methods designed for MDPs can be used to find an optimal policy. In practice, the problem is quite complex because of the nature and the size of the space of information states. As shown by [PAP 87], solving a POMDP with a unique initial state and a finite horizon is already a PSPACE-hard problem. This family of problems has a complexity in memory size, and thus in computation time, that is at least polynomial in the size of the problem. If the horizon N is smaller than the size of \mathcal{S} , then the POMDP is PSPACE-complete.

NOTE 7.1. For the rest of this chapter, only the discounted reward criterion and its associated value function V are considered.

7.3.1. The general case

7.3.1.1. Finite horizon

Theoretically, for a finite horizon N , tools from the dynamic programming framework are able to find an optimal policy and an optimal value function. Thus, starting from the value function $V_0^* = \max_{a \in \mathcal{A}} \rho(I, a)$ for the last time step of the problem, the dynamic programming operator (equation (7.7), section 7.1.5) must be applied N times in the following way:

$$V_n^*(I) = \max_{a \in \mathcal{A}} \left[\rho(I, a) + \gamma \sum_{o \in \Omega} \Pr(o | I, a) V_{n-1}^*(\tau(I, o, a)) \right]. \quad (7.12)$$

Equation (7.12) defines another operator that will be denoted by L . Then, we have $V_n^* = LV_{n-1}^*$.

Once the optimal value function is known, the optimal policy μ_n^* can be deduced using

$$\mu_n^*(I) = \operatorname{argmax}_{a \in \mathcal{A}} \left[\rho(I, a) + \gamma \sum_{o \in \Omega} \Pr(o | I, a) V_{n-1}^*(\tau(I, o, a)) \right].$$

7.3.1.2. Infinite horizon

Let us consider an infinite horizon. The operator L previously defined has the same characteristics as the operator used for MDPs in section 1.5.2, i.e. it is a contraction (for all functions U and V , we have that $\|LU - LV\| \leq \gamma \|U - V\|$, where $\|V\| = \max_{\mathcal{I}} V(I)$).

It is then possible to find an ϵ -optimal value function by using a value iteration-like method where each step is

$$V_{i+1} = LV_i. \quad (7.13)$$

As an optimal value function does exist and as L is a contraction, the Banach theorem proves that these iterations converge to the optimal value function V^* . Then, a finite number of iterations are enough to reach an ϵ -optimal solution from which an optimal policy can be deduced.

7.3.2. Belief states and piecewise linear value function

Practically speaking, the previous computation schemes relying on a variant of value iteration are not tractable. The problem lies in the fact that the space of information states is continuous or, at least, huge. It is then impossible to calculate or represent the optimal value function and an optimal policy.

Belief states and POMDPs that do admit such information states have value functions that show special characteristics. Based on this, it is possible to build more efficient and more tractable algorithms.

The optimal value function for a finite horizon problem is piecewise linear and convex (PWLC) [SON 71]. Thanks to this important property, the value function can be represented with only a finite number of parameters, as shown by Figure 7.6.

As the value function is defined on the space \mathcal{B} which is $|\mathcal{S}| - 1$ -dimensional, each linear segment used to represent it is $|\mathcal{S}|$ -dimensional. It can be encoded by a vector θ with $|\mathcal{S}|$ coefficients. The s th component of such a vector will be denoted by $\theta(s)$. The set of vectors coding the PWLC value function is denoted by Θ . We say that Θ

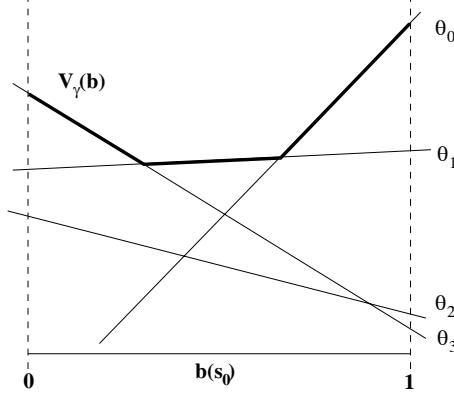


Figure 7.6. Piecewise linear convex value function: In this example, the value function V of a POMDP with two states (s_0 and s_1) is represented with only 4 vectors of parameters θ_i , each one being 2-dimensional. The space of information states lies along the x -axis and the values on the y -axis. A single probability $b(s_0)$ is enough to describe the belief state as $b(s_1) = 1 - b(s_0)$. On this figure, each linear segment of the value function is drawn using a fine line whereas the value function is in bold

represents V , meaning that

$$V(b) = \max_{\theta \in \Theta} \sum_{s \in S} b(s)\theta(s) \quad (7.14)$$

$$= \max_{\theta \in \Theta} b \cdot \theta. \quad (7.15)$$

Now we must prove that the optimal value function is piecewise linear convex. For this, we will need the following theorem, proven by Smallwood.

THEOREM 7.1 [SMA 73] (piecewise linear convex value function). *Let L be the Bellman operator defined by equation (7.12) and V_{init} an initial value function defined on the space \mathcal{B} of belief states that is piecewise linear convex. Then, for a POMDP that accepts belief states, we have that:*

– $V_n = L^n V_{\text{init}}$, the result of n applications of the operator L to V_{init} , is also piecewise linear convex on \mathcal{B} ;

– V_n can be represented by a finite set $\Theta = \{\theta\}$ of vectors of size $|S|$ by $V_n(b) = \max_{\theta \in \Theta} b \cdot \theta$.

Proof. The idea is to show that if, after $i - 1$ applications of operator L , the value function V_{i-1} is piecewise linear convex, then the value function V_i resulting from another application of L is also PWLC.

Let us suppose that V_{i-1} is PWLC, then there exists Θ_{i-1} such that

$$V_{i-1} = \max_{\theta_{i-1} \in \Theta_{i-1}} \sum_{s' \in \mathcal{S}} b_{i-1}(s') \theta_{i-1}(s').$$

If a is the action applied when i actions were still to be chosen and when the observation received was o , then the belief state b_{i-1} is written as

$$b_{i-1} = \Pr(s' | b_i, a, o),$$

leading to

$$V_{i-1} = \max_{\theta_{i-1} \in \Theta_{i-1}} \sum_{s' \in \mathcal{S}} \Pr(s' | b_i, a, o) \theta_{i-1}(s').$$

Then, by replacing V_{i-1} in equation (7.12), which defines operator L , we have

$$V_i(b_i) = \max_{a \in \mathcal{A}} \left[\rho(b_i, a) + \gamma \sum_{o \in \Omega} \Pr(o | b_i, a) \max_{\theta_{i-1} \in \Theta_{i-1}} \sum_{s' \in \mathcal{S}} \Pr(s' | b_i, a, o) \theta_{i-1}(s') \right],$$

that can be rewritten as

$$\begin{aligned} V_i(b_i) &= \max_{a \in \mathcal{A}} \left[\sum_{s \in \mathcal{S}} r(s, a) b_i(s) + \gamma \sum_{o \in \Omega} \Pr(o | b_i, a) \max_{\theta_{i-1} \in \Theta_{i-1}} \sum_{s' \in \mathcal{S}} \Pr(s' | b_i, a, o) \theta_{i-1}(s') \right] \\ &= \max_{a \in \mathcal{A}} \left[\sum_{s \in \mathcal{S}} r(s, a) b_i(s) + \gamma \sum_{o \in \Omega} \max_{\theta_{i-1} \in \Theta_{i-1}} \sum_{s' \in \mathcal{S}} \Pr(o | b_i, a) \Pr(s' | b_i, a, o) \theta_{i-1}(s') \right] \\ &= \max_{a \in \mathcal{A}} \left[\sum_{s \in \mathcal{S}} r(s, a) b_i(s) + \gamma \sum_{o \in \Omega} \max_{\theta_{i-1} \in \Theta_{i-1}} \sum_{s' \in \mathcal{S}} \Pr(s', o | b_i, a) \theta_{i-1}(s') \right] \\ &= \max_{a \in \mathcal{A}} \left[\sum_{s \in \mathcal{S}} r(s, a) b_i(s) + \gamma \sum_{o \in \Omega} \max_{\theta_{i-1} \in \Theta_{i-1}} \sum_{s' \in \mathcal{S}} \left[\sum_{s \in \mathcal{S}} \Pr(s', o | s, a) b_i(s) \right] \theta(s') \right]. \end{aligned}$$

The optimal element of Θ_{i-1} for b , a and o is called $\theta_{i-1}^{a,o}(b)$ (i.e. it is the element such that $\theta_{i-1}^{a,o}(b) = \operatorname{argmax}_{\theta_{i-1} \in \Theta_{i-1}} \sum_{s \in \mathcal{S}} \Pr(s | b_i, a, o) \theta_{i-1}(s)$). It comes down to finding the segment that defines the value function for the given b , a and o . Using this, we can write

$$\begin{aligned} V_i(b_i) &= \max_{a \in \mathcal{A}} \left[\sum_{s \in \mathcal{S}} r(s, a) b_i(s) + \gamma \sum_{o \in \Omega} \sum_{s' \in \mathcal{S}} \left[\sum_{s \in \mathcal{S}} \Pr(s', o | s, a) b_i(s) \right] \theta_{i-1}^{a,o}(b_i, s') \right] \\ &= \max_{a \in \mathcal{A}} \left[\sum_{s \in \mathcal{S}} b_i(s) \left[r(s, a) + \gamma \sum_{o \in \Omega} \sum_{s' \in \mathcal{S}} \Pr(s', o | s, a) \theta_{i-1}^{a,o}(b_i, s') \right] \right]. \end{aligned}$$

In the previous equation, the expression inside the innermost square brackets can be represented using $|\mathcal{A}||\Theta_{i-1}|^{|\Omega|}$ different vectors of size $|\mathcal{S}|$. Indeed, a maximum of one vector is needed for each choice of a and of a sequence of $|\Omega|$ of Θ_{i-1} . If we call Θ_i the set made of these vectors, it is possible to write $V_i(b_i)$ as

$$V_i(b_i) = \max_{\theta_i \in \Theta_i} \sum_{s \in \mathcal{S}} b_i(s) \theta_i(s).$$

This means that V_i is a piecewise linear convex function and that it can be represented by a finite set of vectors θ_i from Θ_i . Each of these vectors θ_i can be written as

$$\theta_i(b, s) = r(s, a) + \gamma \sum_{o \in \Omega} \sum_{s' \in \mathcal{S}} \Pr(s', o | s, a) \theta_{i-1}^{a, o}(b, s'). \quad (7.16)$$

As the initial value function V_{init} is piecewise linear convex, the previous theorem shows that the function obtained after any finite number of applications of the operator L is still PWLC. This ends the proof. \square

To show that the value function is PWLC with the previous theorem, we must prove that any optimal value function for a horizon of 1 is PWLC. Indeed, when there is only one action a to choose, only the instantaneous reward influences the value function. This leads to

$$\begin{aligned} V_1^*(b) &= \max_{a \in \mathcal{A}} \sum_{s \in \mathcal{S}} b(s) r(s, a) \\ &= \max_{a \in \mathcal{A}} b \cdot r(a). \end{aligned}$$

V_1^* is thus clearly PWLC and can be represented by at most $|\mathcal{A}|$ vectors.

So, using the previous theorem, we prove that any optimal value function for a finite horizon is piecewise linear convex. Furthermore, thanks to the constructive proof of the theorem, we can also show that:

- it is always possible to calculate the optimal value function in a finite number of operations;
- the optimal value function for a finite horizon can be represented by a finite set of $|\mathcal{S}|$ -dimensional vectors;
- the optimal policy is also computable in finite time.

7.3.2.1. Choice of the θ vectors

An optimal value function for a finite horizon can be represented by a finite number of vectors. The proof of Theorem 7.1 shows that a maximum of $|\mathcal{A}||\Theta_{i-1}|^{|\Omega|}$ vectors is needed to represent V_i . In fact, this number can be reduced by only considering the

set Ω^{poss} made of the observations that can be observed in a given belief state. The upper limit becomes $|\mathcal{A}| |\Theta_{i-1}|^{|\Omega|^{\text{poss}}}$. Practically speaking, far less vectors are needed to represent the value function. Some vectors (as, e.g. the vector θ_2 in Figure 7.6) are dominated by the others and do not influence the value function. This kind of vector is called a *dominated vector*. Vice versa, a vector that is absolutely needed for computing the value function (even for only one belief state) is called a *useful vector*.

It is easy to understand that the computation of the value function is made easier as the number of representative vectors is reduced. For a given belief state I , every vector of the representation is used to calculate the value function. It is then very interesting to prune every dominated vector from a representation, a complex problem that [LIT 95] proved to be done efficiently only if RP = NP, i.e. only if every non-deterministic decision algorithm has a significant probability of success.

Searching for value functions must face a possible exponential explosion of the number of vectors (in $|\Omega|$) but must also face the difficult task of finding useful vectors.

Search algorithms that look for optimal functions – and which will be detailed in section 7.4 – explore several methods to make this search for useful vectors more efficient.

7.3.2.2. Infinite horizon

Even if every value function V_n^* is piecewise linear convex and

$$\lim_{n \rightarrow \infty} \|V_n^* - V^*\| = 0,$$

nothing tells us that the optimal value function for an infinite horizon is also PWLC. [SON 71] has shown that a family of POMDPs– called *transient* POMDPs– do have optimal value functions V^* that are PWLC. When the POMDP is not transient, the optimal policy for an infinite horizon can be approximated as closely as needed by a transient policy with a PWLC optimal value function. In that case, we speak of ϵ -optimal solutions.

Nevertheless, every optimal value function for an infinite horizon, even those which are not transient, are convex.

7.4. Exact value iteration algorithms

7.4.1. Steps for the dynamic programming operator

The crucial step for applying the dynamic programming operator is building the set of vectors Θ_n from the set Θ_{n-1} . This operation is summed up (or hidden) in equation (7.16). This section details how this set can be built and gives an example

that explicits the crucial notions used in the various algorithms presented later in this section.

Let us start with a set Θ_{n-1} of vectors that represent the optimal value function for a horizon of size $n - 1$. Each vector θ_{n-1} dominates the others in a region of the belief space \mathcal{B} and also represents the best policy to choose in that region, an important fact to keep in mind. A vector θ_{n-1} is associated with a policy π_{n-1} with a horizon of $n - 1$. Figure 7.7 illustrates this on an example with 2 vectors $\theta_{n-1,0}$ and $\theta_{n-1,1}$. Regions of dominance of the vectors are represented by the gray bars along the x -axis.

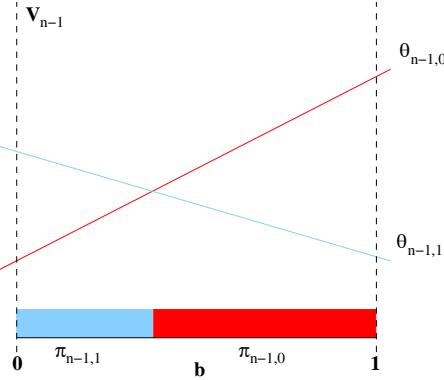


Figure 7.7. Value function at iteration $n - 1$. This POMDP with two states is an example to illustrate the construction of the set Θ_n . At iteration $n - 1$, the value function is represented by two vectors, each one is associated with a policy. Regions of dominance are represented by bars at the bottom of the plot. The policies associated with the regions are written under the bars

To understand how Θ_n is built, it is easier to decompose this process in small steps. Besides, this is also the approach taken by the two algorithms presented later in this section. Our first objective is to calculate the value $V_n^{a1,o1}$, that is, the value of the policy of horizon n starting with action $a1$ and then the optimal $n - 1$ -policy taken after having observed $o1$. This value function is represented by the vectors $\{\theta_n^{a1,o1}\}$ given by the equation (7.16) adapted to the present situation, that is,

$$\theta_n^{a1,o1}(b, s) = \frac{r(s, a1)}{|\Omega|} + \gamma \sum_{s' \in \mathcal{S}} p(s, a1, s') O(s', o1) \theta_{n-1}^{a1,o1}(b^{a1,o1}, s). \quad (7.17)$$

At most $|\Theta_{n-1}|$ vectors are needed to represent the value function. Figure 7.8 shows how the value function is transformed. It must be noted that this value function is a bit peculiar as it takes into account the probability of observing $o1$, leading to the term $\frac{r(s,a)}{|\Omega|}$ in the equation as we have made the choice of uniformly distributing the reward over the observations.

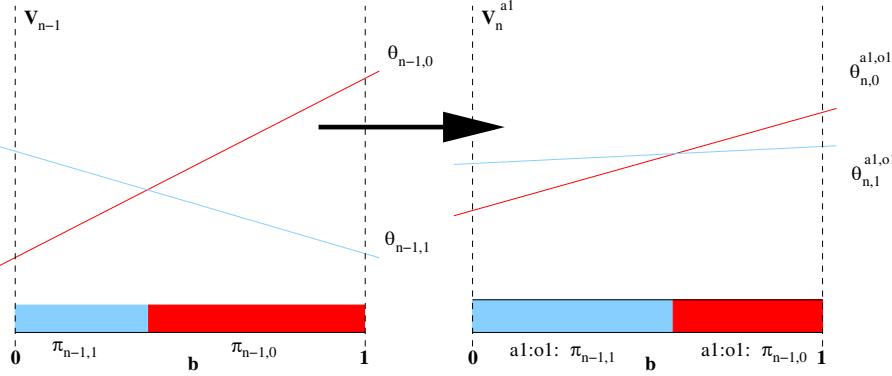


Figure 7.8. Value function at step n for action $a1$ and observation $o1$. Given an action $a1$ and an observation $o1$, the new value function is represented by, at most, the same number of vectors as Θ_{n-1} . These vectors define new dominance regions where the policies of horizon n all start by $a1: o1$ before using the best policy of horizon $n - 1$

Action $a1$ being still fixed, the value function for any possible observation o can be calculated. Then, it is possible to use these $|\Omega|$ value functions to calculate the value function V_n^{a1} received after executing $a1$ as the first action. It is also a PWLC function as it is the expected value of the value functions associated with the pairs $(a1, o)$. As the value function $V_n^{a1,o1}$ already takes into account the observation probability, it is quite straightforward to show that, for every belief state,

$$\theta_n^{a1}(b) = \sum_{o \in \Omega} \theta_n^{a1,o}(b). \quad (7.18)$$

This way, at most $|\Theta_{n-1}|^{|\Omega^{\text{succ}}(a1)|}$ vectors are generated. Each vector is linked to a policy beginning by the action $a1$ and where the observation received conditions what happens next. This is illustrated in Figure 7.9.

Value functions calculated for each action need to be combined in order to calculate the value function V_n for horizon n . For every belief state, the value function is represented by the “best” vector V_n^a and thus

$$\theta_n(b) = \max_{a \in \mathcal{A}} \theta_n^a(b). \quad (7.19)$$

At this step it becomes possible to prune many of the dominated vectors. Figure 7.10 illustrates this by combining value functions V_n^{a1} and V_n^{a2} and pruning dominated vectors (in dotted lines on the plot).

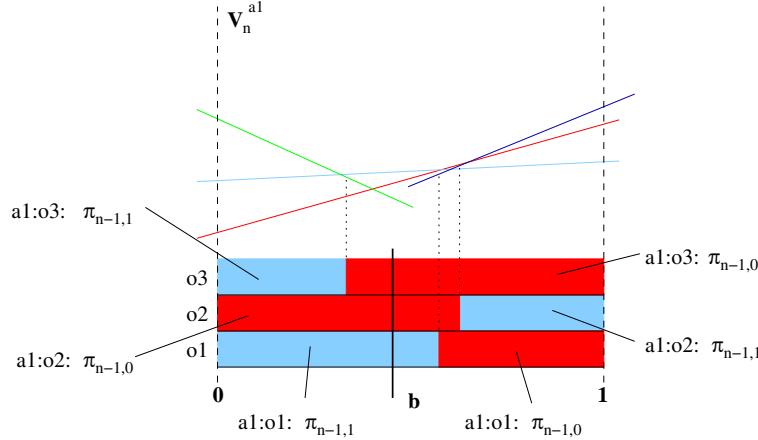


Figure 7.9. Value function at step n for $a1$. For a given action, the new value function is represented with vectors that are sums of the vectors $\theta_n^{a1,o}$. These vectors define new dominance regions where the policies of horizon n all start with action $a1$ before choosing the best policy of horizon $n - 1$ according to the received observation.
For example, for the belief state b of the figure, after action $a1$, policy $\pi_{n-1,1}$ will be chosen if $o = o1$ and policy $\pi_{n-1,0}$ otherwise

7.4.2. A parsimonious representation of V

This section introduces the concept of parsimonious representation of V . Then, some methods to calculate this kind of representation are presented, mainly by pruning dominated vectors.

7.4.2.1. Region

A set Θ defines a partition of the space of belief states \mathcal{B} . Each part of the partition is associated with a vector θ of Θ . For example, in Figure 7.7, the partition is made of 2 elements. The region $R(\theta, \Theta)$ associated with a vector is the part of \mathcal{B} where this vector dominates the others.

DEFINITION 7.5. If \mathcal{B} is the space of belief states and Θ represents a value function, the region $R(\theta, \Theta)$ associated with a vector θ of Θ is defined by

$$R(\theta, \Theta) = \{b \mid b \cdot \theta > b \cdot \theta', \theta' \in \Theta - \{\theta\}, b \in \mathcal{B}\}. \quad (7.20)$$

Due to the strict inequality, regions do not exactly define a partition of \mathcal{B} . Some points of \mathcal{B} do not belong to any regions, these are points where multiple vectors give the same value to the value function. These points are quite special and are potential problems, as detailed in section 7.4.2.5.

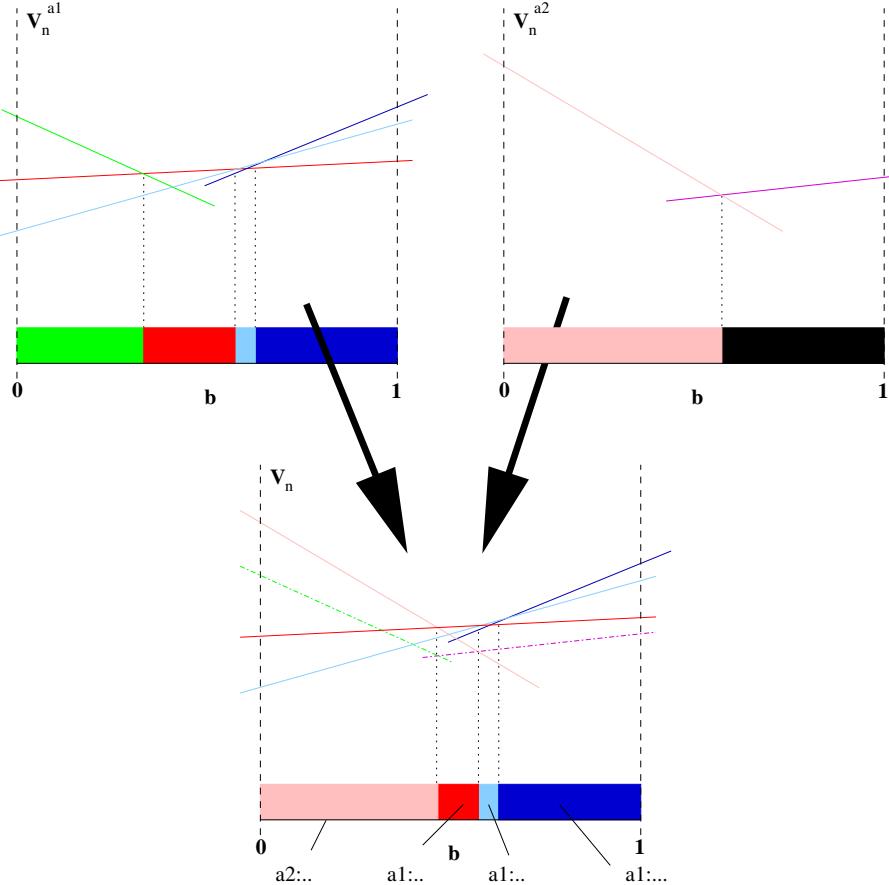


Figure 7.10. Value function at iteration n . To select the best starting action means to select the best vector for the starting belief state. On this plot, value functions for two actions (a_1 and a_2) are combined. As a consequence, some vectors are dominated (in dotted line). For each of the new dominance region, the first steps of the optimal policy of horizon n is depicted

The notion of region is very important. It is used by many algorithms for computing the value function. The `FindVecInRegion` algorithm (Algorithm 7.1) determines if the region of a vector is empty (it is then a dominated vector) and, if it is not, returns one vector from this region. A linear program must be solved to find the optimum of a constrained function which is defined by the `SetUpLinearProgram` algorithm (Algorithm 7.2).

Algorithm 7.1: FindVecInRegion(θ, Θ)

Input: A representation Θ , a vector $\theta \in \Theta$
Output: A point of the region or **null**

```

LP ← SetUpLinearProgram(  $\theta, \Theta$  )
SolveLinearProg( LP )
if NoSolution( LP ) then
     $\sqsubset$  return null
if val( LP )  $\leq 0$  then
     $\sqsubset$  return null
return Solution( LP )

```

Algorithm 7.2: SetUpLinearProgram(θ, Θ)

Input: A representation Θ , a vector $\theta \in \Theta$
Output: A Linear Program Problem
 solve

$$\max_{\mathbb{R}} \epsilon$$

with

$$x \cdot (\theta - \tilde{\theta}) \geq \epsilon, \forall \tilde{\theta} \in \Theta, \tilde{\theta} \neq \theta$$

$$x \in \Pi(\mathcal{S})$$

7.4.2.2. *Parsimonious representation*

In a set Θ , some vectors may not be useful to represent the value function. It is the case for the dominated vectors.

DEFINITION 7.6. Let Θ be a representation of a value function. A vector θ from Θ is dominated if $\forall b \in \mathcal{B}$ it holds that $b \cdot \theta \leq \max_{\theta' \in \Theta} b \cdot \theta'$.

If θ is dominated, it is trivial to show that Θ and $\Theta \setminus \{\theta\}$ represent the same value function: a dominated vector can be removed from Θ without harm. In fact, it has been shown (see, e.g. [LIT 96]) that a PWLC value function has a unique minimum representation with no dominated vectors. The representation is said to be parsimonious. In the example of Figure 7.11, vectors θ_2 and θ_4 are clearly not useful for representing V . In fact, the parsimonious representation is $\Theta = \{\theta_0, \theta_1, \theta_3\}$.

DEFINITION 7.7. The parsimonious representation Θ of a PWLC value function is such that, for any $\theta \in \Theta$, the region $R(\theta, \Theta)$ is not empty.

Now, the problem lies in building (or extracting) this parsimonious representation.

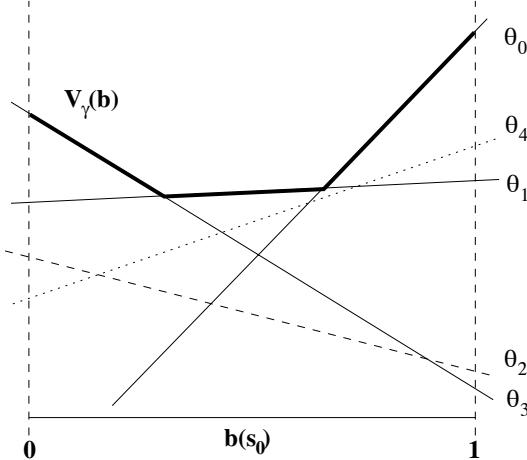


Figure 7.11. Parsimonious representation of V . All the vectors of Θ are not useful for representing V . The vector θ_2 , entirely dominated by θ_1 , will be pruned by the CheckDomination algorithm. But the vector θ_4 needs Pruning, a more complex algorithm, to be removed

7.4.2.3. Pruning of dominated vectors

The simplest method for pruning some dominated vectors is to look for vectors that are entirely dominated by a single other vector. For such a vector θ , there exists a single vector $\tilde{\theta} \in \Theta$ such that $\forall s' \in \mathcal{S}, \theta(s) \leq \tilde{\theta}(s')$. Although this method does not guarantee that all dominated vectors will be found and removed from Θ (see Figure 7.11 for some examples of dominated vectors that are not detected), it has a very good computation time.

The CheckDomination algorithm (Algorithm 7.3) details how entirely dominated vectors are detected and removed from a representation. It uses the RemoveElement algorithm that removes an element from a set.

7.4.2.4. Pruning

Another simple method for extracting a parsimonious representation is to check, for every vector, if its associated region is empty. The algorithm proposed by Monahan [MON 82] is based on this idea, but it is rather inefficient. A much more efficient pruning method has been developed by Lark and White [WHI 91]. It is detailed in the Pruning algorithm (Algorithm 7.4).

The main idea of Pruning is to incrementally build the parsimonious representation by keeping, at every moment, a subset $\tilde{\Theta} \subset \Theta$ of the parsimonious representation. For each new candidate vector θ from $\tilde{\Theta}$, checking if its associated

Algorithm 7.3: CheckDomination(Θ)

Input: A representation Θ
Output: A representation without any entirely dominated vector

```

if  $|\Theta| < 2$  then
    return  $\Theta$ 
 $\tilde{\Theta} \leftarrow \emptyset$ 
repeat
     $\theta \leftarrow \text{RemoveElement}(\Theta)$ 
    if  $\exists \theta' \in \tilde{\Theta}$  t.q.  $\theta' \geq \theta$  then
         $\tilde{\Theta} \leftarrow \{\theta' | \theta' \in \tilde{\Theta}, \theta \ngeq \theta'\}$ 
     $\tilde{\Theta} \leftarrow \tilde{\Theta} \cup \{\theta\}$ 
until  $\Theta = \emptyset$ 
return  $\tilde{\Theta}$ 
```

Algorithm 7.4: Pruning($\tilde{\Theta}$)

Input: A representation $\tilde{\Theta}$ of V
Output: A parsimonious representation $\hat{\Theta}$ of V

```

 $\hat{\Theta} \leftarrow \emptyset$ 
while  $\tilde{\Theta} \neq \emptyset$  do
     $\theta \leftarrow \text{RemoveElement}(\tilde{\Theta})$ 
     $b \leftarrow \text{FindVectInRegion}(\theta, \hat{\Theta})$ 
    if  $b \neq \text{null}$  then
         $\tilde{\Theta} \leftarrow \tilde{\Theta} \cup \{\theta\}$ 
         $\theta^* \leftarrow \text{BestVector}(\tilde{\Theta}, b)$ 
         $\tilde{\Theta} \leftarrow \tilde{\Theta} - \{\theta\}$ 
         $\hat{\Theta} \leftarrow \hat{\Theta} \cup \{\theta^*\}$ 
     $\Theta \leftarrow \hat{\Theta}$ 
return  $\Theta$ 
```

region *in* $\hat{\Theta}$ is not empty is very quick (because $\hat{\Theta}$ is small). But, a positive answer does not ensure that it is a dominating vector (as the definitive parsimonious representation is not known yet), it simply tells that $\hat{\Theta}$ is still not complete. In order to actually add another new dominating vector to $\hat{\Theta}$, we must use the procedure `BestVector` while θ is still in $\tilde{\Theta}$. The `BestVector` procedure looks for the best vector from $\hat{\Theta}$ for the belief state returned by `FindVectInRegion`, this vector will be added to $\hat{\Theta}$. The choice of the best vector sometimes needs some special attention, as detailed below.

7.4.2.5. Choice of a vector for a belief state

As detailed in section 7.4.1, the various formulations of equation (7.16) allow the computation of a dominating vector for any belief state b . But, when many vectors represent the value function for a belief state (as shown in Figure 7.12), which one will be the dominating one?

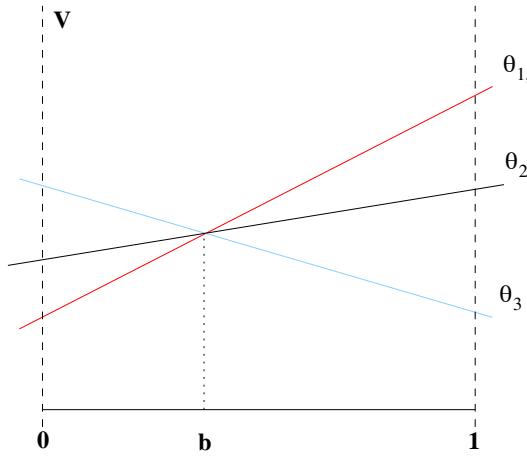


Figure 7.12. Dominating vector. Which vector dominates for the point b ?

One solution would be to calculate the validity of each vector in a neighborhood of b but this would require a large amount of computation. A simpler solution aims at lexicographically ordering the vectors. To do this, the states themselves must be ordered and an arbitrary ordering noted $s \prec s'$ is chosen on \mathcal{S} .

DEFINITION 7.8. The vector θ is lexicographically greater than θ' (denoted by $\theta \overset{L}{>} \theta'$) if there exists a state s such that $\theta(s) > \theta'(s)$ and $\theta(s') = \theta'(s')$ for any $s' \prec s$.

Theorem 7.2 (proven by Littman) shows that, in case of a choice, the maximum vector using the lexicographic order is indeed a vector that belongs to the parsimonious representation of the value function. The region of this vector will not be empty. This is the vector returned by the `BestVector` algorithm (Algorithm 7.6), using the `LexicographicMaximum` algorithm (Algorithm 7.5) that returns the lexicographic maximum of two vectors.

THEOREM 7.2 [LIT 96]. Let Θ be a value function, b a point of the belief space and Λ the set of vectors that give a maximum value to the value function in b ($\Lambda = \text{argmax}_{\theta \in \Theta} b \cdot \theta$). If there exists a $\lambda^* \in \Lambda$ such that $\lambda^* \overset{L}{>} \lambda$ for any other $\lambda \in \Lambda$, then $R(\lambda^*, \Theta)$ is not empty.

Algorithm 7.5: LexicographicMaximum($\theta, \tilde{\theta}$)

Input: Two vectors θ and $\tilde{\theta}$ from Θ
Output: The lexicographic maximum of the two vectors

```

for each  $s \in \mathcal{S}$  do
  if  $\theta(s) > \tilde{\theta}(s)$  then
    return  $\theta$ 
  if  $\theta(s) < \tilde{\theta}(s)$  then
    return  $\tilde{\theta}$ 
return  $\theta$ 
```

Algorithm 7.6: BestVector(Θ, b)

Input: A representation Θ , a belief state b
Output: The best vector of Θ for this state

```

 $v^* \leftarrow -\infty$ 
for each  $\theta \in \Theta$  do
   $v \leftarrow b.\theta$ 
  if  $v = v^*$  then
     $v^* \leftarrow \text{LexicographicMaximum}(\theta^*, \theta)$ 
  if  $v > v^*$  then
     $v^* \leftarrow v$ 
     $\theta^* \leftarrow \theta$ 
return  $\theta^*$ 
```

7.4.3. The WITNESS algorithm

The WITNESS algorithm was developed by Cassandra, Littman and Kaelbling [CAS 94] in 1994. Its optimality has then been formally proven [LIT 96]. For each action a of \mathcal{A} , the algorithm calculates a parsimonious representation of Θ_n^a from Θ_{n-1} by exploring a finite number of regions of \mathcal{B} . The cleverness of the algorithm lies in the way the regions are chosen. It is worth noting that other methods also rely on an exploration of regions but in a view to directly build Θ_n (see [SON 71, SMA 73, CHE 88]).

7.4.3.1. Neighborhood of a vector

The notion of *neighborhood* of a vector θ_n^a of V_n^a is at the heart of the WITNESS algorithm. It is by checking the neighbors of a vector that the algorithm detects if the current parsimonious representation of V_n^a is complete or not.

Equation (7.17) can be rewritten as

$$\theta_n^{a,o} = \frac{r(a)}{|\Omega|} + \gamma P^{a,o} \theta_{n-1}^{a,o}(b^{a,o}), \quad (7.21)$$

where $\theta_{n-1}^{a,o}(b^{a,o})$ is the best vector of Θ_{n-1} for the belief state $b^{a,o}$. Removing the link with a particular belief state, it is possible to build a family of vectors:

$$\tilde{\theta}^{a,o} = \frac{r(a)}{|\Omega|} + \gamma P^{a,o} \theta_{n-1}, \quad (7.22)$$

where θ_{n-1} is simply a vector of Θ_{n-1} . The set $\tilde{\Theta}_n^{a,o}$ is thus established, and combining these vectors leads to the set $\tilde{\Theta}^a$ that contains the parsimonious representation Θ_n^a of V_n^a . The set $|\tilde{\Theta}_n^a|$ is made of $|\Theta_{n-1}|^{|\Omega|}$ vectors. The neighborhood of the vector is then defined as follows.

DEFINITION 7.9. A vector ν of $\tilde{\Theta}_n^a$ is a neighbor of the vector $\theta_n^a = \sum_{o \in \Omega} \theta_n^{a,o}$ (that is also in Θ_n^a) if

$$\nu = \tilde{\theta}_n^{a,o'} + \sum_{o \neq o'} \theta_n^{a,o},$$

where $o' \in \Omega$, $\tilde{\theta}_n^{a,o'} \in \tilde{\Theta}_n^{a,o}$ and $\tilde{\theta}_n^{a,o'} \neq \theta_n^{a,o'}$.

A vector $\tilde{\theta}$ has $|\Omega|(|\Theta_{n-1}| - 1)$ neighbors, this set of neighbors is called $\mathcal{N}(\tilde{\theta})$. This importance of the set of neighbors comes from the following theorem that says that if there is a point of \mathcal{B} where a better vector exists then it is also true for one the neighbors of this vector (see [CAS 98]).

THEOREM 7.3. For any $\tilde{\theta}_n^a \in \tilde{\Theta}_n^a$, there exist $b \in \mathcal{B}$ and $\tilde{\theta}'_n^a \in \tilde{\Theta}_n^a$ such that $b \cdot \tilde{\theta}'_n^a > b \cdot \tilde{\theta}_n^a$ if and only if there exists a neighbor $\nu \in \mathcal{N}(\tilde{\theta}_n^a)$ such that $b \cdot \nu > b \cdot \tilde{\theta}_n^a$.

Proof. The proof will proceed in two steps.

– The first step aims at showing that $b \cdot \nu > b \cdot \tilde{\theta}_n^a \Rightarrow b \cdot \tilde{\theta}'_n^a > b \cdot \tilde{\theta}_n^a$. As $\nu \in \tilde{\Theta}_n^a$, it is trivial.

– Then, let us show that $b \cdot \tilde{\theta}'_n^a > b \cdot \tilde{\theta}_n^a \Rightarrow b \cdot \nu > b \cdot \tilde{\theta}_n^a$. The existence of ν will be proven by construction. We have

$$b \cdot \tilde{\theta}'_n^a > b \cdot \tilde{\theta}_n^a, \quad \text{therefore}$$

$$\sum_o b \cdot \tilde{\theta}'_n^{a,o} > \sum_o b \cdot \tilde{\theta}_n^{a,o}.$$

As the first sum is greater than the second one, there exists an observation o' such that

$$b \cdot \tilde{\theta}'_n^{a,o'} > b \cdot \tilde{\theta}_n^{a,o'}.$$

This fact allows us to build ν :

$$\begin{aligned} \sum_{o \neq o'} b \cdot \tilde{\theta}_n^{a,o} &= \sum_{o \neq o'} b \cdot \tilde{\theta}_n^{a,o} \\ b \cdot \tilde{\theta}'_n^{a,o'} + \sum_{o \neq o'} b \cdot \tilde{\theta}_n^{a,o} &> b \cdot \tilde{\theta}_n^{a,o'} + \sum_{o \neq o'} b \cdot \tilde{\theta}_n^{a,o} \\ b \cdot \left(\tilde{\theta}'_n^{a,o'} + \sum_{o \neq o'} \tilde{\theta}_n^{a,o} \right) &> b \cdot \tilde{\theta}_n^a. \end{aligned}$$

We need only to say that $\nu = \tilde{\theta}'_n^{a,o'} + \sum_{o \neq o'} \tilde{\theta}_n^{a,o}$ to end the proof of this second implication and then to prove the theorem. \square

7.4.3.2. The algorithm

The WITNESS algorithm incrementally builds a parsimonious representation Θ_n^a of a value function V_n^a for a fixed action a . As described in Algorithm 7.7, the set $\hat{\Theta}$ grows by storing the vectors of Θ in order to ensure that $\hat{V}(b) \leq V(b)$.

To do this, the algorithm first chooses a vector b of \mathcal{B} and looks for the best vector in the region of b and for all the neighbors of this best vector. The set of neighbors is called Υ and will be a kind of agenda as, one by one, the vectors v of the agenda are checked:

- either to be removed from the agenda if their region is empty (as v is then a dominated vector for V);
- or to add the best vector of the region of v to the vectors representing V and add all the neighbors of v to the agenda. It is also important to put v back in the agenda as v might prove to be useful later as the representation grows.

The validity of the algorithm is grounded on Theorem 7.3. To sum up, for any combination of vectors currently in $\hat{\Theta}$, if a vector of Θ_n^a would be better for a point b , then one of the neighbors v of the vector $\hat{\theta}$ of $\hat{\Theta}$ that is currently the best answer in b will also be a better answer for b . As the algorithm checks all the neighbors of the vectors of $\hat{\Theta}$, no vector from Θ_n^a is missed. The formal proof is a bit more complex.

The efficiency of the algorithm can be improved in several ways. In particular, [CAS 98] suggests choosing the initial vectors of $\hat{\Theta}$ with care, to avoid testing the same vector v from Υ and to check if the neighbor of a vector is useful before adding

Algorithm 7.7: Witness(Θ_{n-1}, a)

Input: A parsimonious representation Θ_{n-1} of V_{n-1}^* , an action a
Output: A parsimonious representation $V_n^{*,a}$

```

 $b \leftarrow$  a belief state of  $\mathcal{B}$ 
 $\hat{\Theta} \leftarrow \{\theta_n^a(b)\}$ 
 $\Upsilon \leftarrow \mathcal{N}(\theta_n^a(b))$ 
while  $\Upsilon \neq \emptyset$  do
     $v \leftarrow \text{RemoveElement}(\Upsilon)$ 
    if  $v \in \hat{\Theta}$  then
        |  $b \leftarrow \text{null}$ 
    otherwise
        |  $b \leftarrow \text{FindVectInRegion}(v, \hat{\Theta})$ 
        if  $b \neq \text{null}$  then
            |  $\hat{\Theta} \leftarrow \hat{\Theta} \cup \{\theta_n^a(b)\}$ 
            |  $\Upsilon \leftarrow \Upsilon \cup \{v\}$ 
    |
 $\Theta_n^a \leftarrow \hat{\Theta}$ 
return  $\Theta_n^a$ 

```

it to Υ . Still, the practical interest of this algorithm is limited because memory and computation considerations prevent us from applying this algorithm for more than a few iterations (usually, nothing more than a horizon of 4 or 5) for problems with only a handful of states.

7.4.4. Iterative pruning

The ITERATIVE PRUNING algorithm is a bit more efficient than the WITNESS algorithm. First, a very simple pruning that checks every vector is presented.

7.4.4.1. Complete enumeration

In order to find a parsimonious representation of Θ_n , one possibility is to enumerate all the vectors of this set and then to prune them. The method, proposed by Monahan [MON 82], is detailed now in order to ease the understanding of iterative pruning.

Let us write: $\overline{\Theta}_n^{a,o} = \left\{ \frac{r(a)}{|\Omega|} + \gamma \sum_{s' \in \mathcal{S}} p(s, a, s') O(s', o) \theta_{n-1}^{a,o}(b^{a,o}) \right\}_{(a,o)}$.

It is the set of all the possible vectors of $\Theta_n^{a,o}$. Combining these vectors in all possible ways (using equation (7.18)) gives the new set $\bar{\Theta}_n^a$ that contains Θ_n^a . This operation is called the *crossed sum* and is written as

$$\bar{\Theta}_n^a = \bigoplus_o \bar{\Theta}_n^{a,o}.$$

So, the complete set of vectors that generate V_n can be written as

$$\bar{\Theta}_n = \bigcup_a \bar{\Theta}_n^a.$$

Pruning this set leads to a parsimonious representation:

$$\Theta_n = \text{PRUNE} \left(\bigcup_a \bar{\Theta}_n^a \right).$$

The main drawback of this method is its complexity that is exponential in the size of Ω . From this came the idea to prune the set incrementally.

7.4.4.2. Incremental enumeration

Similarly to the WITNESS algorithm, the incremental pruning algorithm first looks for parsimonious representations of Θ_n^a before combining them in order to build a parsimonious representation of Θ_n . As $\Theta_n^a = \text{PRUNE}(\bar{\Theta}_n^a)$, we have

$$\Theta_n^a = \text{PRUNE} \left(\bigoplus_o \bar{\Theta}_n^{a,o} \right).$$

As the PRUNE procedure looks in fact for maximum vectors, we can show quite easily that they are indeed combinations of vectors that are also maximum, which leads to

$$\begin{aligned} \Theta_n^a &= \text{PRUNE} \left(\bigoplus_o \text{PRUNE} (\bar{\Theta}_n^{a,o}) \right) \\ &= \text{PRUNE} \left(\bigoplus_o \Theta_n^{a,o} \right). \end{aligned}$$

The PRUNE operator is kept outside the crossed summation because even if all maximum vectors are combinations of maximum vectors, some of these combinations are still useless (i.e. dominated).

Then we have

$$\begin{aligned}\Theta_n^a &= \text{PRUNE} \left(\bigoplus_o \Theta_n^{a,o} \right) \\ &= \text{PRUNE} \left(\Theta_n^{a,0} \oplus \Theta_n^{a,1} \oplus \Theta_n^{a,2} \oplus \dots \oplus \Theta_n^{a,|\Omega|-1} \right) \\ &= \text{PRUNE} \left(\dots \text{PRUNE} \left(\text{PRUNE} \left(\Theta_n^{a,0} \oplus \Theta_n^{a,1} \right) \oplus \Theta_n^{a,2} \right) \dots \oplus \Theta_n^{a,|\Omega|-1} \right).\end{aligned}$$

This equation sums up the incremental pruning algorithm. As shown in Figure 7.13, the central idea is to prune the combinations of the two sets $\Theta_n^{a,0}$ and $\Theta_n^{a,1}$ (seen as partitions of regions) in order to get an intermediate pruned set. The vectors of this new set are then combined with the vectors of $\Theta_n^{a,2}$, and so on.

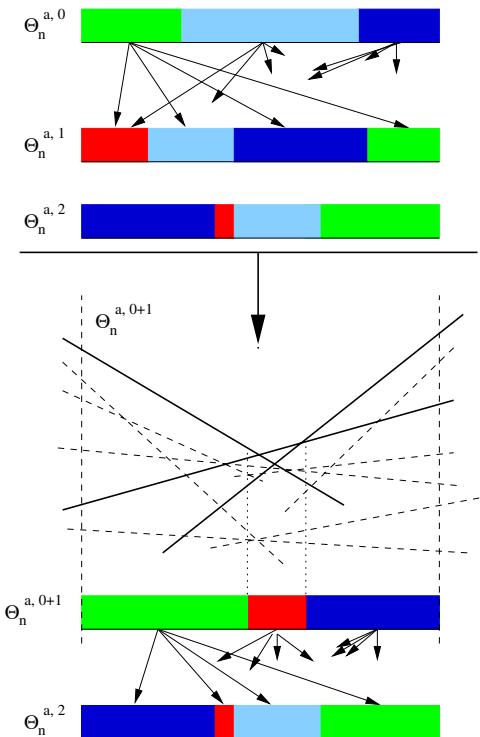


Figure 7.13. Incremental pruning of Θ_n^a . In order to build a parsimonious representation of Θ_n^a , “incremental pruning” starts from two sets $\Theta_n^{a,0}$ and $\Theta_n^{a,1}$. The set made of all the combinations of vectors from these two sets is then pruned to make the parsimonious set $\Theta_n^{a,0+1}$. Next, this new set is combined with $\Theta_n^{a,2}$ to form $\Theta_n^{a,0+1+2}$, and the process is iterated to get Θ_n^a

The method is detailed in Algorithm 7.8. The set Ψ is there to temporary store the vectors of $\Theta_n^{a,o}$ and the intermediate results from the pruning of the combinations of two vectors.

Algorithm 7.8: IncrementalPruning(Θ_{n-1} , a)

Input: A parsimonious representation Θ_{n-1} of V_{n-1}^* , an action a

Output: A parsimonious representation $V_n^{*,a}$

$\Psi \leftarrow \bigcup_o \{\Theta_n^{a,o}\}$

while $|\Psi| > 1$ **do**

$A \leftarrow \text{RemoveElement}(\Psi)$

$B \leftarrow \text{RemoveElement}(\Psi)$

$D \leftarrow \text{PRUNE}(A \oplus B)$

$\Psi \leftarrow \Psi \cup \{D\}$

return Ψ

7.5. Policy iteration algorithms

As for MDPs (see Algorithm 1.5), it is possible to look for an optimal solution to a POMDP directly in the space of policies, using a variant of policy iteration. The main difficulty lies in defining an adequate policy space to search.

For policies defined on the space of belief states ($\pi : \mathcal{B} \rightarrow \mathcal{A}$), the problem is then to find a global maximum in a continuous space. The algorithm proposed by Sondik [SON 78] is theoretically able to find an exact optimal solution for transient POMDPs and an ϵ -optimal solution otherwise. But the complexity of each iteration restricts its practical use to very simple problems.

An interesting alternative has been suggested by Hansen [HAN 98b]. This algorithm is based on the fact that policies for transient POMDPs can be represented by finite state automata (see Figure 7.3). The algorithm of Sondik, that needs to split \mathcal{B} into regions before building a finite state controller from the splitting, is simplified by the direct use of finite state controller. This is the main idea of Hansen's algorithm.

The heart of the algorithm is the update of the finite state controller. This step uses the dynamic programming operator to incrementally improve the controller and its value.

Sondik has proved that the value function of a finite state machine policy is piecewise linear [SON 78]. For such a controller δ , let us call V^δ its value function; it can be described by a set of vectors $\{c^i\}$ with exactly one vector for each node i of the controller. For each observation o , an action $a(i)$ and a transition to the node

$l(i, o)$ (or to the vector $\mathbf{c}^{l(i,o)}$) can be associated with each vector \mathbf{c}^i linked to the node i . For each node i of the controller and for each state s of the POMDP, the value function obeys

$$\mathbf{c}^i(s) = r(s, a(i)) + \gamma \sum_{s', o} \Pr(s' | s, a(i)) \Pr(o | s') \mathbf{c}^{l(i,o)}. \quad (7.23)$$

Using the dynamic programming operator (see section 7.4.1) on the vectors of V^δ leads to a new value function \hat{V}^δ . We can show that each vector $\hat{\mathbf{c}}^j$ of \hat{V}^δ is associated with an action $a(j)$ and, for each observation o , with a transition $\hat{l}(j, o)$ with a vector $\hat{\mathbf{c}}^{\hat{l}(j,o)}$ of V^δ . These vectors $\hat{\mathbf{c}}^j$ of \hat{V}^δ can be the same as some vectors of δ (same action and same transitions). They can also be new vectors that will change V^δ and thus alter the controller in the following way:

- if $\hat{\mathbf{c}}^j$ dominates a vector \mathbf{c}^i of V^δ , the action and the transitions of the node j are associated with the node i ;
- otherwise, the node j is added to δ .

δ must then be pruned of all the nodes that do not have any associated vector in \hat{V}^δ but that can be reached from another node of δ that has an associated vector of \hat{V}^δ . Eventually, a new controller $\hat{\delta}$ is built. This is the central notion of Hansen's algorithm (see Algorithm 7.9). In his PhD dissertation [HAN 98a], Hansen proves the theorem that ensures that each iteration improves the value of the policy, thus proving the convergence to an ϵ -optimal solution after a finite number of iterations (or, in the case of transient POMDPs, to the optimal solution).

THEOREM 7.4. *If a finite state controller δ is not optimal, one iteration of the PolicyIteration algorithm turns it into a controller $\hat{\delta}$ with a value function that is at least as good for all belief states and better for some belief states.*

7.6. Conclusion and perspectives

Partially observable Markov decision processes (POMDPs) can be used to model and control dynamic systems with uncertainty where the state is only partially known. The controller does not know the state of the process and can only have imperfect observations about this hidden state. Generally, the knowledge of the current observation is not sufficient to control a POMDP in an optimal way. An optimal policy requires far more information like, for example, the complete knowledge of the entire history (observations and actions) of the process. Thus, classical methods are grounded on belief states that are sufficient statistics of the process. In fact, methods derived from dynamic programming (value iteration and policy iteration) can be used with the belief states.

Algorithm 7.9: PolicyIteration(δ, ϵ)

Input: A finite state controller δ and a positive real ϵ
Output: A finite state controller $\hat{\delta}^*$ which is ϵ -optimal

```

repeat
    Calculate  $V^\delta$  from  $\delta$  by solving equations (7.23)
    Build  $\hat{V}^\delta \leftarrow \text{DynamicProgOperator}(V^\delta)$ 
     $\hat{\delta} \leftarrow \emptyset$ 
    for each  $\hat{c}^j \in \hat{V}^\delta$  do
        if there exists a node  $i$  of  $\delta$  associated with  $\hat{c}^j$  with identical action and
        links then
            | add  $i$  to  $\hat{\delta}$ 
        otherwise if there exists a node  $i$  such that  $\hat{c}^j$  dominates  $c^i$  then
            | add  $i$  to  $\hat{\delta}$ , with the action and the links of  $\hat{c}^j$ 
        otherwise
            | add a new node to  $\hat{\delta}$  with the actions and links of  $\hat{c}^j$ 
    Add to  $\hat{\delta}$  all the other nodes of  $\delta$  that are reachable from  $\hat{\delta}$ 
     $\delta \leftarrow \hat{\delta}$ 
until  $\|\hat{V}^\delta - V^\delta\| \leq \epsilon(1 - \gamma)/\gamma$ 
return  $\hat{\delta}$ 

```

In practice, classical algorithms like WITNESS or ITERATIVE PRUNING are based on the fact that the value function is piecewise linear convex, with efficient and compact representations. Nevertheless, these exact algorithms can only be applied to toy problems (no more than a few states) because of the exponential number of elements needed to represent the value function. It is also the case with other classical algorithms that were not detailed here like, for example, the algorithms of Monahan [MON 82] or Cheng [CHE 88]. One alternative is to factorize states and observations, as done by [GUE 01, HAN 00, POU 05]. It may also be the case that recent works on using linear programming to represent a policy by its set of reachable trajectories lead to more efficient algorithms but, for now, advances in terms of complexity are relatively small (see [KOL 94, ARA 07]).

These theoretical works, with limited practical impact, have yet inspired some algorithms that look for approximate solutions to POMDPs. One possibility is to build an optimal value function for only a subset of points of the space of belief states and hope that this approximation will be close to the real optimal solution [PIN 03, SPA 05]. Some other works with a “forward search” form a unique belief state in order to only calculate the value function on reachable belief states (see, e.g. [BON 00]).

Some works combine dynamic programming with heuristic search in order to strongly reduce, in each iteration, the size of the set of belief points for which a value function is calculated. As a consequence, complex problems can be tackled [SEU 07].

Another approach explored in the field of POMDPs relies on “online” methods of planning. As described in Chapter 6, the idea is to take advantage of the current situation of the agent to calculate the optimal actions to take only in situations (or belief states) that the agent is likely to meet in the future. Then, a large part of the state-action space does not need to be explored and computation resources are focused on “useful” states. An example of such method for POMDPs can be found in [ROS 08].

The most crucial problem of POMDPs is the question of learning. Indirect methods (i.e. learning a model first and then computing a plan) are not very efficient because of the difficulty of learning the underlying structure of a POMDP. At this point, we can only use methods that look for approximate solutions. The work of [MCC 95, DUT 00] is based on the hypothesis that the POMDP is in fact a MDP of order k . Other works use gradient ascent methods to look for a local optimum in the space of parameterized policies [BAX 00] (see Chapter 5).

Finally, we must mention the recent works on *Predictive State Representations* (PSRs). [LIT 02] has shown that a POMDP can be efficiently represented using a PSR that represents in fact probabilities for some trajectories to be reached in the future of the process. Some learning algorithms have been developed in order to discover these PSRs and to learn their probabilities [SIN 03, ABE 07].

7.7. Bibliography

- [ABE 07] ABERDEEN D., BUFFET O. and THOMAS O., “Policy-gradients for PSRs and POMDPs”, *Proceedings of the 11th International Conference on Artificial Intelligence and Statistics (AISTATS’07)*, San Juan, Puerto Rico, 2007.
- [ARA 07] ARAS R., DUTECH A. and CHARPILLETT F., “Mixed integer linear programming for exact finite-horizon planning in decentralized POMDPs”, *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS’07)*, Providence, Rhode Island, pp. 18–25, 2007.
- [AST 65] ASTRÖM K., “Optimal control of Markov decision processes with incomplete state estimation”, *Journal of Mathematical Analysis and Applications*, vol. 10, pp. 174–205, 1965.
- [BAX 00] BAXTER J. and BARTLETT P., “Reinforcement learning in POMDP’s via direct gradient ascent”, *In Proceedings of the 17th International Conference on Machine Learning (ICML’00)*, Morgan Kaufmann, San Francisco, CA, pp. 41–48, 2000.
- [BER 95] BERTSEKAS D., *Dynamic Programming and Optimal Control*, Athena Scientific, Belmont, MA, 1995.

- [BON 00] BONET B. and GEFFNER H., “Planning with incomplete information as heuristic search in belief space”, *Proceedings of the 5th International Conference on Artificial Intelligence Planning and Scheduling (AIPS'00)*, Breckenridge, CO, pp. 52–61, 2000.
- [CAS 94] CASSANDRA A., KAEHLING L. and LITTMAN M., “Acting optimally in partially observable stochastic domains”, *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI'94)*, Seattle, WA, pp. 1023–1028, 1994.
- [CAS 98] CASSANDRA A., Exact and approximate algorithms for partially observable markov decision processes, PhD thesis, Brown University, Providence, RI, 1998.
- [CHE 88] CHENG H.-T., Algorithms for partially observable Markov decision processes, PhD thesis, University of British Columbia, Canada, 1988.
- [DUT 00] DUTECH A., “Solving POMDP using selected past-events”, *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI'00)*, Berlin, Germany, 2000.
- [GUE 01] GUESTRIN C., KOLLER D. and PARR R., “Solving factored POMDPs with linear value functions”, *Proceedings of the IJCAI-01 Workshop on Planning under Uncertainty and Incomplete Information*, Seattle, WA, pp. 67–75, 2001.
- [HAN 98a] HANSEN E., Finite-memory control of partially observable systems, PhD thesis, Department of Computer Science, University of Massachusetts at Amherst, 1998.
- [HAN 98b] HANSEN E., “An improved policy iteration algorithm for partially observable MDPs”, *Advances in Neural Information Processing Systems 10 (NIPS'97)*, Denver, CO, 1015–1021, 1998.
- [HAN 00] HANSEN E. and FENG Z., “Dynamic programming for POMDPs using a factored state representation”, *Proceedings of the 5th International Conference on Artificial Intelligence Planning and Scheduling (AIPS'00)*, Breckenridge, CO, pp. 130–139, 2000.
- [JAA 95] JAAKKOLA T., SINGH S. and JORDAN M., “Reinforcement learning algorithm for partially observable Markov decision problems”, *Advances in Neural Information Processing Systems 7 (NIPS'94)*, MIT Press, Cambridge, MA, pp. 345–352, 1995.
- [KOL 94] KOLLER D., MEGIDDO N. and VON STENGEL B., “Fast algorithms for finding randomized strategies in game trees”, *Proceedings of the 26th ACM Symposium on Theory of Computing (STOC '94)*, Montreal, Canada, pp. 750–759, 1994.
- [LIT 95] LITTMAN M., CASSANDRA A. and KAEHLING L., Efficient dynamic programming updates in partially observable Markov decision processes, Report no. CS-95-19, Brown University, 1995.
- [LIT 96] LITTMAN M. L., Algorithms for sequential decision making, PhD thesis, Computer Science Department, Brown University, 1996.
- [LIT 02] LITTMAN M., SUTTON R. and SINGH S., “Predictive representations of state”, *Advances in Neural Information Processing Systems 14 (NIPS'01)*, MIT Press, Cambridge, MA, pp. 1555–1561, 2002.
- [MCC 95] MCCALLUM A., Reinforcement learning with selective perception and hidden state, PhD thesis, Department of Computer Science, University of Rochester, Rochester, NY, 1995.

- [MON 82] MONAHAN G. E., “A survey of partially observable Markov decision processes: theory, models and algorithms”, *Management Science*, vol. 28, no. 1, pp. 1–16, 1982.
- [PAP 87] PAPADIMITRIOU C. H. and TSITSIKLIS J. N., “The complexity of Markov decision processes”, *Mathematics of Operations Research*, vol. 12, no. 3, pp. 441–450, 1987.
- [PIN 03] PINEAU J., GORDON G. and THRUN S., “Point-based value iteration: An anytime algorithm for POMDPs”, *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*, Acapulco, Mexico, pp. 1025–1032, 2003.
- [POU 05] POUPART P., Exploiting structure to efficiently solve large scale partially observable Markov decision processes, PhD thesis, University of Toronto, 2005.
- [ROS 08] ROSS S., PINEAU J., PAQUET S. and CHAIB-DRAA B., “Online planning algorithms for POMDPs”, *Journal of Artificial Intelligence Research*, vol. 32, no. 1, pp. 663–704, 2008.
- [SEU 07] SEUKEN S. and ZILBERSTEIN S., “Memory-bounded dynamic programming for DEC-POMDPs”, *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, Hyderabad, India, pp. 2009–2015, 2007.
- [SIN 94] SINGH S., JAAKKOLA T. and JORDAN M., “Learning without state estimation in partially observable Markovian decision processes”, *Proceedings of the 11th International Conference on Machine Learning (ICML'94)*, Morgan Kaufmann, San Francisco, CA, pp. 284–292, 1994.
- [SIN 03] SINGH S., LITTMAN M., JONG N., PARDOE D. and STONE P., “Learning predictive state representations”, *Proceedings of the 20th International Conference of Machine Learning (ICML'03)*, Washington DC, pp. 712–719, 2003.
- [SMA 73] SMALLWOOD R. D. and SONDIK E. J., “The optimal control of partially observable Markov processes over a finite horizon”, *Operations Research*, vol. 21, pp. 1071–1088, 1973.
- [SON 71] SONDIK E., The optimal control of partially observable Markov decision processes, PhD thesis, Stanford University, California, 1971.
- [SON 78] SONDIK E., “The optimal control of partially observable Markov processes over the infinite horizon: discounted costs”, *Operations Research*, vol. 26, no. 2, pp. 282–304, 1978.
- [SPA 05] SPAAN M. and VLASSIS N., “Perseus: randomized point-based value iteration for POMDPs”, *Journal of Artificial Intelligence Research*, vol. 24, pp. 195–220, 2005.
- [WHI 91] WHITE C. C., “Partially observed Markov decision processes: a survey”, *Annals of Operational Research*, vol. 32, pp. 215–230, 1991.
- [ZAN 96] ZANG N. and LIO W., Planning in stochastic domains: problem characteristics and approximation, Report no. HKUST-CS96-31, Honk-Kong University of Science and Technology, 1996.

Chapter 8

Stochastic Games

8.1. Introduction

Game theory [AUM 02] is a formal model for studying the interactions among agents, assuming that such interactions can span from cooperation to conflict. Originally conceived as a mathematical tool for economic sciences, Game theory demonstrated its utility through the years in logic and in set theory [GRÄ 02, MAR 75], in biology and in evolution theory [SMI 02], in computer science [PAP 95, PAR 02, GIE 06], in conflict analysis [MYE 97], etc.

Interaction is the corner stone of the studies underlying game theory. Modeling interactions first requires extending the notion of game theory to *dynamic games*, generally used to represent the competition among processes that evolve over time. Stochastic transitions are then used to formalize uncertainty. *Stochastic (or Markov) games* are dynamic games with stochastic transitions. Nowadays they form a rich and mature mathematical theory, used in many applications like economy, biology and other domains with similar properties, such as evolution and populations, queues, telecommunications, model checking, etc. Recently, stochastic games have taken increasing importance in computer science, in particular through multi-agent systems for decision-making, automated planning and learning in environments where several autonomous agents operate [STO 00].

In this chapter, we consider stochastic games as the most generic multi-agent model. We present in detail several algorithms whose purpose is to solve problems

Chapter written by Andriy BURKOV, Laëtitia MATIGNON and Brahim CHAIB-DRAA.

which can be modeled as stochastic games, even if most of these algorithms can also solve simpler problems.

The remainder of this chapter is organized as follows. Section 8.2 presents the main concepts of classical game theory, such as games in strategic form, dynamic games, Nash equilibria and others. In section 8.3, stochastic games (also called “Markov games”) are defined, and various resolution algorithms are presented and discussed. Section 8.4 concludes this chapter.

8.2. Background on game theory

8.2.1. Some basic definitions

A poker game, the formation of a team or a negotiation between agents organizing a meeting are all different games following specific rules. In these games, none of the participants can fully control its fate; the participants are then said to be in a situation of strategic interaction [THI 04]. Game theory aims at formally studying this type of games where the fate of each agent taking part in the game depends not only on its own decisions, but also on the decisions made by the other participating agents. From this moment, the “best” choice for an agent generally depends on what other agents do.

Agents participating in a game are called players. Therefore a player is an agent which could represent a company, a robot, a consumer, etc. A player acts for its own benefit, following the principle of rationality, i.e. it aims at acting as well as possible according to its objectives, for example, by maximizing a certain performance measure estimating the agent’s success, as explained by Russell and Norvig [RUS 95]. Thus, each agent tries to make the “best decisions” for itself and with no “sacrifice” for other agents. Of course, this is not valid anymore if we consider teams of agents where participants follow a common objective.

In game theory, it is important to keep in mind that the agents participating in a game (we can interchangeably call these agents players or agents from now on) have to choose their own actions, taking into account the actions of other agents. They have to reason about others and have an idea – as precise as possible – about the possible behaviors of other players. To this end, game theory admits: (i) that each player tries hard to make the best decisions for itself and assumes that the others do the same; (ii) that the preceding fact, that is (i), is a knowledge common to all players.

8.2.1.1. Criteria distinguishing different forms of games

One-player games (skill games, brain teasers, enigmas, etc.) often reduce to classical optimization problems. For example, a planning problem in a large search space for brain teasers, or a planning problem under uncertainty (MDP) for skill

games. Here, we are therefore interested in games with multiple players and, most of the time, with *complete information*. The latter means that each player knows the rules of the game (including the moves that can be taken by itself and by the other players depending on the time in the game) as well as the objectives of all players. Otherwise, when the game is one of *incomplete information*, we have to act cautiously, or to try to understand the objectives of the other players.

With two players or more, the word “game” can first remind one of chess or checkers, i.e. *sequential* games – because the players play in turns – with *perfect information* – since one acts with full knowledge of the situation in the game.

A different situation is that of classical card games, where players also act in turns, but where the game is often with *imperfect information*: we do not know which cards the other players have or how they are ordered in the deck. Another difference here is that the notion of chance appears: the cards must have been shuffled so that nobody knows how they are being distributed. A game can also be of imperfect information if a player does not observe all the moves made by the other players, for example, which cards have moved from one hand to another or have been put in the stack.

In some other games, several players (if not all of them) can act simultaneously. The game is then called *simultaneous*. This is the case, for example, in “diplomacy”, a strategy game in which the next moves of each player’s troops are, on each turn: (1) discussed among players (who can lie), (2) written on a piece of paper and (3) performed simultaneously (this is where the players discover who lied to who). This is also the case in the simpler case of a penalty in soccer: the shooter and the goalkeeper often have to simultaneously decide about their moves, hoping to guess which side their opponent has chosen.

Most games can be reduced to *zero-sum* games, that is, games in which the players’ gains always sum to zero. In a two-player game, this simply means that, when one player wins, the other player loses.

As written earlier, game theory is often studied in the framework of economic problems, where several agents act in the same environment – and therefore “interact” – each with its own objectives. These objectives are not necessarily in complete opposition. There may be situations leading to compromises or even cooperation. A delicate problem is, for example, that of public transportation: each player has to choose between using the bus or its own car, the traveling time of a player (which each player tries to minimize) depending on one’s own mode of transportation (car>bus) and on the cluttering of the roads (more cars make more cluttering). The result is that the ideal would be for all players to take the bus, although each player tends to be selfish and prefer its car.

Finally, let us mention a particular problem, that of “cooperative games”, which focuses on the formation of coalitions of players who decide to share their goals while facing other players or coalitions. We will not go further into this topic, but it is at the heart of a lot of research work in the field of multi-agent systems.

In this chapter, we are mainly interested in games with complete information, this section starting with the presentation of simple games: static games played in a single turn.

8.2.2. Static games of complete information

Numerous game theoretic problems can be formalized and studied as static games (also called simultaneous games). These are chance games played in a single turn during which all players simultaneously decide on their only move.

Two coaches of soccer teams face a very similar situation when their teams are going to play against each other. They have to simultaneously choose (before the match): the players who will participate, their respective roles on the field and other tactical aspects that will remain fixed during the game.

8.2.2.1. Games in strategic form, pure strategy, mixed strategy

A game G in strategic form (or normal form game) is a tuple $\langle Ag, \{A_i : i = 1, \dots, |Ag|\}, \{R_i : i = 1, \dots, |Ag|\} \rangle$, where [THI 04]:

- $Ag = 1, \dots, |Ag|$ is the finite set of players. To avoid confusions, a player is denoted by i . Of course $i \in Ag$;

- a_i is an action (or pure strategy) of player i . Such a pure strategy describes in detail how the player i acts. The set A_i contains the pure strategies available to player i . Of course, $a_i \in A_i$. A game in strategic form is finite if the set of actions (pure strategies) of each player, A_i , is finite.

Besides, all players act simultaneously (or at least without knowing the strategy chosen by the other players);

- from this, $\mathbf{a} = (a_1, \dots, a_i, \dots, a_{|Ag|}) \in A_1 \times \dots \times A_i \times \dots \times A_{|Ag|} \equiv \mathbf{A}$ is an outcome of the game, that is, a combination of strategies where a_i is agent i 's strategy. In the remainder of this chapter, $\mathbf{a}_{-i} \in \mathbf{A}_{-i}$ denotes the combination of strategies chosen by all players except for player i ;

- $R_i(\mathbf{a}) \in \mathbb{R}$ is player i 's reward function. As can be observed, this reward function depends not only on its strategy a_i , but also on the other players' strategies \mathbf{a}_{-i} . Of course, player i strictly prefers an outcome \mathbf{a} over an outcome \mathbf{a}' if $R_i(\mathbf{a}) > R_i(\mathbf{a}')$. In the case where $R_i(\mathbf{a}) = R_i(\mathbf{a}')$, i is said to be indifferent between the outcomes \mathbf{a} and \mathbf{a}' ;

- each player knows the sets of strategies and the payoff functions of all other players.

The latter hypothesis characterizes the game as being of *complete information*. On the contrary, a game is said to be of *incomplete information* if the players know some elements of the game at best through probabilities.

Figure 8.1 represents as a matrix an example of a game in strategic form with two players, COMPANY1 and COMPANY2, having at their disposal the actions *produce* and *do not produce*. By convention the payoffs are written as (x, y) for a combination of actions $(action_{Company1}, action_{Company2})$, where x is the payoff for the row player (here COMPANY1) and y is the payoff for the column player (here COMPANY2).

		COMPANY2	
		Produce	Do not produce
COMPANY1	Produce	-3, -2	10, 0
	Do not produce	0, 8	0, 0

Figure 8.1. Example of game in strategic form. The values in each cell are the payoffs (gains in terms of utility) of each player for each combination of actions jointly performed by the players

A *pure strategy* reflects an action or a sequence of actions chosen by each player in a deterministic manner (in contraposition to stochastic). In some cases, it is preferable to resort to a *mixed strategy* defined as a probability distribution¹ over the set of pure strategies, that is, $\pi_i \in \Delta A_i$. In accordance with this notation, we will also denote by $\pi_i^{a_i}$ the probability for agent i to play action $a_i \in A_i$ and $\pi_{-i} \in \Delta \mathbf{A}_{-i}$ the joint strategy of other agents with respect to agent i .

Each player searches for a strategy that will allow it to maximize its expected payoff given the strategies followed by the other players. The term “expected payoff” (or, also, “expected utility”) of a player denotes the average reward that it expects to receive given its mixed strategy and the other players’ strategies. Let us denote by u_i the expected payoff of agent i . Then, the expected payoff for player i of the joint policy (π_i, π_{-i}) is

$$\begin{aligned} u_i^{(\pi_i, \pi_{-i})} &= E_{\mathbf{a} \in \mathbf{A}}^{(\pi_i, \pi_{-i})} R_i(\mathbf{a}) \\ &= \sum_{a_i \in A_i} \sum_{\mathbf{a}_{-i} \in \mathbf{A}_{-i}} R_i(a_i, \mathbf{a}_{-i}) \pi_i^{a_i} \pi_{-i}^{\mathbf{a}_{-i}}. \end{aligned} \quad (8.1)$$

1. We denote by ΔE the set of probability distributions over a set E .

In the case of a two-player game, we can use matrix notations. R_1 and R_2 being matrices, and π_1 and π_2 being vectors, this leads to writing

$$u_i^{(\pi_1, \pi_2)} = \pi_1^\top R_i \pi_2, \quad \forall i \in \{1, 2\}.$$

8.2.2.2. Zero-sum game and minimax

A game is called zero-sum if, for all $\mathbf{a} \in \mathbf{A}$, $\sum_i R_i(\mathbf{a}) = 0$. Here, we will focus on the case of two players (denoted by 1 and 2). The gains of one of them are then the losses of the other, and the players are therefore opponents “in the strict sense”. We can, for example, model a robust decision-making problem as a zero-sum game, the objective being to find the safest decisions when facing an uncertain environment (see section 10.3). This particular kind of game can be approximated (but not necessarily solved) using the *minimax* (or *maximin*) principle.

Minimax is a technique to search for a solution in zero-sum games. The Minimax algorithm has been developed by John von Neumann in 1928 [NEU 28]. It applies to all two-player zero-sum games. The agents are assigned to one of two roles: either MAX or MIN. The MAX player is supposed to maximize its payoff, while the MIN player is supposed to minimize MAX’s payoff (which is equivalent to maximizing its own payoff).

To illustrate this, let us assume that player i is the one seeking to maximize. It has at its disposal m strategies a_{ik} with $k = 1, 2, \dots, m$. Player j is the one seeking to minimize. It has at its disposal n strategies $a_{jk'}$ with $k' = 1, 2, \dots, n$. The set of all possible payoffs the player i can receive is represented by a matrix R_i of dimension $m \times n$ with entries $R_i(k, k')$. From this, agent i selects the rows in R_i , while agent j selects the columns. In this case, if player i chooses strategy k while j chooses strategy k' , then j has to pay i the payoff $R_i(k, k')$. We will note that negative payments are allowed because we are in a zero-sum game. We could also say that i receives the quantity $R_i(k, k')$ while j receives the quantity $-R_i(k, k')$.

Let us now consider the example shown in Figure 8.2. Of course, such a game can be represented simply by the matrix in Figure 8.3.

		MIN		
		3, -3	1, -1	8, -8
MAX	3, -3	1, -1	8, -8	
	4, -4	10, -10	0, 0	

Figure 8.2. Zero-sum game. In the matrix, values of the form \cdot, \cdot represent, respectively, the utilities of the MAX and MIN agents for each of the joint actions

	MIN						
MAX	<table border="1" style="border-collapse: collapse; width: 100%; height: 100%;"> <tr> <td style="width: 33.33%;">3</td> <td style="width: 33.33%;">1</td> <td style="width: 33.33%;">8</td> </tr> <tr> <td>4</td> <td>10</td> <td>0</td> </tr> </table>	3	1	8	4	10	0
3	1	8					
4	10	0					

Figure 8.3. Another matrix representation of a zero-sum game. The values in the matrix only represent the utilities of the MAX agent

In this game, the problem is to know which option the rational agent will choose. To that end, we can consider the *security levels* of each agent [HAU 98]. It is indeed easy to see that, if MAX chooses the first row, whatever MIN does, it will have at least a payoff of 1. By choosing the second row, it risks making a null payoff. Similarly, by choosing the first column, MIN will not have to pay more than 4, while if it chooses the second or third column, it risks losing, respectively, 10 or 8. We then say that the first security level of MAX is 1 and that it is guaranteed by the choice of the first row, while the security level of MIN is 4 and it is guaranteed by the choice of the first column. We will note that $\max_k \min_{k'} R_1(k, k') = 1$ and $\min_{k'} \max_k R_1(k, k') = 4$. This shows that the strategy that guarantees to the MAX agent its security level is the *maximin strategy*. Symmetrically, the strategy that guarantees to the MIN agent its security level is the *minimax strategy*.

PROPOSITION 8.1 [HAU 98]. *In a matrix representing a two-player zero-sum game, we have the following inequality:*

$$\max_k \min_{k'} R_1(k, k') \leq \min_{k'} \max_k R_1(k, k').$$

The maximin (or minimax) solution is acceptable (optimal) if the players play in turns. For example, player i starts by choosing an action, then player j makes its choice depending on the action played by j . Nevertheless, if both players have to play simultaneously, a thorough study of the previous game, illustrated in Figure 8.2, would show that the maximin and minimax strategies are not satisfactory solutions for this game. In some cases, however, the game can converge to a stable solution. Let us consider another example (Figure 8.4) taken from [HAU 98].

	MIN									
MAX	<table border="1" style="border-collapse: collapse; width: 100%; height: 100%;"> <tr> <td style="width: 33.33%;">10</td> <td style="width: 33.33%;">-15</td> <td style="width: 33.33%;">20</td> </tr> <tr> <td>20</td> <td>-30</td> <td>40</td> </tr> <tr> <td>30</td> <td>-45</td> <td>60</td> </tr> </table>	10	-15	20	20	-30	40	30	-45	60
10	-15	20								
20	-30	40								
30	-45	60								

Figure 8.4. A game where Maximin = Minimax

It is easy to see that

$$\begin{aligned}\max\{-15, -30, -45\} &= -15, \\ \min\{30, -15, 60\} &= -15.\end{aligned}$$

Thus, the pair of payoffs corresponding to the maximin and minimax strategies is given by $R_i(k, k') = (-15, 15)$ and corresponds to $(k, k') = (1, 2)$, that is, to the first row and to the second column.

If in the matrix of a zero-sum game there is a pair (k^*, k'^*) such that

$$\forall k, k', \quad R_1(k, k'^*) \leq R_1(k^*, k'^*) \leq R_1(k^*, k'),$$

then we say that the pair (k^*, k'^*) is a saddle point.

PROPOSITION 8.2 [HAU 98]. *If, in the matrix of a zero-sum game, we have*

$$\max_k \min_{k'} R_1(k, k') = \min_{k'} \max_k R_1(k, k') = v,$$

then the game admits a saddle point in pure strategies.

If (k^*, k'^*) is a saddle point for a game's matrix, then the MAX and MIN players cannot improve their payoffs by unilaterally deviating from k^* and k'^* , respectively. We then say that (k^*, k'^*) is an *equilibrium* because all the players had better stick to it.

Even if there does not always exist an equilibrium in pure strategies in a two-player zero-sum game, there does always exist one in mixed strategies (called mixed minimax equilibrium). Indeed, if we allow for mixed strategies $\pi_i \in \Delta A_i$, we have

$$\max_{\pi_1 \in \Delta A_1} \min_{\pi_2 \in \Delta A_2} \pi_1^\top R_1 \pi_2 = \min_{\pi_2 \in \Delta A_2} \max_{\pi_1 \in \Delta A_1} \pi_1^\top R_1 \pi_2.$$

We can in fact obtain the strategies for both players by solving the two following simplest problems (we denote by a_{ik} the pure strategy k of player i):

$$\operatorname{argmax}_{\pi_1 \in \Delta A_1} \min_{k'=1}^{|A_2|} \pi_1^\top R_1 a_2 k' = ?$$

$$\operatorname{argmin}_{\pi_2 \in \Delta A_2} \max_{k=1}^{|A_1|} a_1 k^\top R_1 \pi_2 = ?$$

8.2.2.3. Equilibrium in dominating strategies

The minimax is a classical approach for two-player zero-sum games. In other situations, different algorithms have to be employed. This problem can be illustrated

by the famous prisoner's dilemma. It reads as follows: two suspects (SUSPECT1 and SUSPECT2) are separately questioned by a judge about a serious offense. The judge does not have at his disposal a sufficient proof to sentence them and the defection (confession) of one of them is necessary. The judge therefore offers each defendant freedom if he confesses. On the other hand, if he denies and the other does confess, the former gets a sentence of 15 years. If both defect, they can expect to benefit from extenuating circumstances and get a sentence of 8 years each. Then, if both cooperate, they will be condemned for minor offenses to 1 year of prison each. Let us denote the action D as "defect" and C as "cooperate". The corresponding payoff matrix for both players appears in Figure 8.5.

		SUSPECT2	
		D	C
SUSPECT1	D	-8, -8 0, -15	
	C	-15, 0 -1, -1	

Figure 8.5. Payoff matrix of both players in the prisoner's dilemma

We are therefore conducted to ask the question: how should both suspect behave, assuming they are rational? We can notice that to *defect* is a strategy that leads to a smaller sentence than the *cooperate* strategy, whatever the choice of the other player. As a consequence, each suspect had better opt for this strategy in a view to reduce its sentence. According to the payoff matrix, each then gets an eight-year jail sentence, what is a relatively heavy punishment. This strategy bets on the fact that it is chosen because it gives a higher payoff than the other strategy without having to care about the other player's action. In game theory, such a strategy is called a *dominating strategy*.

DEFINITION 8.1. In a game in strategic form, a strategy $a_i \in A_i$ is said to be *dominating* for player i if, for all $\hat{a}_i \in A_i$ and $\hat{a}_i \neq a_i$, we have

$$R_i(a_i, \mathbf{a}_{-i}) \geq R_i(\hat{a}_i, \mathbf{a}_{-i}), \quad \forall \mathbf{a}_{-i} \in \mathbf{A}_{-i}.$$

In our example, we clearly see that our two suspects had better play their dominating strategies and not deviate. The joint strategy (D, D) is therefore an equilibrium (a kind of fixed point) for both of them where each one had better not deviate. More generally if, in a given game, all players have at their disposal a dominating strategy, then they had better actually choose it and, in this case, the result of the game is called an *equilibrium in dominating strategies*.

In fact, the equilibrium in dominating strategies rarely exists and we have to appeal to other types of solutions. For these cases, there exists a weaker solution concept called Nash Equilibrium.

8.2.2.4. Nash equilibrium

DEFINITION 8.2. A combination of strategies \mathbf{a}^* is said to be a **Nash Equilibrium** (in pure strategies) if the following inequality holds for each player i :

$$R_i(a_i^*, \mathbf{a}_{-i}^*) \geq R_i(a_i, \mathbf{a}_{-i}^*) \quad \forall a_i \in A_i.$$

In other words, if player i anticipates that other game participants will choose the strategies associated with a vector \mathbf{a}_{-i}^* , it can only maximize its payoff by selecting the strategy a_i^* . For i , this strategy is in fact a best response to \mathbf{a}_{-i}^* (denoted by br_i) and, when exists, it corresponds to

$$br_i : \mathbf{a}_{-i}^* \longmapsto \underset{a_i \in A_i}{\operatorname{argmax}} R_i(a_i, \mathbf{a}_{-i}^*).$$

Then, the Nash equilibrium can also be written as

$$\forall i, a_i^* \in br_i(\mathbf{a}_{-i}^*).$$

As can be seen, a Nash equilibrium constitutes a combination of strategies where each player maximizes its payoff assuming the other players' specific actions. It therefore has a "stability" property, which is satisfied for each player. That is why it is called an "equilibrium".

In the example from Figure 8.6, two companies COMPANY1 and COMPANY2 have the opportunity to engage in the production of a new good whose outlets are limited. No compromise is possible between the two companies if both decide to produce. This game includes two Nash equilibria, (*do not produce, produce*) whose payoffs are (0, 8) and (*produce, do not produce*) whose payoffs are (10, 0), each corresponding to a situation where one company produces, while the other does not. This example shows that, among both equilibria, none of them seems more reasonable than the other.

		COMPANY2	
		Produce	Do not produce
COMPANY1	Produce	-3, -2	10, 0
	Do not produce	0, 8	0, 0

Figure 8.6. Multiplicity of Nash equilibria. This game includes two Nash equilibria in pure strategies: (*do not produce, produce*) whose payoffs are (0, 8) and (*produce, do not produce*) whose payoffs are (10, 0)

In addition to this difficulty of facing multiple equilibria, there may be no equilibrium in pure strategies in a particular game. A well-known example is that of *Matching pennies* game presented in strategic form in Figure 8.7.

		PLAYER2	
		Tails	Heads
PLAYER1	Tails	1, -1	-1, 1
	Heads	-1, 1	1, -1

Figure 8.7. Matching pennies: an example of a game with no Nash equilibrium in pure strategies

In this case, we have to resort to mixed strategies. The notion of Nash equilibrium (as well as the notion of best response) indeed extends naturally – as previously did the notion of minimax – to the case of mixed strategies.

DEFINITION 8.3. A combination of mixed strategies π^* is a Nash equilibrium (in mixed strategies) if the following inequality holds for each player i :

$$u_i^{(\pi_i^*, \pi_{-i}^*)} \geq u_i^{(\pi_i, \pi_{-i}^*)} \quad \forall \pi_i \in \Delta A_i.$$

In the matching pennies game, presented in Figure 8.7, PLAYER1 chooses *Tails* with probability p and *Heads* with probability $1 - p$. For PLAYER2 the respective probabilities are q and $1 - q$. Then, the expected payoff of PLAYER1 is given by the function u_1 linear in p :

$$\begin{aligned} u_1^{(p,q)} = & pqR_1(Tails, Tails) + p(1-q)R_1(Tails, Heads) \\ & + (1-p)qR_1(Heads, Tails) + (1-p)(1-q)R_1(Heads, Heads). \end{aligned}$$

In this particular two-player game, the Nash equilibrium is a saddle point on the surface $u_i^{(p,q)}$. To maximize its expected payoff, PLAYER1 has to find a probability p such that $\frac{d(u_1)}{dp}(p) = 0$, that is,

$$\begin{aligned} & pR_1(Tails, Tails) - pR_1(Tails, Heads) + (1-p)R_1(Heads, Tails) \\ & - (1-p)R_1(Heads, Heads) = 0. \end{aligned}$$

Replacing R_1 parameters with the values given by the matrix in Figure 8.7, we get $p + p - (1-p) - (1-p) = 0$, hence $p = 1/2$.

The same reasoning for PLAYER2 (now we are searching for a point where $\frac{d(u_1)}{dp}(q) = 0$) gives $q = 1/2$.

The result $(1/2, 1/2)$ is called an *equilibrium in mixed strategies* and corresponds to a situation in which each player chooses half of the time *Tails* and half of the time *Heads*.

But can we find an equilibrium of this type in any game in strategic form? If the game is finite, the answer is yes, thanks to the following theorem.

THEOREM 8.1 [NAS 51]. *Each finite game in strategic form admits a Nash equilibrium in mixed strategies.*

We could wonder whether the solution given by a Nash equilibrium corresponds to an efficient coordination mechanism. Two concepts can help answering this question: Pareto efficiency and Pareto optimality.

For the first concept, a combination of strategies $\hat{\mathbf{a}}$ is said to *Pareto dominate* another combination \mathbf{a} if

$$\begin{aligned} \forall i, \quad R_i(\hat{a}_i, \hat{\mathbf{a}}_{-i}) &\geq R_i(a_i, \mathbf{a}_{-i}), \\ \exists j \text{ such that } R_j(\hat{a}_j, \hat{\mathbf{a}}_{-j}) &> R_j(a_j, \mathbf{a}_{-j}). \end{aligned}$$

A combination of strategies $\hat{\mathbf{a}}^*$ is *Pareto optimal* if no other combination exists that Pareto dominates it. For example, in the prisoner's dilemma presented in Figure 8.5, the (D,D) combination is a Nash equilibrium but the (C,C) combination Pareto dominates it. We will therefore have to remember that a Nash equilibrium is not necessarily a Pareto optimal joint strategy.

Yet, when there are multiple Nash equilibria, one equilibrium can Pareto dominate another one. The problem is to know how to “attract” the players towards this dominating equilibrium instead of another, dominated, equilibrium.

8.2.3. Dynamic games of complete information

Contrary to static games (such as games in strategic form) which are played only once, dynamic games describe processes that extend through time. These processes are made up of several participating agents (players) which can condition their current behavior on the past observable decisions of other agents. In this section, we assume that the game is played in multiple stages and that all past actions are observable and known to all participants. In this context, a stage could represent, though not always, a time period. In such a game, a strategy specifies the action played by each player at each stage where it acts depending on the current state of the game. Then, the game history is important, and each player chooses the appropriate action to perform taking into account the past history of the game.

Two types of dynamic games of incomplete information are usually distinguished. Games of the first type are called *games in extensive form* where players play in turns. We will limit ourselves to the case where the game is not only of complete information,

but also of *perfect information*. This means that on a player's turn, this player knows the situation of the game, in particular the moves made by the other players until now.

In the second type of games, called *repeated games*, multiple agents choose their actions simultaneously at a given stage of the game. For each player, the other players' actions are unknown, but the past history is known and influences each player's choice.

A general conceptual framework of these dynamic games (with perfect information) can be defined as follows. Let us denote by \mathbf{a}^θ the vector of joint actions chosen at stage θ of the game by the participants operating at this stage. Let t be some particular stage of the game. The game history at stage t , $h^t = \{\mathbf{a}^0, \mathbf{a}^1, \dots, \mathbf{a}^{t-1}\}$, is then defined by the sequence of all decisions made by the players in past stages $\theta = 0, 1, \dots, t-1$, with h^0 being the initial history. Moreover, all past actions are observable and known to all players. Let us note that for $\theta > t$, the remainder of the game can be seen as another game induced by the history h^t ; this new game is called the sub-game $G(h^t)$.

Such a formal framework will now allow us to define what is called a pure strategy in a T -stages dynamic game. To that end, let us denote by $A(h_i)$ the set of actions available to player i in the situation described by history h_i , and let A_i be the set of all actions available to player i . A pure strategy δ_i for player i is a mapping $\delta_i : H \mapsto A_i$, where $\delta_i(h_i) \in A(h_i)$, and H is the set of all possible histories. For a given player i , a pure strategy is therefore a set of rules for selecting a particular action at each stage of the game, given the past history. Contrary to what has been seen previously in the framework of pure strategies in a static game, here the definition takes into account previous decisions. It therefore allows for a dynamic analysis of choices.

Let us first consider games in extensive form with perfect information.

8.2.3.1. Games in extensive form with perfect information

Let us take the example of the tree represented in Figure 8.8 where the game takes place in multiple stages. Thus, for player 1 (denoted by node 1), a strategy consists of choosing between actions a and b . For player 2 (node 2) who acts just after, its strategy is a function defined over the set of strategies of player 1. The principle consists here in using the *backward induction* where the reasoning is done in reverse order to the normal progress of the game. Thus, player 1 will consider that player 2 is going to play:

- b if it plays a , leading to payoff $(2, 1)$ or,
- a if it plays b , leading to payoff $(1, 1)$.

As a consequence, player 1 will play a , leading player 2 to play b and ending up with the Nash equilibrium $(2, 1)$ for which no unilateral deviation pays off.

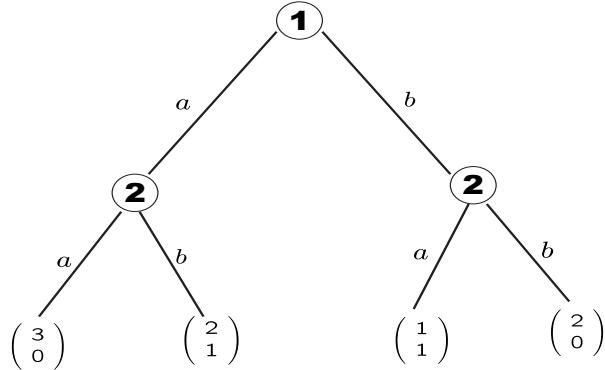


Figure 8.8. An example of a game in extensive form (stage by stage dynamic game)

In fact, this reasoning comes from the fact that player 2 is supposed to choose the best action for itself. Thus, if 1 plays a then 2 plays b , but if 1 plays b , then 2 plays a . So, player 1 takes this knowledge into account and consequently acts to provide its best response.

In general, $|A_g|$ players can appear in the tree in no particular order. Let us note that the knowledge of the current node n_t in the tree is an information equivalent to the current history h_t . In a game with incomplete information, the “active” player only has incomplete information about the current history – the actions of some agents at some time steps are hidden – which amounts to only knowing that the current node is in a set $\{n_1, \dots, n_t\}$.

Another important point is that games in extensive form can all be written in an equivalent manner as normal form games. It is sufficient to observe that, in the corresponding normal form game, each player will have a finite number of pure strategies a_i available. This is because the tree has a finite size. The latter allows us to rewrite the game in extensive form of Figure 8.8 into normal form as in Figure 8.9. Let us note that, in the case of perfect information, we end up with a game for which an equilibrium in dominating strategies exists.

		PLAYER 2				
		$a \rightarrow a / b \rightarrow a$	$a \rightarrow a / b \rightarrow b$	$a \rightarrow b / b \rightarrow a$	$a \rightarrow b / b \rightarrow b$	
PLAYER 1		a	3, 0	3, 0	1, 1	1, 1
		b	1, 1	2, 0	1, 1	2, 0

Figure 8.9. Game from Figure 8.8 rewritten in a matrix form. For player 2, we indicate what is the response to player 1's move

Because of this equivalence, we will not go further into extensive form games. To know more about these games, the reader can refer to one of the books [YIL 03, FUD 91].

Repeated games, another type of dynamic games with complete information, are of higher interest for us, since they are at the root of the concept of stochastic games – one of the most powerful models of interaction among rational agents.

8.2.3.2. *Repeated games*

Repeated games (or iterated games) model situations where players repeatedly interact, playing the same game. Contrary to extensive form games with perfect information, players in a repeated game choose their actions simultaneously (without knowing the choices of other players). Once the actions are chosen, they are known to other agents and make part of the game history.

Thus, the prisoner's dilemma can, for example, be repeated several times, producing the history $((C, C), (D, C), (D, D)$, etc.).

In such interactions, a player can, at each stage, condition their behavior on the past behaviors of other players. Repeated games are a particular case of the previously introduced dynamic games. When the players are committed to a repetitive situation, they have to consider not only their short-term (immediate) payoff, but also their long-term payoff. Such phenomena as reputation, trust, etc. are therefore introduced. For example, if a prisoner's dilemma is played once, the players will tend to play the Nash equilibrium: (D, D) . On the contrary, if the same game is repeated by the same two players, some behaviors could give rise to a cooperation (implicit agreement) leading to (C, C) . Two classes of repeated games are usually distinguished: (a) finite time horizon repeated games and (b) infinite time horizon repeated games.

For example, coming back to the definition of a dynamic game's pure strategy given above, the following strategy could be specified in the case of the prisoner's dilemma:

$$\begin{aligned} a_i(h^0) &= C, \\ a_i(h^t) &= \begin{cases} C & \text{if } a_j^\tau = C, j \neq i, \text{ for } \tau = 0, 1, \dots, t-1, \\ D & \text{otherwise.} \end{cases} \end{aligned}$$

This strategy amounts to “start by cooperating in the first period and keep on doing so as long as the other player also cooperates in the previous periods; otherwise, defect”. This strategy is called the *grim trigger* strategy. Other strategies can be described using the same formulation: (i) always Defect (ALL-D); (ii) always cooperate (ALL-C); (iii) Tit-For-Tat (TFT); etc.

As in static games, the notion of mixed strategy can be defined in the framework of dynamic games, and in particular in the framework of repeated games. In a repeated game, a *mixed strategy* π_i for player i is a function $\pi_i : H \mapsto \mathcal{A}_i$, where $\mathcal{A}_i = \Delta A_i$ is the space of probability distributions over A_i . This function therefore links the possible t -period histories with *mixed actions*² $\alpha_i \in \mathcal{A}_i$. Because only the moves actually done by each player are observable, player i 's strategy depends not on the past mixed actions of all players (including itself), but on the past observed actions (that is, histories).

Let us now see how a repeated game can be analyzed in terms of expected payoff and let us just consider pure strategies for the prisoner's dilemma (see Figure 8.5). In this case, the players' payoffs for the complete sequence of game repetitions $(\mathbf{a}^0, \mathbf{a}^1, \dots)$ deterministically induced by a pure strategy profile $\pi = (\pi_1, \pi_2)$ can be represented by the *average payoff* they get at each period. In the case of SUSPECT1, for example, its payoff function over the complete repeated game given the strategy profile π is

$$\bar{u}_1^\pi = \lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=0}^T R_1(\mathbf{a}^t),$$

where \mathbf{a}^t is the joint action executed in period t according to the joint strategy π , R_1 is the reward function of such a suspect. We can similarly imagine that such a suspect uses the *discounted value of its payoffs* on the complete game sequence:

$$u_{1,\gamma}^\pi = \sum_{t=0}^{\infty} \gamma^t R_1(\mathbf{a}^t), \quad \gamma \in [0, 1],$$

where γ is the discount factor for SUSPECT1. Combining the two previous functions leads to the *average discounted payoff*:

$$\bar{u}_{1,\gamma} = (1 - \gamma) \sum_{t=0}^{\infty} \gamma^t R_1(\mathbf{a}^t),$$

where we expect that the following equality is verified:

$$\lim_{\gamma \rightarrow 1} \bar{u}_{1,\gamma} = \bar{u}_1.$$

Thus, there exist several possibilities to represent the long-term payoffs of the players in a repeated game.³ Here is an example taken from [YIL 03] on how to

2. Notice that mixed actions in repeated games mean the same as mixed strategies in normal form games, i.e. distributions over actions. We changed the notation to keep π_i denoting the strategy of player i .

3. The average and γ -discounted criteria seen for MDPs in section 1.2.3 can be recognized.

use them. Let us come back to the infinitely repeated prisoner's dilemma illustrated in Figure 8.5 and let us consider the *grim trigger* strategy discussed above. Let us suppose that the player SUSPECT1 follows this strategy and that SUSPECT2 is aware of that. We could wonder what is then the optimal strategy for SUSPECT2 in face of the strategy of SUSPECT1. If it always plays C , it will get payoffs equal to -1 forever. The discounted value of these rewards in this case is

$$\sum_{t=0}^{\infty} -\gamma^t = \frac{-1}{1-\gamma}.$$

On the contrary, if it starts with D from the first turn, it will initially get 0, then -8 for remaining periods, what gives it a discounted value of

$$0 + (-8)\gamma + (-8)\gamma^2 + \dots = -8(\gamma + \gamma^2 + \dots) = \frac{-8\gamma}{1-\gamma}.$$

The player SUSPECT2 will therefore be better of cooperating from the beginning if

$$\frac{-1}{1-\gamma} > \frac{-8\gamma}{1-\gamma}, \quad \text{that is} \quad \gamma > \frac{1}{8}.$$

The previous strategy of SUSPECT1 and its counterpart that we have just detailed for SUSPECT2 constitute a Nash equilibrium for the infinitely repeated prisoner's dilemma with discounted values for $\gamma > \frac{1}{8}$. By choosing a different form for the agents' utility functions (e.g. the average payoffs or the average discounted payoffs), we could obtain another solution to the same problem. For more details on these aspects, the reader is encouraged to consult the reference books [FUD 99, GEN 00, OSB 04, YIL 03].

8.3. Stochastic games

Stochastic games (SG) – also called Markov games – extend MDPs to the case where there are multiple players in a common environment. These agents perform a joint action that defines both the reward obtained by the agents and the new state of the environment. On the other hand, a stochastic game can be seen as a repeated game with multiple states. This means that, after having played a matrix game (corresponding to one state of the stochastic game), the players are transferred to another state of the stochastic game to play another matrix game. This is therefore a hybrid model bringing together (repeated) dynamic games and MDPs.

8.3.1. Definition and equilibrium of a stochastic game

Formally, a stochastic game is defined as a five-tuple $\langle Ag, S, \mathbf{A} \equiv \times_{i \in Ag} A_i, \{R_i : i = 1, \dots, |Ag|\}, T \rangle$, where Ag is the set of agents and $|Ag|$ is their number, S is the finite set of states of the game, $A_i = \{a_i^1, \dots, a_i^{|A_i|}\}$ is the set of actions of agent i , $R_i : S \times \mathbf{A} \mapsto \mathbb{R}$ is the reward (payoff) function of agent i , and the transition function $T : S \times \mathbf{A} \times S \mapsto [0, 1]$ models the state transitions, depending on the current state and the agents' joint action (denoted by $\mathbf{a} \in \mathbf{A}$).

As single-state games, stochastic games can be classified based on the structure of their reward functions. In *fully cooperative* stochastic games, all agents have the same reward function ($R_1 = \dots = R_{|Ag|}$). A stochastic game is called *zero-sum* if, for all $s \in S$ and $\mathbf{a} \in \mathbf{A}$, $\sum_i R_i(s, \mathbf{a}) = 0$. The term of *general-sum* stochastic game is used to refer to all other types of games.

At each turn of the game, given the current state $s \in S$, the agents choose their actions and the joint action $\mathbf{a} = (a_1, \dots, a_{|Ag|})$ is obtained. Each agent i then gets the reward $R_i(s, \mathbf{a})$ and the system moves to the next state s' following the transition function T . A *policy* $\pi_i : S \mapsto \Delta A_i$ for agent i defines for each state a local strategy (in the game theoretic sense). Player i 's policy $\pi_i(s)$ can be viewed as a $|S|$ -dimensional vector whose elements define a probability distribution over player i 's actions specific to the normal form game defined by state s .

The game theoretic notion of expected utility of a joint strategy $\pi \equiv (\pi_1, \dots, \pi_{|Ag|})$ for a player refers to the expected *immediate* reward over the *opponent players' strategies*: $u_i^\pi = E_{\mathbf{a} \in \mathbf{A}}[R_i(\mathbf{a})]$. On the other hand the MDPs' value function is the *temporal* expectation over the reward. We will therefore understand the concept of utility $U_i(s)$ in a stochastic game as the temporal expectation of expected immediate utilities $u_i(s)$ for agent i . These expected immediate utilities are defined in each state s as the expected utilities of normal form games, i.e. $u_i^\pi(s) = E_{\mathbf{a} \in \mathbf{A}} R_i(s, \mathbf{a})$. As a consequence, state utilities $U_i(s)^\pi$ for each player i , associated with the joint policy $\pi \equiv \times_{i \in Ag} \pi_i$, are defined as the utility expected by agent i if all agents infinitely follow this joint policy from state s :

$$\begin{aligned} U_i^\pi(s) &= E \left[\sum_{t=0}^{\infty} \gamma^t u_i^\pi(s^t) \mid s^0 = s \right] \\ &= u_i^\pi(s) + \gamma \sum_{\mathbf{a} \in \mathbf{A}} \sum_{s' \in S} T(s, \mathbf{a}, s') \pi^\mathbf{a}(s) U_i^\pi(s'), \end{aligned}$$

where $\pi^\mathbf{a}(s)$ denotes the probability of joint action \mathbf{a} in state s given the joint policy π . Like in MDPs, s^0 is the initial state and $\gamma \in [0, 1]$ is the discount factor.

As explained above, each state of a stochastic game can be seen – locally – as a normal form game. A transition between two states is illustrated in Figure 8.10.

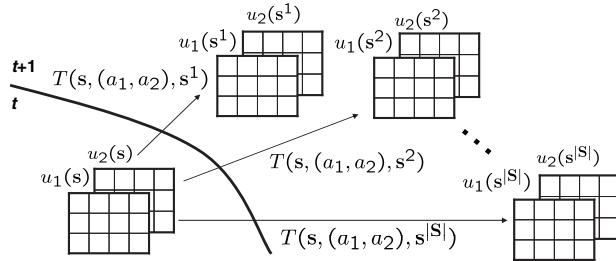


Figure 8.10. Two-player stochastic game: possible transitions between turn t and turn $t + 1$ when the players play a joint action $\mathbf{a} = (a_1, a_2)$; each state can be seen as a normal form game

Stochastic games are a formal framework for modeling multi-agent environments. Agents are *a priori* non-cooperative since they follow individual objectives. Yet, they may be led to coordinate themselves, even to cooperate, so as to reach individual objectives.

In a stochastic game, a vector of strategies $(\pi_1^*, \dots, \pi_{|Ag|}^*)$ is a Nash equilibrium if, for all states $s \in S$ and $i = 1, \dots, |Ag|$,

$$U_i^{(\pi_1^*, \dots, \pi_{|Ag|}^*)}(s) \geq U_i^{(\pi_1^*, \dots, \pi_{i-1}^*, \pi_i, \pi_{i+1}^*, \dots, \pi_{|Ag|}^*)}(s) \quad \forall \pi_i \in \Pi_i,$$

where Π_i is the set of policies available to agent i . Thus no agent can improve its utility by unilaterally deviating from a Nash equilibrium.

The following theorem shows the existence of a Nash equilibrium in stationary (in the sense of MDPs) strategies.

THEOREM 8.2 [FIN 64]. Any discounted n -player ($n \geq 2$) stochastic game has at least one Nash equilibrium in stationary strategies.

As in single-state games, the concepts of Pareto efficiency (Pareto optimality) can be defined.

DEFINITION 8.4. A joint policy $\hat{\pi}$ Pareto-dominates another joint policy π (we write $\hat{\pi} > \pi$) if and only if, in all states $s \in S$,

$$\forall i \in Ag, \quad U_i^{\hat{\pi}}(s) \geq U_i^\pi(s) \quad \text{and} \quad \exists j \in Ag \quad \text{s.t.} \quad U_j^{\hat{\pi}}(s) > U_j^\pi(s).$$

DEFINITION 8.5. If a joint policy $\hat{\pi}^*$ is not Pareto-dominated by any other joint policy, then $\hat{\pi}^*$ is Pareto-optimal.

So a Pareto-optimal solution is a joint policy in which no agent's expected gain can be improved upon without decreasing the expected gain of any other agent. There are many examples of general-sum games where a Pareto-optimal solution is not a Nash equilibrium and vice versa (as, e.g. in the prisoner's dilemma from Figure 8.5 in section 8.2). However, it has to be noticed that in fully-cooperative games, every Pareto-optimal solution is also a Nash equilibrium by definition: if a joint action provides one agent with maximum possible reward, it also maximizes the reward received by the other agents.

8.3.2. Solving stochastic games

We present here some algorithms for solving stochastic games. As observed in the previous section, stochastic games are multi-stage games which can be seen as an extension of MDPs to the multi-agent setting. Being games, they do not possess an optimal solution in the MDP sense, i.e. independent of other players. We will see that stochastic games essentially ask two different questions:

- 1) What is, for player i , the best response to the other players' $(-i)$ strategies?
- 2) What are the equilibrium situations for these games?

Looking at equilibria, the concept that is most commonly used as a solution of a stochastic game is the one of Nash equilibrium in stationary strategies, whose existence has been proved. Yet, the solution (or equilibrium) is not always unique (as, e.g. in the game of Figure 8.6 in section 8.2). Having multiple equilibria with different values leads to coordination problems. Indeed, in that case, if agents decide to play different equilibria, the joint action being played may not correspond to an equilibrium. Depending on the game at hand, this can constitute a disaster if the values of utilities vary a lot from one joint action to another. Readers interested in the problem of coordination in stochastic games may refer to [LIT 94].

All the algorithms that we will see in the framework of stochastic games are presented relatively to assumptions made about the knowledge or observability of some game properties. Those properties guide the choice of the algorithm.

The first difference is whether a model for the game is available or not. Most game theoretic algorithms assume that the game is fully known, as they make use of the transition and reward functions. Vice versa, reinforcement learning algorithms assume that the game being played is not known and must be observed through experience. Among reinforcement learning methods, another distinguishing characteristic is whether the actions selected by the other agents or the other agents' rewards are observable or not. Thus the algorithms for SGs are grouped in three categories:

- 1) the game model is available;
- 2) the game model is unavailable + the actions of the other agents are observable;

3) the game model is unavailable + the actions of the other agents are not observable.

Moreover, within the second category, equilibrium learning algorithms and algorithms based on opponent modeling techniques are distinguished.

8.3.2.1. Game model available

This subsection presents three algorithms worked out by researchers from the game theory community. These algorithms assume that the game is fully known, i.e. transition and reward functions are known. The first two algorithms use iterative techniques and only consider two-player games. The third algorithm concerns the technique of fictitious play for finding equilibria in a known stochastic game.

The first algorithm finds a Nash equilibrium in two-player zero-sum stochastic games [SHA 53]. This algorithm is an extension of value iteration (seen in Chapter 1) to the case of stochastic games. To that end, Shapley searches for a Nash equilibrium of the normal form game associated with the current state. The complete definition of Shapley's algorithm is given by Algorithm 8.1. In this algorithm, $U(s)$ indicates the vector of expected utilities for all agents in state s .

Algorithm 8.1: Shapley's algorithm [SHA 53] for zero-sum SGs

```

Initialize  $U^0(s)$  with arbitrary values
 $n \leftarrow 0$ 
repeat
  for  $s \in S$  do
    Build matrix  $G(s) = \{g_{\mathbf{a}} : g_{\mathbf{a}} = R(s, \mathbf{a}) + \gamma \sum_{s' \in S} T(s, \mathbf{a}, s') U^n(s')\}$ 
     $U^{n+1}(s) = Value(G(s))$ 
     $n \leftarrow n + 1$ 
  until  $\|U^{n+1}(s) - U^n(s)\| < \epsilon \forall s$ 
  for  $s \in S$  do
    Build matrix  $G(s)$ 
     $\pi(s) = Equilibrium(G(s))$ 
return  $U^n(s), \pi(s) \forall s$ 

```

Shapley's algorithm is capable of finding an equilibrium in any two-player zero-sum game, because it exploits the following theorem.

THEOREM 8.3 [SHA 53]. *Any zero-sum discounted stochastic game G with infinite time horizon has a unique value. This value is given by the sequence of unique values of the state games $G(s) \forall s$. Each player of game G possesses an optimal strategy which uses mixed minimax strategies in each state game $G(s)$.*

Kearns, Mansour and Singh [KEA 00] have gone further. They have proposed an algorithm similar to the one of Shapley, but their value iteration-based algorithm, named FiniteVI, has been designed for general-sum stochastic games. The authors have proven that FiniteVI converges to a Nash equilibrium when the stochastic game horizon is finite. Concerning infinite horizon games, it has been recently demonstrated [ZIN 06] that there exists a class of stochastic games for which no value iteration algorithm based on the Bellman equation (like, e.g. FiniteVI) can converge to a stationary policy.

The structure of the FiniteVI algorithm is the same as that of Shapley's algorithm. It is yet different in that (1) the horizon T is used as a stopping criterion, and (2) the functions *Equilibrium* (returning an equilibrium of a normal form game) and *Value* (returning the value of an equilibrium of a normal form game) are defined differently.

While the minimax equilibrium used by Shapley's algorithm is guaranteed to be unique, this is not the case in the framework of general-sum games. On the contrary, there may be several equilibria with different values in a general-sum game. Thus, since only one equilibrium should be chosen at each iteration by the function *Value*, Kearns and his colleagues have proposed the use of a function $f(G(s))$ that selects a single equilibrium among many. However, the function $f(G(s))$ has not been defined. This makes difficult a direct application of FiniteVI. We should also note that, contrary to the zero-sum case, no computationally efficient algorithm exists that finds a Nash equilibrium in a general-sum game.

Among other value iteration algorithms for stochastic games, the reader may refer to [POL 69, HOF 66, BOW 03a].

Another algorithm comes from the technique known as “fictitious play” which allows a player to estimate the strategy played by its opponent in a view to play the best response to this estimate. This technique was first proposed by Brown in the setting of repeated games [BRO 51].

In the fictitious play algorithm, players participating in a repeated game keep individual empirical beliefs about the strategies followed by other players. The fictitious play model assumes that each player i chooses its actions at each period so as to maximize its expected utility, $u_i^{\hat{\pi}_{-i}}$, given its estimate $\hat{\pi}_j$ of the mixed strategy of each opponent j (where $\hat{\pi}_{-i}$ denotes the vector of such estimates, $\hat{\pi}_{-i} \equiv (\hat{\pi}_j)_{j \neq i}$). The estimate of an opponent's strategy takes the following form. Player i is supposed to have, for each other player j , an initial weighting function $c_{ij}^0 : A_j \mapsto \mathbb{R}_+$. This weight is updated by adding 1 to the weight of an opponent strategy a_j when this strategy is played by the opponent:

$$c_{ij}^t(a_j) = c_{ij}^{t-1}(a_j) + \begin{cases} 1 & \text{if } a_j^{t-1} = a_j, \\ 0 & \text{otherwise.} \end{cases}$$

In these conditions, at any moment of time t , the estimated probability that an opponent j plays an action a_j is given by

$$(\hat{\pi}_j^{a_j})^t = \frac{c_{ij}^t(a_j)}{\sum_{a'_j} c_{ij}^t(a'_j)}.$$

Then, the best action to play for agent i is the one that maximizes its expected utility given its estimate of opponent policies (in other words, its best response to the corresponding joint policy):

$$a_i^t = \operatorname{argmax}_{a_i} \sum_{\mathbf{a}_{-i}} R_i(a_i, \mathbf{a}_{-i}) (\hat{\pi}_{-i}^{\mathbf{a}_{-i}})^t.$$

Fictitious play converges to a Nash equilibrium in *iterated dominance solvable* games, that is, games in which dominated actions can be removed iteratively to finally obtain a single action or a set of equivalent actions [FUD 99].

The version of fictitious play adapted to stochastic games, given the transition function $T(s, \mathbf{a}, s')$, is presented in Algorithm 8.2. It is due to Vrieze [VRI 87].

Algorithm 8.2: Fictitious play algorithm for SGs for a player i

```

for all  $s, a_i$  do Initialize  $q_i(s, a_i) \leftarrow \frac{1}{|\mathbf{A}_{-i}|} \sum_{\mathbf{a}_{-i} \in \mathbf{A}_{-i}} R_i(s, (a_i, \mathbf{a}_{-i}))$ .
 $n \leftarrow 0$ .
repeat
  for all  $s$  do
    Choose action  $a_i^n = \operatorname{argmax}_{a'_i} \frac{q_i(s, a'_i)}{n}$ .
    Play action  $a_i^n$ .
    Observe joint action and put it in  $\mathbf{a}$ .
    for all  $a_i$  do
      Update value  $q_i(s, a_i)$ :
       $q_i(s, a_i) \leftarrow$ 
         $q_i(s, a_i) + R_i(s, (a_i, \mathbf{a}_{-i})) + \gamma \sum_{s' \in S} T(s, \mathbf{a}, s') Value(s')$ 
        where  $Value(s') = \max_{a_i} \frac{q_i(s, a_i)}{n}$ 
     $n \leftarrow n + 1$ .
  until  $n > N$ 
for all  $s$  do  $\pi_i^{a_i}(s) \leftarrow 1$  if  $a_i = a_i^N$  and  $\pi_i^{a_i}(s) \leftarrow 0$  otherwise.
return  $\pi_i(s) \forall s$ .

```

It is worth noting that an algorithm similar to fictitious play has been proposed by Young [YOU 93]. This algorithm for solving repeated games is called adaptive play. The difference between the two algorithms lies in the way the opponent's strategy is estimated. While, in fictitious play, the whole history is taken into account, in adaptive play the agent uses a limited sample of the recent history. Such a modification allows proving the convergence of adaptive play for a larger class of coordination games called *weakly acyclic games* [YOU 98].

8.3.2.2. Game model unavailable, \mathbf{A}_{-i} observed, equilibrium learners

Algorithms in this category, which we call equilibrium learners, learn an equilibrium policy through observations of the game's transitions and rewards and also assume that each agent is capable of observing the other agents' actions. Most of the time, equilibrium learners seek to learn and play a specific Nash equilibrium of the stochastic game.

The idea underlying this kind of algorithms is the following. The agents learn by reinforcement while simultaneously acting and receiving rewards. They share a common reinforcement learning structure based on Q-learning – seen in section 2.5.3 of Chapter 2 – combined with techniques to search for equilibria in normal form games. The notion of a Q -function is extended to maintain the value of joint actions. After each turn of the game (or after each “joint action–transition” sequence) each agent i updates its Q_i -function that associates real numbers with “joint action–state” pairs according to

$$Q_i(s, \mathbf{a}) \leftarrow Q_i(s, \mathbf{a}) + \alpha(R_i(s, \mathbf{a}) + \gamma Value_i(s') - Q_i(s, \mathbf{a})), \quad (8.2)$$

where the function $Value_i(s)$ returns player i 's value in an equilibrium of a game G . Such a game is composed of the Q_i -values of agents in state s . It is clear here that, to be capable of computing this function, agent i generally needs to observe the actions and rewards of all other agents, or that the other agents communicate this information to agent i .

The algorithms reviewed in this subsection differ mainly in the definition of the function $Value_i$. The specific structure of this function is chosen according to the structure of the reward function of the stochastic game.

Littman's Friend- Q ⁴ [LIT 01c] algorithm is specifically designed for fully cooperative SGs. In this algorithm, $Value_i$ returns to player i the largest Q_i -value among those defined for different joint actions in the given state. In the case of multiple such actions, one action is randomly taken. In consequence, in the absence of additional coordination mechanisms, the resulting joint action played by the

4. Also called Team- Q .

players can be suboptimal. In multi-agent systems, this problem is generally known as equilibrium selection problem. Friend- Q has no any coordination mechanism to avoid such problems. It simply assumes that in the game being solved the optimal joint action in every state (and therefore, Nash equilibrium) is unique.

At the opposite extreme, in the Minimax- Q algorithm, Littman [LIT 94] studies two-player zero-sum SGs where the function $Value_i$ returns the minimax value of a normal-form game composed of the agents' Q_i -values. The policy followed by the agents is then the minimax policy in mixed strategies.

Hu and Wellman [HU 03] extend the Minimax- Q algorithm to general-sum n -player SGs. Their algorithm is called Nash- Q . Here, $Value_i$ returns the value of a Nash equilibrium of the state game built using the Q_i -functions of all agents. Therefore, each agent i needs to maintain the Q_j -functions of all agents $j \neq i$. In Nash- Q , all agents always choose the same equilibrium thanks to a coordination mechanism that prescribes them to always choose a certain pre-defined equilibrium. For example, this can be the first equilibrium found using the same deterministic method. As a consequence, the policy jointly followed by the agents is the policy corresponding to the chosen Nash equilibrium.

A more formal definition of the base algorithm for the Nash- Q and Minimax- Q algorithms is represented by Algorithm 8.3. As it was already the case with Shapley's and Kearns' algorithms, the difference between both algorithms lies in the definition of the $Equilibrium_i$ and $Value_i$ functions. In Minimax- Q , these functions, respectively, return to agent i the component π_i of the minimax equilibrium policy in state s and the utility of this equilibrium for player i . In Nash- Q , these two functions, respectively, return player i 's policy in a certain Nash equilibrium and the utility of this equilibrium for player i . In Algorithm 8.3 (as well as in the other algorithms presented in the remaining part of this chapter), the expression "some exploration" refers to the various exploration techniques discussed in Chapter 2.

Notice that the convergence proofs for both Minimax- Q and Nash- Q can be found in [LIT 94] for the former and [HU 03] for the latter.

Littman [LIT 01b] pointed out that the assumptions for Nash- Q are quite restrictive. Consequently, he reinterpreted this algorithm as the Friend-or-Foe- Q (FoF- Q) algorithm. Compared to Nash- Q , FoF- Q does not require that agent i maintain the Q_j -functions of the other agents $j \neq i$. Instead, all agents using FoF- Q are told whether an opposing agent is a "friend" or a "foe". In other words, FoF- Q assumes that there are two competing teams playing the SG. Thus, from one agent's perspective, each other agent is either a member of its own team, or it is a member of the competing team. In the former case, agent i maximizes its own payoff and employs Friend- Q ; in the latter case, agent i uses Foe- Q , i.e. Minimax- Q . While

Algorithm 8.3: A base algorithm for the Minimax- Q and Nash- Q algorithms for player i

```

for all  $s, \mathbf{a}$  and  $j \in Ag$  do Initialize  $Q_j(s, \mathbf{a})$  with arbitrary values.
Set  $s$  to the current state.
Build the game  $G(s)$  based on the values  $Q_j$  in  $s$ ,  $\forall j \in Ag$ .
Choose a policy  $\pi_i(s) = Equilibrium_i(G(s))$ .
repeat
    Play policy  $\pi_i(s)$  with some exploration.
    Observe joint action and put it in  $\mathbf{a}$ .
    Observe each agent's reward and put them in  $\mathbf{r}$ .
    Observe new state and put it in  $s'$ .
    Build the game  $G(s')$  based on the values  $Q_j$  in  $s'$ ,  $\forall j \in Ag$ .
    Choose a policy  $\pi_i(s') = Equilibrium_i(G(s'))$ .
    for each player  $j$  do Update value  $Q_j(s, \mathbf{a})$  using equation (8.2).
     $s \leftarrow s'$ ,  $t \leftarrow t + 1$ .
until  $t > T$ 
for  $s \in S$  do
    Build matrix  $G(s)$  as previously.
     $\pi_i(s) = Equilibrium_i(G(s))$ .
return  $\pi_i(s) \forall s$ .
```

FoF- Q converges to a Nash equilibrium with less restrictions than Nash- Q , its applicability is still limited to only special cases of stochastic games.

8.3.2.3. Game model unavailable, \mathbf{A}_{-i} observed, opponent modeling

Algorithms in this category also assume that each agent is capable of observing the other agents' actions. However, it then uses this information in a different way. More precisely, in the spirit of Fictitious Play, this information is used to predict how other agents will behave.

The techniques to predict the future behavior of an agent giving its observed past behavior are called opponent modeling. Such techniques are largely used in dynamic game theory to make each player capable of adapting to its opponents and the changes in their strategies. The interest in using opponent modeling in stochastic games is that if a game has multiple equilibria, then the optimal policy for one agent must depend on the policies actually played by the other agents. As we already mentioned, such opponent-independent algorithms like Nash- Q , Minimax- Q , or Friend- Q do not always yield optimal behaviors; a number of coordination mechanism (based on sets of rules, previously distributed information or communication) must be employed to overcome inefficiency.

Uther and Veloso [UTH 03] investigated the opponent modeling technique in the context of zero-sum repeated games. They proposed the opponent modeling Q-learning algorithm (OMQ). Claus and Boutilier [CLA 98] examined a similar approach for fully-cooperative repeated games with *joint action learners* (JALs). Their algorithm is a combination of the Q-learning based on joint actions (equation (8.2)) and fictitious play. Indeed, like in fictitious play, the joint action learners maintain estimates of the other agents' strategies, which requires observing other agents' actions. However, unlike fictitious play for stochastic games [VRI 87], the JAL algorithm does not require the knowledge of the transition function.

The JAL algorithm is presented in Algorithm 8.4. Agent i learns models of the other agents as stationary distributions over their actions: $c_i(s, \mathbf{a}_{-i})$ counts the number of times agent i observed other agents taking joint action \mathbf{a}_{-i} in state s , while $c(s)$ counts the total number of times state s has been visited. Therefore, $\frac{c_i(s, \mathbf{a}_{-i})}{c(s)}$ is the probability that the other agents will select joint action \mathbf{a}_{-i} based on the learned model. Then, these distributions combined with "state-joint action" Q -values from the usual Q-learning update are used by each player to select an action.

Algorithm 8.4: OMQ [UTH 03] or JAL [CLA 98] algorithm for SGs for player i , adapted from [BOW 02a]

```

for all  $s$  and  $\mathbf{a}_{-i}$  do Initialize  $Q_i(s, \mathbf{a}_{-i})$  arbitrarily,  $c_i(s, \mathbf{a}_{-i}) \leftarrow 0$  and
 $c(s) \leftarrow 0$ .
Initialize  $n \leftarrow 0$ ,  $s \leftarrow s^0$ .
repeat
  Choose action  $a_i^n = \operatorname{argmax}_{a_i} \sum_{\mathbf{a}_{-i}} \frac{c_i(s, \mathbf{a}_{-i})}{c(s)} Q_i(s, (a_i, \mathbf{a}_{-i}))$ .
  Play action  $a_i^n$  with some exploration noise.
  Observe joint action of other agents,  $\mathbf{a}_{-i}$ .
  Observe reward,  $R_i(s, (a_i, \mathbf{a}_{-i}))$ .
  Observe new state,  $s'$ .
   $Q_i(s, (a_i, \mathbf{a}_{-i}))$ 
     $\leftarrow (1 - \alpha)Q_i(s, (a_i, \mathbf{a}_{-i})) + \alpha(R(s, (a_i, \mathbf{a}_{-i})) + \gamma \operatorname{Value}(s'))$ 
    where  $\operatorname{Value}(s') = \max_{a_i} \sum_{\mathbf{a}_{-i}} \frac{c_i(s', \mathbf{a}_{-i})}{c(s')} Q_i(s', (a_i, \mathbf{a}_{-i}))$ .
   $c(s) \leftarrow c(s) + 1$ ,  $c_i(s, \mathbf{a}_{-i}) \leftarrow c_i(s, \mathbf{a}_{-i}) + 1$ ,  $s \leftarrow s'$ ,  $n \leftarrow n + 1$ .
until  $n > N$ 
for all  $s$  do
  if  $a_i = \operatorname{argmax}_{a_i} \sum_{\mathbf{a}_{-i}} \frac{c_i(s, \mathbf{a}_{-i})}{c(s)} Q_i(s, (a_i, \mathbf{a}_{-i}))$  then
     $\pi_i^{a_i}(s) \leftarrow 1$ ,
  otherwise
     $\pi_i^{a_i}(s) \leftarrow 0$ .
return  $\pi_i(s) \forall s$ .
```

A similar algorithm, proposed by Gies and Chaib-Draa [GIE 06], is called *adaptive play Q-learning*. This approach extends adaptive play to the Q-learning based on joint actions. Let us recall that adaptive play is a variant of fictitious play in which the agents use a limited sample of the recent game history to calculate the opponents' model [YOU 93].

8.3.2.4. Game model unavailable, \mathbf{A}_{-i} not observed

A less restrictive assumption is to consider non-communicating agents, unable to observe each others' actions and rewards. Let us call such agents independent learners (ILs) [CLA 98]. Such assumption better reflects a number of practical applications, such as multi-robot tasks, where it is usually difficult to ensure joint action observability. With no observability of other players' actions, the problem of behaving effectively becomes more complicated. On the other hand, this often brings the benefit that the size of the state-action space does not grow considerably as the number of agents increases.

The first algorithm that has been applied to ILs was the standard Q-learning algorithm, in which each agent independently maintains the Q_i -values of its individual actions [TAN 93]. For each agent $i \in Ag$, the Q_i -value update rule in such decentralized Q-learning can be written as follows:

$$Q_i(s, a_i) \leftarrow Q_i(s, a_i) + \alpha \left(r + \gamma \max_{a'_i} Q_i(s', a'_i) - Q_i(s, a_i) \right), \quad (8.3)$$

where $r = R(s, \mathbf{a})$ is the observed received reward.

This approach results in considerable storage and computational savings due to the fact that the dimensionality of each player's Q_i -function remains constant with the growing number of agents. However, convergence hypotheses of single-agent Q-learning become invalid in multi-agent environments. For instance, the presence of multiple concurrent learners makes the environment non-Markovian from a single agent's perspective. Despite the lack of guaranteed convergence, it has been successfully applied to such domains as block-pushing problems [SEN 98], the control of a two-link rigid manipulator [BUS 06], air traffic flow management [TUM 07], and a distributed-air-jet micromanipulation system [MAT 09] and several others.

In fully cooperative games, each agent may assume that the other agents will learn to act in the best interest of the group. Under this optimistic assumption, Lauer and Riedmiller [LAU 00] showed that in certain stochastic games (such as the games with deterministic inter-state transitions) the Q_i -functions that are locally maintained by the ILs can yield a globally optimal joint policy. In other cases, when, for example, the transition function is non-deterministic, or when in one state of the environment, there can be more than one optimal action, it can happen that the joint behavior of the

ILs is not optimal (in the sense of Pareto-optimality). It is clearly an example of an equilibrium selection problem.

The central idea behind the optimistic ILs introduced by Lauer and Riedmiller [LAU 00] is to update the current policy π_i only if it is no longer optimal. Therefore, optimistic ILs neglect the penalties (low rewards) and update the evaluation of an action only if the new evaluation is greater than the previous one. To address the problem of equilibrium selection, Lauer and Riedmiller applied a coordination procedure to the update of the current player's policy π_i . This equilibrium selection mechanism is a form of social convention⁵, which places constraints on the possible action choices of the agents. The formal definition of this algorithm, called distributed Q-learning, is presented in Algorithm 8.5. Distributed Q-learning is proved to find optimal policies in deterministic fully cooperative SGs [LAU 00]. However this approach may not converge to optimal policies in stochastic environments.

Some authors were interested in varying the “degree of optimism” of the agents. In the lenient Q-learning algorithm [PAN 06], each IL has a time-dependent degree of lenience (or optimism). Indeed, at early stages of learning, being optimistic may be useful to identify promising actions. Moreover, if, at the earlier stages of the learning, the agents are actively exploring the environment, most selected actions are poor choices and ignoring penalties is therefore justified. But after the players have sufficiently explored the environment, it becomes interesting to achieve accurate estimates of utilities of actions. Experimental results show that lenient Q-learning outperforms distributed Q-learning in fully-cooperative stochastic environments, but the former was only designed for repeated (or single-state stochastic) games. Let us note here that another type of ILs, called hysteretic learners [MAT 07], use, in the update rule given by equation (8.3), two different values for the learning rate α . The first value is used to increase and the second one to decrease the Q_i -values [MAT 07]. Such agents are chiefly optimistic but are not completely blind to the penalties. This allows for taking into account the stochastic dynamics of the game.

Gradient ascent algorithms. In the framework of independent learners acting in general-sum stochastic games, a number of algorithms based on gradient ascent techniques have been proposed. We will begin by examining the first such algorithm, by [SIN 94], even if this algorithm does not verify the assumptions of action unobservability and game model unavailability made in this subsection. Further this will permit us to introduce two gradient ascent based algorithms for ILs.

Gradient ascent techniques (see Chapter 5) have been first studied in multi-agent learning by Singh *et al.* [SIN 94]. Their algorithm, IGA (for *Infinitesimal Gradient*

5. Conventions (or social laws) are restrictions on the possible action choices of agents [BOU 96].

Algorithm 8.5: Distributed Q-learning algorithm for fully-cooperative SGs and independent learner i , adapted from [LAU 00]

```

Initialize  $\gamma \in [0, 1)$ .
for all  $s \in S$  and  $a_i \in A_i$  do
  Initialize  $Q_i(s, a_i) \leftarrow \min_{s,a} R(s, a)$ .
  Initialize arbitrarily policy  $\pi_i^{a_i}(s)$  while being limited to legal probability
  distributions.

  Current state  $s \leftarrow s^0$ .
  repeat
    Choose action  $a_i$  using strategy  $\pi_i(s)$ .
    Play  $a_i$  with some exploration.
    Observe new state  $s'$  and received reward  $r_i$ .
     $q \leftarrow r_i + \gamma \max_{a'_i} Q_i(s', a'_i)$ 
    if  $q > Q_i(s, a_i)$  then
       $Q_i(s, a_i) \leftarrow q$ 
    if  $Q_i(s, \text{argmax}_{b_i \in A_i} \pi_i^{b_i}(s)) \neq \max_{b_i \in A_i} Q_i(s, b_i)$  then
      Select a random action  $a'_i \in \text{argmax}_{b_i \in A_i} Q_i(s, b_i)$ .
      for all  $b_i \in A_i$  do
         $\pi_i^{b_i}(s) \leftarrow \begin{cases} 1 & \text{if } b_i = a'_i \\ 0 & \text{otherwise.} \end{cases}$ 
    Update current state  $s \leftarrow s'$ .
     $n \leftarrow n + 1$ .
  until  $n > N$ 
return  $\pi_i(s) \forall s$ .

```

Ascent), assumes that each player in a repeated game performs a gradient ascent in the space of mixed strategies in order to find a strategy that maximizes its expected utility. This algorithm has been studied in the framework of a two-player two-actions-per-player matrix game, where the reward functions R_i of all players are known. The game is represented by two payoff matrices for the row and column players, r and c , as follows:

$$R_r = \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix},$$

$$R_c = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}.$$

Players r and c simultaneously choose an action in the set $A_{l,c} = \{1, 2\}$. The row player r plays action i and the column player c plays action j . The resulting payoffs are then, respectively, r_{ij} and c_{ij} .

Since this is a game with two actions per player, a mixed strategy can be represented with a single value. Let $\alpha \in [0, 1]$ be the probability for player r to play action 1 and $(1 - \alpha)$ the probability to play action 2. Similarly, let $\beta \in [0, 1]$ and $(1 - \beta)$ be the probabilities for player c to choose, respectively, actions 1 and 2. The expected utility of joint strategy $\pi = (\alpha, \beta)$ can then be calculated as follows:

$$\begin{aligned} u_r^{(\alpha, \beta)} &= r_{11}\alpha\beta + r_{22}(1 - \alpha)(1 - \beta) + r_{12}\alpha(1 - \beta) + r_{21}(1 - \alpha)\beta, \\ u_c^{(\alpha, \beta)} &= c_{11}\alpha\beta + c_{22}(1 - \alpha)(1 - \beta) + c_{12}\alpha(1 - \beta) + c_{21}(1 - \alpha)\beta. \end{aligned}$$

To estimate the effect of a change in the current policy, a player can calculate the partial derivative of its expected utility with respect to its mixed strategy:

$$\begin{aligned} \frac{\partial u_r^{(\alpha, \beta)}}{\partial \alpha} &= \beta u - (r_{22} - r_{12}), \\ \frac{\partial u_c^{(\alpha, \beta)}}{\partial \beta} &= \alpha u' - (c_{22} - c_{21}), \end{aligned}$$

where $u = (r_{11} + r_{22}) - (r_{21} + r_{12})$ and $u' = (c_{11} + c_{22}) - (c_{21} + c_{12})$.

At each time step, the IGA player adjusts its current strategy in the direction of the gradient so as to maximize its expected utility:

$$\begin{aligned} \alpha_{t+1} &= \alpha_t + \eta \frac{\partial u_r^{(\alpha_t, \beta_t)}}{\partial \alpha}, \\ \beta_{t+1} &= \beta_t + \eta \frac{\partial u_c^{(\alpha_t, \beta_t)}}{\partial \beta}, \end{aligned}$$

where η is the step size, typically $0 < \eta \ll 1$. Of course, the mixed strategy of the opponent is supposed to be known to the players.

Singh and colleagues have proven the convergence of IGA towards a Nash equilibrium or towards an equivalent average value – where both strategies keep on evolving while describing an elliptic trajectory when drawing the curve $x = \alpha_t$, $y = \beta_t$ – in self-play (i.e. when both players use the same algorithm) and this with a step size η that tends to 0 ($\lim_{\eta \rightarrow 0}$), hence name of the algorithm. The IGA algorithm is not meant for being used in a large number of real-world problems, for two main reasons:

- 1) it requires an omniscient knowledge of the other player's current strategy,
- 2) it has been designed for the “two players–two actions” case; there is no obvious direct extension to the “multiple payers–multiple actions” case.

The first “practical” algorithm empirically inheriting IGA’s convergence properties is *Policy Hill Climbing* (PHC), proposed by Bowling and Veloso [BOW 02a] in the framework of stochastic games. This algorithm can be adopted by ILs and does not require knowing the opponent’s current policy as IGA does. PHC essentially performs a local hill climbing in the space of mixed strategies. This is therefore a simple modification of the decentralized Q-learning making it capable of searching for a stochastic policy, i.e. mixed strategies in each state.

PHC is made of two parts: the first one is based on the decentralized Q-learning using equation (8.3) so as to maintain the utilities of simple actions (and not joint actions) in every state; the second one is a game-theoretic part that maintains current mixed strategies in each state. The current policy is adjusted by increasing the probability of choosing the action having the highest utility. This is achieved by using a small step δ . If δ is equal to 1, the algorithm becomes equivalent to the single-agent Q-learning, since the agent will then always execute the action with the highest utility. According to Bowling and Veloso, this algorithm results in a “rational” player because, when other players converge to stationary policies, its strategy converges to the best response to these strategies. Yet, if the other players are concurrently learning their own policies, PHC does not necessarily converge to a stationary policy although its average reward has been empirically observed to converge to the reward of a Nash equilibrium [BOW 02a]. The formal definition of the PHC algorithm is presented in Algorithm 8.6.

Bowling and Veloso [BOW 02a] have introduced the WoLF principle (for *win or learn fast*). This is a technique where agent i changes the size of its policy adjustment step δ depending on whether it “wins” or “loses”. If it is winning, a smaller value for δ is used; if the player is losing, a bigger value of the step is used. The authors then developed two important extensions for the IGA and PHC algorithms, respectively, called WoLF-IGA and WoLF-PHC. Bowling and Veloso proved formally that the WoLF principle guarantees the convergence of WoLF-IGA in self-play to a Nash equilibrium in any two-player two-actions-per-player repeated game. In practice, WoLF-PHC’s convergence has been observed in several “problematic” games. Intuitively, the WoLF principle helps the convergence by giving other players more time to adapt to changes in the player’s strategy. Reciprocally, it allows a player to adapt faster to changes in the strategies of other players when those changes are unfavorable.

More specifically, the WoLF-PHC algorithm (illustrated by Algorithm 8.7) works by applying the WoLF principle to the PHC algorithm. It uses two values of the

Algorithm 8.6: The PHC algorithm for SGs and independent learner i , adapted from [BOW 02a]

Initializations:

$$\alpha \in (0, 1], \delta \in (0, 1], \gamma \in [0, 1).$$

for all $s \in S$ and $a_i \in A_i$ **do**

$Q_i(s, a_i) \leftarrow 0.$	/* Current policy. */
$\pi_i^{a_i}(s) \leftarrow \frac{1}{ A_i }.$	/* Current state. */
$s \leftarrow s^0.$	

repeat

- (a) Choose action a_i using strategy $\pi_i(s)$ and play a_i with some exploration.
- (b) Observe new state s' and received reward r_i .
- (c) Update $Q_i(s, a_i)$ using the following rule:

$$Q_i(s, a_i) \leftarrow (1 - \alpha)Q_i(s, a_i) + \alpha \left(r_i + \gamma \max_{b_i \in A_i} Q_i(s', b_i) \right).$$

- (d) Update current strategy, $\pi_i(s)$, using the following rule:

if $a_i \neq \operatorname{argmax}_{b_i \in A_i} Q_i(s, b_i)$ **then**

$$\pi_i^{a_i}(s) \leftarrow \pi_i^{a_i}(s) - \delta_{sa_i}$$

otherwise

$$\pi_i^{a_i}(s) \leftarrow \pi_i^{a_i}(s) + \sum_{b_i \neq a_i} \delta_{sb_i}$$

with $\delta_{sa_i} = \min(\pi_i^{a_i}(s), \frac{\delta}{|A_i|-1})$ and while being limited to legal probability distributions.

- (e) Update current state $s \leftarrow s'$.

$$n \leftarrow n + 1.$$

until $n > N$

return $\pi_i(s) \forall s$.

step-size δ : δ_{lose} and δ_{win} . Whether agent i wins or loses is determined by comparing the expected value of the current policy, π_i with the expected value of an “average policy”, $\bar{\pi}_i$. This average policy is an online average of player i ’s policies since the beginning of the learning. If player i ’s expected value of the policy π_i is less than that of $\bar{\pi}_i$, such player is considered as losing and the value δ_{lose} is used during policy updates; otherwise δ_{win} is used.

Algorithm 8.7: The WoLF-PHC algorithm for SGs and independent learner i , adapted from [BOW 02a]

Initializations:
 $\alpha \in (0, 1]$, $\delta, \delta_{\text{win}}, \delta_{\text{loose}} \in (0, 1]$, $\gamma \in [0, 1)$.

for all $s \in S$ and $a_i \in A_i$ **do**

$Q_i(s, a_i) \leftarrow 0$ and $c(s) \leftarrow 0$.	$\pi_i^{a_i}(s) \leftarrow \frac{1}{ A_i }$. /* Current policy. */
	$\bar{\pi}_i^{a_i}(s) \leftarrow \frac{1}{ A_i }$. /* Average policy. */

$s \leftarrow s^0$. /* Current state. */

repeat

Steps (a)(b)(c) same as in PHC in Algorithm 8.6. Update the estimate of the average policy, $\bar{\pi}_i(s)$, as follows:	$c(s) \leftarrow c(s) + 1$, $\bar{\pi}_i^{b_i}(s) \leftarrow \bar{\pi}_i^{b_i}(s) + \frac{1}{c(s)} (\pi_i^{b_i}(s) - \bar{\pi}_i^{b_i}(s)), \quad \forall b_i \in A_i.$
---	---

Update the size of the step δ using the following rule:
if $\sum_{b_i \in A_i} \pi_i^{b_i}(s) Q_i(s, b_i) > \sum_{b_i \in A_i} \bar{\pi}_i^{b_i}(s) Q_i(s, b_i)$ **then**

$\delta \leftarrow \delta_{\text{win}}$

otherwise

$\delta \leftarrow \delta_{\text{loose}}$

Steps (d)(e) same as in PHC in Algorithm 8.6.
 $n \leftarrow n + 1$.

until $n > N$

return $\pi_i(s) \forall s$.

8.3.3. Complexity and scalability of multi-agent learning algorithms

This section puts forward the algorithmic complexity of some multi-agent learning algorithms and their scalability to large size problems.

As explained earlier, Shapley's algorithm [SHA 53] for (two-player zero-sum) stochastic games, which relies on Value Iteration, uses the minimax algorithm to find the equilibrium in each state and at each iteration. Since the execution time of the Value Iteration algorithm is itself polynomial in the size of the state space, then the execution time of Shapley's algorithm is polynomial in the number of players and in the number of states of the stochastic game.

The same reasoning can be applied for bounding the execution time of the algorithm of Kearns *et al.* [KEA 00]. This time is quadratic in the size of the state space assuming that function f (supposed to find a Nash equilibrium) executes in a time unit. However, as there is no known polynomial algorithm for finding a Nash equilibrium in an arbitrary normal form game, the execution time of FiniteVI that makes use of one of the known algorithms to calculate a Nash equilibrium cannot be polynomial either.

The analysis of the execution time of algorithms based on Q-learning is more complex. It is known [KOE 96] that the Q-learning's execution time can be exponential in the size of the state space, but this time complexity can be made polynomial if some restrictions on the reward model apply [KOE 96]. Moreover, in practice, the sizes of the state and joint action spaces often are themselves exponential in the number of agents. As a consequence, the execution time of an algorithm such as Nash-Q is not polynomial in the size of the state game matrix (since the process used to calculate a Nash equilibrium is itself not polynomial). Moreover, this time can be exponential in the number of states (because of the structure of the reward function used in the “Q-learning” part of this algorithm).

As a consequence, directly applying multi-agent algorithms for stochastic games to large size problems is problematic because of their generally high complexity. An exception is Shapley's algorithm, which has a polynomial execution time but only applies to two-player zero-sum games.

A good candidate for large size problems is Bowling's WoLF-PHC [BOW 02a]. Considering its structural complexity, its tolerant convergence conditions and its modest requirements relative to the information coming from the environment (in fact, WoLF-PHC agents need only perceive their states and rewards), a number of known function approximation techniques can be directly applied to this algorithm. In this context, Bowling [BOW 02b] proposed combining: (i) a technique called *Tile Coding* [SUT 98] to generalize the value function,⁶ (ii) a policy-gradient algorithm⁷ [SUT 00] as a basic learning method (instead of the Q-learning) and (iii) the WoLF principle to favor the convergence to a Nash equilibrium. This method made it possible to obtain an easily scalable multi-agent learning algorithm called GraWoLF [BOW 02b]. Bowling has experimentally shown that GraWoLF can be used for multi-agent learning in very large size problems, like the Goofspiel card game [BOW 02b] and the training of competing robots [BOW 03b] as in the RoboCup [KIT 97].

6. See Chapter 3.

7. See Chapter 5.

8.3.4. Beyond the search for an equilibrium

In the previously mentioned work, the main preoccupation was to search for one-stage (or stationary) equilibria (mainly Nash equilibria). Recently, researchers asked themselves about the necessity of the primordial requirement for a learning algorithm to converge to a stationary Nash equilibrium. There exist several reasons for that. First, there may be multiple equilibria in a game and there may be no method to coordinate the agents' choices. Furthermore, there exist no efficient algorithms to calculate such an equilibrium.

The third reason is that, in many games, as for example in the prisoners' dilemma, playing the stationary Nash equilibrium can be considered as a disaster for the agents, while the cooperative action, although not being a stationary equilibrium, would be a more appropriate choice in a long-term perspective. In other words, under certain assumptions playing the cooperative action can also be a "rational" (in a long-term sense) choice.

Finally, a player can prefer exploiting its opponent's weaknesses, if the opponent's play is not "perfect", instead of being in equilibrium.

8.3.4.1. Efficient play

Some recent work has stressed on playing "more efficiently" against some predefined type of opponents rather than converging to a stable value or policy, as usually done. Among these approaches, it is advisable to mention the following work [CHA 02, POW 05b, POW 05a, TES 04].

Chang and Kaelbling [CHA 02] were the first ones to propose a method allowing a player to exploit their opponent's learning algorithm if the latter has some specific form (in repeated games). In particular, their algorithm called "PHC-exploiter" can win more often than its opponent in zero-sum games, such as rock-paper-scissors, if the opponent uses the PHC algorithm [BOW 02a]. In fact, PHC-Exploiter is able to estimate the value of the learning step size δ of the opponent's PHC algorithm and to change its policy at the right time so as to exploit this information.

Tesauro [TES 04] has gone further. He has proposed a technique that seems more generic than that of Chang and Kaelbling. Indeed, Tesauro's Hyper- Q algorithm can play more efficiently in zero-sum games against a player whose algorithm has some known properties. For example, Tesauro experimentally showed that the Hyper- Q algorithm is more efficient than PHC and IGA in the rock-paper-scissors game. The idea underlying Tesauro's approach is the following. The Hyper- Q agent uniformly discretizes its opponent's strategy space. To estimate its opponent's strategy, it uses a probability distribution estimation technique. Finally, by interacting with the opponent, the Hyper- Q agent learns Q -values for

each triplet “own-strategy–opponent strategy estimate–simple action”. No extension of this algorithm to stochastic games has been proposed. An approach similar to that of Tesauro, called “ADL” (for *adaptive dynamics learning*) has been proposed by Burkov and Chaib-Draa [BUR 09]. The difference of the latter approach with Hyper- Q lies in the way Q -values are assigned; in ADL, they are assigned to histories of limited size instead of probability distributions. So, ADL’s Q -values have the form “game history–simple action”. These values are learned with the standard update rule of Q-learning.

8.3.4.2. Regret minimization

Another research direction [HAR 00, AUE 95, ZIN 03] aims at answering the following question. Given a game history, to what extent could the strategy executed by the agent (using a learning algorithm or a fixed policy) be improved? More precisely, the idea is to use the notion of “regret” that measures to what extent the performance demonstrated by an algorithm is worse than always playing the best stationary strategy given the observed history of opponents’ play.

Let us describe the notion of regret more formally and let restrict ourselves to repeated games. Let the vector $\mathbf{r}_i^t \in \mathbb{R}^{|A_i|}$ contain the rewards player i could obtain at time t by playing each of its actions, had it known the choices made by the other players at this time step. Let π_i^t be the strategy adopted by player i at time t . Then the expected utility u_i^t of strategy π_i^t is

$$u_i^t = \sum_{a_i \in A_i} \pi_i^{a_i, t} r_i^{a_i, t},$$

where $r_i^{a_i, t}$ denotes i ’s reward for action a_i according to \mathbf{r}_i^t . In vector form, this can be written as

$$u_i^t = \pi_i^t \cdot \mathbf{r}_i^t.$$

At time step t , agent i ’s regret \mathcal{R}_i^t for having played a policy π_i^t instead of a simple action a_i is the difference between both strategies’ payoffs, given the opponent’s choice of strategies:

$$\mathcal{R}_i^t(\pi_i^t, a_i) = r_i^{a_i, t} - u_i^t.$$

By denoting as 1_{a_i} a policy of agent i in which probability 1 is assigned to action a_i , the total regret \mathcal{R}_i^T of agent i for a sequence of game turns of length T is written as

$$\mathcal{R}_i^T = \max_{a_i \in A_i} \sum_{t=1}^T ((\mathbf{r}_i^t \cdot 1_{a_i}) - (\mathbf{r}_i^t \cdot \pi_i^t)).$$

The average regret $\bar{\mathcal{R}}_i$ for the same sequence can then be written as

$$\bar{\mathcal{R}}_i = \lim_{T \rightarrow \infty} \frac{1}{T} \mathcal{R}_i^T.$$

For a given algorithm, the property of having *no regret* in the sense of the average regret means that the average reward the agent receives using this algorithm is at least as high as what a fixed pure strategy can bring (the regret $\bar{\mathcal{R}}_i$ is non-positive whatever the opponent).

Zinkevich [ZIN 03] proposed an extension of the IGA algorithm [SIN 94] to games with more than two actions and two players. The author demonstrated that his algorithm, called GIGA (for *Generalized IGA*), has no regret in all general-sum games. The algorithm is similar to IGA and PHC. Agent i 's policy is updated as follows:

$$\pi_i^{t+1} = P(\pi_i^t + \eta \mathbf{r}_i^t).$$

The function $P(\tilde{\pi}_i)$ is a function that reduces the resulting policy $\tilde{\pi}_i$ from the sum $(\pi_i^t + \eta \mathbf{r}_i^t)$ to a legal probability distribution:

$$P(\tilde{\pi}_i) = \operatorname{argmin}_{\pi_i \in \Delta A_i} \|\tilde{\pi}_i - \pi_i\|,$$

where the $\|\cdot\|$ operator is the usual Euclidean norm. Zinkevich has shown that GIGA's total regret is bounded by

$$\mathcal{R}_i^T \leq \frac{(\max_{\pi_i, \pi'_i} \|\pi_i - \pi'_i\|)^2 \sqrt{T}}{2} + \left(\sqrt{T} - \frac{1}{2} \right) \left(\sup_{\pi_i, t=1, \dots, T} \|\mathbf{r}_i^t\| \right)^2.$$

Supposing that all rewards $r_i^{a_i}$ of player i are bounded by r_i^{\max} and accounting for the fact that $\max_{\pi_i, \pi'_i} \|\pi_i - \pi'_i\| = \sqrt{2}$, we can write that

$$\mathcal{R}_i^T \leq \sqrt{T} + \left(\sqrt{T} - \frac{1}{2} \right) |A_i| (r_i^{\max})^2.$$

Using l'Hôpital's rule⁸, we find that

$$\lim_{T \rightarrow \infty} \frac{1}{T} \left(\sqrt{T} + \left(\sqrt{T} - \frac{1}{2} \right) |A_i| (r_i^{\max})^2 \right) = 0,$$

which leads us to conclude that $\bar{\mathcal{R}}_i \leq 0$ and, therefore, GIGA has no regret in the sense of the average regret.

8. This rule is defined as follows: if f and g are two functions differentiable in a , null in a and such that the quotient $\frac{f'(a)}{g'(a)}$ is defined, then $\lim_{x \rightarrow a} \frac{f(x)}{g(x)} = \frac{f'(a)}{g'(a)}$.

Bowling [BOW 05] in turn has shown that WoLF's principle applied to GIGA lets GIGA converge to an equilibrium in self-play. His WoLF-GIGA then inherits GIGA's property of no-regret and converges to a Nash equilibrium in mixed strategies in two-player two-actions-per-player general-sum games.

8.3.4.3. Metastrategies

Another point of view on learning for repeated games has been proposed by Powers and Shoham [POW 05b, POW 05a]. Their approach guarantees different properties depending on the opponent's class:

- *Targeted Optimality*: if the opponent belongs to a target class, the player provides the best response;
- *Compatibility*: against itself (in self-play), the player finds a Nash equilibrium that is not Pareto dominated by another Nash equilibrium;
- *Safety*: against any other opponent, the player guarantees its safety level in the game.

The proposed algorithms (Metastrategy [POW 05b] and Manipulator [POW 05a]) both start by a phase of identification of the opponent's class, before choosing the best “expert” algorithm to use against it.

It is instructive to observe that, in repeated games, we passed successively:

- from hand-written strategies (Tit-for-Tat, etc.),
- to algorithms learning how to adapt when facing “fixed” strategies,
- then to algorithms able to adapt to learning opponents,
- and here to meta-algorithms selecting an expert algorithm depending on the identified type of opponent.

The question that comes naturally is: What happens when a meta-algorithm meets another one? Do we need meta-meta-heuristics?

8.3.5. Discussion

It is interesting to see how different previously introduced algorithms compare together. To that end, Table 8.1 summarizes different characteristics of the multi-agent algorithms seen in this chapter. This table is inspired from the work of Aras [DUT 06].

In this table, all the algorithms introduced in this chapter are placed relatively to their requirements about game properties. The $T + R_i$ column indicates whether the transition and reward functions are supposed to be known by the agents or if they are observed through experience. The \mathbf{A}_{-i} and \mathbf{R}_{-i} columns indicate whether the agents

Algorithm	Game properties				Equ. type	All games	Theoretical properties
	$T + R_i$	\mathbf{R}_{-i}	\mathbf{A}_{-i}	π_{-i}			
[SHA 53]	known	known	known	known	Nash	no	yes
[KEA 00]	known	known	known	known	Nash	yes	yes
Fictitious Play	known	observed	observed	model	Nash	no	yes
Minimax- Q	observed	observed	observed	no	Nash	no	yes
Nash- Q	observed	observed	observed	no	Nash	no	yes
Friend- Q	observed	no	observed	no	Nash	no	yes
FoF- Q	observed	no	observed	no	Nash	no	yes
JAL / OMQ	observed	no	observed	model	Nash	no	no
Adaptive Play Q	observed	no	observed	model	Nash	no	no
Hyper- Q	observed	no	observed	model	?	?	no
ADL	observed	no	observed	model	?	?	no
GIGA-WoLF	observed	no	observed	no	Nash	no	yes
Decentralized- Q	observed	no	no	no	Nash	yes	no
Distributed- Q	observed	no	no	no	Nash	no	yes
WoLF-PHC	observed	no	no	no	Nash	yes	no
Manipulator	observed	no	no	no	?	?	yes
MetaStrategy	observed	no	no	no	?	?	yes

Table 8.1. A comparison of the mentioned algorithms

are supposed to know in advance or to observe other players' actions and rewards. The π_{-i} column indicates whether the agents know or model the policies played by their opponents. "No" in the cells means that the agents cannot know or observe some properties of the game. The "Equ. type" and "All games" columns indicate whether a Nash equilibrium is reached by each algorithm and whether this fact is valid for any game, or if some restrictions apply (as, e.g. the fact that the game must be zero-sum, or that only two-actions–two-players are allowed). The "Theoretical property" column indicates whether some theoretical properties of the algorithms have been proved.

Note that question marks in Table 8.1's cells mean that this aspect has not been well studied and/or understood in the literature. As some mentioned algorithms are rather recent, there remains a lot of work to do in a view to explain their behavior in various situations and to study their convergence properties, their complexity and so on.

Another interesting classification of (repeated or stochastic) game algorithms has been proposed by Chang and Kaelbling [CHA 02]. They propose a classification of the algorithms via the cross product of possible strategies and possible beliefs over the opponents' strategies. In this context, agent's possible strategies are classified depending on the length of the history kept in memory, from \mathcal{H}_0 to \mathcal{H}_∞ . Of course, having more memory, agents can work out more complex and “clever” strategies. In the same vein, an agent can classify its opponents according to the same principle. For instance, if an agent believes that its opponent has no memory, it puts it in \mathcal{B}_0 . On the other hand, if it believes that its opponent's memory is unbounded, then it puts it in \mathcal{B}_∞ . The classification proposed by Chang and Kaelbling, adapted to reflect some of the algorithms mentioned in this chapter, is presented in Table 8.2. Let us note that the “Bully”, “Godfather” and “Multiplicative weights” algorithms have not been discussed in this chapter. For more details, the reader can refer to the following documents [LIT 01a, FRE 99].

	\mathcal{B}_0	\mathcal{B}_1	\mathcal{B}_∞
\mathcal{H}_0	Minimax- Q , Nash- Q		Bully
\mathcal{H}_1			Godfather
\mathcal{H}_∞	Q-learning, (WoLF)-PHC, Fictitious Play	ADL with $ h = 1$	Multiplicative Weights

Table 8.2. The algorithm classification proposed by Chang and Kaelbling and adapted from [CHA 02] for the algorithms presented in this chapter

8.4. Conclusion and outlook

In this chapter, we have presented an overview of game theory and its application to decision making in multi-agent environments. To that end, normal form games and dynamic games have first been discussed, as models of interaction between rational agents. These models have then been extended to the multi-state world by combining them with MDPs. This led to the emergence of a powerful model, called “stochastic games”, which makes it possible to describe complex temporally extended real-world interactions.

Several multi-agent algorithms based on the formal model of stochastic games have then been presented and discussed. We have observed that some of them rely on a good theoretical basis while others have only been studied experimentally. As this research field is rather young and goes through a period of rapid growth, it can be expected that, in the near future, certain of these algorithms will be properly analyzed from a theoretical point of view, and a number of new interesting approaches will be proposed.

In addition to the missing theoretical bases, there exist other problems that need to be solved so as to make these algorithms applicable for real-world problems. First,

the algorithms' complexity remains high. In fact, this problem is hidden in the model of stochastic games itself whose number of internal variables grows exponentially with the number of agents. Moreover, some algorithms require perceiving the joint actions of the other players, which is not always a realistic hypothesis. This adds up to the problem induced by the existence of multiple equilibria, and for which the coordination on a single coordinated choice is not obvious for now.

We should note that this chapter has not addressed another research direction in the field of decision making under uncertainty in multi-agent environments: Partially Observable Stochastic Games (POSG). In this model, normal form games are combined with the POMDP model, which allows solving problems where agents have a partial view of the environment's state. Good examples of this problem and of approaches proposed for their resolution can be found in [HAN 04, EME 04].

This chapter has addressed a number of works focusing on fully cooperative stochastic games, meaning that all agents have the same reward function. Other formalisms and approaches will be presented in Chapter 9, for example, MMDPs, MTDPs or DEC-POMDPs. The later model, that of decentralized POMDPs, can be seen as the POSG model with $R_i = R_j$ for all i, j . Specific algorithms have been developed that exploit this cooperation. In particular, it is common – though not systematic – for these algorithms to be centralized [BER 02].

Finally, we have to be aware that a tool, called GAMUT, has been developed to facilitate the experimental evaluation of algorithms in the field of game theory. The article [NUD 04] makes a complete presentation of this tool.

Acknowledgment

The authors wish to thank Michael Bowling for his help on the technical part about GraWoLF.

8.5. Bibliography

- [AUE 95] AUER P., CESA-BIANCHI N., FREUND Y. and SCHAPIRE R. E., “Gambling in a rigged casino: The adversarial multi-armed bandit problem”, *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pp. 322–331, 1995.
- [AUM 02] AUMANN R. J. and HART S., Eds., *Handbook of Game Theory with Economic Applications*, vol. 3, Elsevier Science, Netherlands, 2002.
- [BER 02] BERNSTEIN D., GIVAN R., IMMERMAN N. and ZILBERSTEIN S., “The complexity of decentralized control of Markov decision processes”, *Mathematics of Operations Research*, vol. 27, no. 4, pp. 819–840, 2002.
- [BOU 96] BOUTILIER C., “Planning, learning and coordination in multiagent decision processes”, *Proceedings of the 6th Conference on Theoretical Aspects of Rationality and Knowledge (TARK'96)*, Morgan Kaufmann, San Francisco, CA, pp. 195–201, 1996.

- [BOW 02a] BOWLING M. and VELOSO M., “Multiagent learning using a variable learning rate”, *Artificial Intelligence*, vol. 136, no. 2, pp. 215–250, 2002.
- [BOW 02b] BOWLING M. and VELOSO M., “Scalable learning in stochastic games”, *Proceedings of the AAAI-2002 Workshop on Game Theoretic and Decision Theoretic Agents*, Edmonton, Canada, 2002.
- [BOW 03a] BOWLING M., Multiagent learning in the presence of agents with limitations, PhD thesis, University of Toronto, 2003.
- [BOW 03b] BOWLING M. and VELOSO M., “Simultaneous adversarial multi-robot learning”, *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*, Acapulco, Mexico, pp. 699–704, 2003.
- [BOW 05] BOWLING M., “Convergence and no-regret in multiagent learning”, *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*, Vancouver, Canada, pp. 209–216, 2005.
- [BRO 51] BROWN G. W., “Iterative solution of games by fictitious play”, *Activity Analysis of Production and Allocation*, Cowles Commission Monograph, no. 13, John Wiley & Sons, New York, pp. 374–376, 1951.
- [BUR 09] BURKOV A. and CHAIB-DRAA B., “Effective learning in the presence of adaptive counterparts”, *Journal of Algorithms*, vol. 64, no. 4, pp. 127–138, 2009, doi:10.1016/j.jalgor.2009.04.003.
- [BUS 06] BUSONIU L., BABUSKA R. and DE SCHUTTER B., “Decentralized reinforcement learning control of a robotic manipulator”, *Proceedings of the 9th International Conference on Control, Automation, Robotics and Vision (ICARCV 2006)*, Singapore, pp. 1347–1352, 2006.
- [CHA 02] CHANG Y. and KAEHLING L. P., “Playing is believing: The role of beliefs in multi-agent learning”, *Advances in Neural Information Processing Systems 14 (NIPS'01)*, MIT Press, Cambridge, MA, pp. 1483–1490, 2002.
- [CLA 98] CLAUS C. and BOUTILIER C., “The dynamics of reinforcement learning in cooperative multiagent systems”, *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI'98)*, AAAI Press, Menlo Park, CA, pp. 746–752, 1998.
- [DUT 06] DUTECH A., ARAS R. and CHARPILLE F., “Apprentissage par renforcement et théorie des jeux pour la coordination des systèmes multi-agents”, *Colloque Africain pour la Recherche en Informatique (CARI'06)*, Cotonou, Benin, 2006.
- [EME 04] EMERY-MONTEMERLO R., GORDON G., SCHNEIDER J. and THRUN S., “Approximate solutions for partially observable stochastic games with common payoffs”, *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'04)*, New York, pp. 136–143, 2004.
- [FIN 64] FINK A. M., “Equilibrium in a stochastic n -person game”, *Journal of Science in Hiroshima University Series*, vol. 28, pp. 89–93, 1964.
- [FRE 99] FREUND Y. and SCHAPIRE R. E., “Adaptive game playing using multiplicative weights”, *Games and Economic Behavior*, vol. 29, pp. 79–103, 1999.
- [FUD 91] FUDENBERG D. and TIROLE J., *Game Theory*, MIT Press, Cambridge, MA, 1991.

- [FUD 99] FUDENBERG D. and LEVINE D. K., *The Theory of Learning in Games*, MIT Press, Cambridge, MA, 1999.
- [GEN 00] GENTIS H., Ed., *Game Theory Evolving: A Problem-Centered Introduction to Modeling Strategic Interaction*, Princeton University Press, Princeton, NJ, 2000.
- [GIE 06] GIES O. and CHAIB-DRAA B., “Apprentissage de la coordination multiagent: une méthode basée sur le Q-learning par jeu adaptatif”, *Revue d’Intelligence Artificielle*, vol. 20, no. 2-3, pp. 385–412, 2006.
- [GRÄ 02] GRÄDEL E., THOMAS W. and WILKE T., Eds., *Automata, Logics and Infinite Games*, vol. 2500 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, 2002.
- [HAN 04] HANSEN E. A., BERNSTEIN D. S. and ZILBERSTEIN S., “Dynamic programming for partially observable stochastic games”, *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI’04)*, San Jose, CA, pp. 709–715, 2004.
- [HAR 00] HART S. and MAS-COLELL A., “A simple adaptive procedure leading to correlated equilibrium”, *Econometrica*, vol. 68, no. 5, pp. 1127–1150, 2000.
- [HAU 98] HAURIE A. and KRAWCZYK J. B., *An introduction to dynamic games*, Faculty of Economics and Social Sciences, University of Geneva, Geneva, Switzerland, 1998, Handouts.
- [HOF 66] HOFFMAN A. J. and KARP R. M., “On nonterminating stochastic games”, *Management Science*, vol. 12, no. 5, pp. 359–370, 1966.
- [HU 03] HU J. and WELLMAN M. P., “Nash Q-learning for general-sum stochastic games”, *Journal of Machine Learning Research*, vol. 4, pp. 1039–1069, 2003.
- [KEA 00] KEARNES M., MANSOUR Y. and SINGH S., “Fast planning in stochastic games”, *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI’00)*, Morgan Kaufman, San Francisco, CA, pp. 309–316, 2000.
- [KIT 97] KITANO H., ASADA M., KUNIYOSHI Y., NODA I. and OSAWA E., “RoboCup: The robot world cup initiative”, *Proceedings of the 1st International Conference on Autonomous Agents (Agents’97)*, ACM Press, New York, pp. 340–347, 1997.
- [KOE 96] KOENIG S. and SIMMONS R. G., “The effect of representation and knowledge on goal-directed exploration with reinforcement-learning algorithms”, *Machine Learning*, vol. 22, pp. 227–250, 1996.
- [LAU 00] LAUER M. and RIEDMILLER M., “An algorithm for distributed reinforcement learning in cooperative multi-agent systems”, *Proceedings of the 17th International Conference on Machine Learning*, Morgan Kaufmann, San Francisco, CA, pp. 535–542, 2000.
- [LIT 94] LITTMAN M. L., “Markov games as a framework for multi-agent reinforcement learning”, *Proceedings of the 11th International Conference on Machine Learning (ICML’94)*, Morgan Kaufmann, San Francisco, CA, pp. 157–163, 1994.
- [LIT 01a] LITTMAN M. and STONE P., “Leading best-response strategies in repeated games”, *IJCAI’01 Workshop on Economic Agents, Models, and Mechanisms*, Seattle, WA, 2001.

- [LIT 01b] LITTMAN M. L., “Friend-or-foe Q-learning in general-sum games”, *Proceedings of the 18th International Conference on Machine Learning*, Morgan Kaufmann, San Francisco, CA, pp. 322–328, 2001.
- [LIT 01c] LITTMAN M. L., “Value-function reinforcement learning in Markov games”, *Journal of Cognitive Systems Research*, vol. 2, pp. 55–66, 2001.
- [MAR 75] MARTIN D. A., “Borel determinacy”, *Annals of Mathematics*, vol. 102, pp. 363–371, 1975.
- [MAT 07] MATIGNON L., LAURENT G. J. and LE FORT-PIAT N., “Hysteretic Q-learning: an algorithm for decentralized reinforcement learning in cooperative multi-agent teams”, *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, pp. 64–69, 2007.
- [MAT 09] MATIGNON L., LAURENT G. J. and LE FORT-PIAT N., “Design of semi-decentralized control laws for distributed-air-jet micromanipulators by reinforcement learning”, *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, 2009.
- [MYE 97] MYERSON R. B., Ed., *Game Theory: Analysis of Conflict*, Harvard University Press, Boston, MA, 1997.
- [NAS 51] NASH J. F., “Non-cooperative games”, *Annals of Mathematics*, vol. 54, pp. 286–295, 1951.
- [NEU 28] VON NEUMANN J., “Zur Theorie der Gesellschaftsspiele”, *Mathematische Annalen*, vol. 100, no. 1928, pp. 295–320, 1928.
- [NUD 04] NUDELMAN E., WORTMAN J., SHOHAM Y. and LEYTON-BROWN K., “Run the GAMUT: A comprehensive approach to evaluating game-theoretic algorithms”, *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS’04)*, New York, pp. 880–887, 2004.
- [OSB 04] OSBORNE M. J., Ed., *An Introduction to Game Theory*, Oxford University Press, Oxford, 2004.
- [PAN 06] PANAIT L., SULLIVAN K. and LUKE S., “Lenient learners in cooperative multiagent systems”, *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS’06)*, ACM Press, New York, pp. 801–803, 2006.
- [PAP 95] PAPADIMITRIOU C. H., “Algorithms, games, and the Internet”, *Proceedings of the Annual ACM Symposium on Theory of Computing (STOC’91)*, ACM Press, New York, pp. 749–753, 1995.
- [PAR 02] PARSONS S., GMYTRASIEWICZ P. and WOOLWRIDGE M., Eds., *Game Theory and Decision Theory in Agent-Based Systems*, vol. 5 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, Kluwer Academic Publishers, Boston, MA, 2002.
- [POL 69] POLLATSCHKE M. A. and AVI-ITZHAK B., “Algorithms for stochastic games with geometrical interpretation”, *Management Science*, vol. 15, no. 7, pp. 399–415, 1969.
- [POW 05a] POWERS R. and SHOHAM Y., “Learning against opponents with bounded memory”, *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI’05)*, Edinburgh, Scotland, pp. 817–822, 2005.

- [POW 05b] POWERS R. and SHOHAM Y., “New criteria and a new algorithm for learning in multi-agent systems”, SAUL L. K., WEISS Y. and BOTTOU L., Eds., *Advances in Neural Information Processing Systems 17 (NIPS'04)*, MIT Press, Cambridge, MA, pp. 1089–1096, 2005.
- [RUS 95] RUSSELL S. and NORVIG P., *Artificial Intelligence: A Modern Approach*, Prentice Hall Series in Artificial Intelligence, Prentice Hall, Englewood Cliffs, NJ, 1995.
- [SEN 98] SEN S. and SEKARAN M., “Individual learning of coordination knowledge”, *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 10, no. 3, pp. 333–356, 1998.
- [SHA 53] SHAPLEY L. S., “Stochastic games”, *Proceedings of the National Academy of Sciences of the United States of America (PNAS)*, vol. 39, pp. 1095–1100, 1953.
- [SIN 94] SINGH S., KEARNS M. and MANSOUR Y., “Nash convergence of gradient dynamics in general-sum games”, *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI'94)*, Morgan Kaufman, San Francisco, CA, pp. 541–548, 1994.
- [SMI 02] SMITH J. M., Ed., *Evolution and the Theory of Games*, Cambridge University Press, Cambridge, 2002.
- [STO 00] STONE P. and VELOSO M., “Multiagent systems: a survey from a machine learning perspective”, *Autonomous Robots*, vol. 8, no. 3, pp. 345–383, 2000.
- [SUT 98] SUTTON R. S. and BARTO A. G., *Reinforcement Learning: An Introduction*, Bradford Book, MIT Press, Cambridge, MA, 1998.
- [SUT 00] SUTTON R., MCALLESTER D., SINGH S. and MANSOUR Y., “Policy gradient methods for reinforcement learning with function approximation”, *Advances in Neural Information Processing Systems 12 (NIPS'99)*, MIT Press, Cambridge, MA, pp. 1057–1063, 2000.
- [TAN 93] TAN M., “Multiagent reinforcement learning: independent vs. cooperative agents”, *Proceedings of the 10th International Conference on Machine Learning (ICML'93)*, Amherst, MA, pp. 330–337, 1993.
- [TES 04] TESAURO G., “Extending Q-learning to general adaptive multi-agent systems”, THRUN S., SAUL L. and SCHÖLKOPF B., Eds., *Advances in Neural Information Processing Systems 16 (NIPS'03)*, MIT Press, Cambridge, MA, pp. 871–878, 2004.
- [THI 04] THISSE J.-F., *Théorie des jeux : une introduction*, Catholic University of Leuven, Department of Economics, 2004, course notes.
- [TUM 07] TUMER K. and AGOGINO A., “Distributed agent-based air traffic flow management”, *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'07)*, ACM Press, New York, pp. 1–8, 2007.
- [UTH 03] UTHER W. and VELOSO M., Adversarial reinforcement learning, Report no. CMU-CS-03-107, School of Computer Science, Carnegie Mellon University, 2003.
- [VRI 87] VRIEZE O. J., *Stochastic games with finite state and action spaces*, Centrum voor Wiskunde en Informatica, Amsterdam, Netherlands, 1987.
- [YIL 03] YILDIZOGLU M., Ed., *Introduction à la Théorie des Jeux*, Dunod, Paris, 2003.

- [YOU 93] YOUNG H. P., “The evolution of conventions”, *Econometrica*, vol. 61, no. 1, pp. 57–84, 1993.
- [YOU 98] YOUNG H. P., *Individual Strategy and Social Structure: An Evolutionary Theory of Institutions*, Princeton University Press, Princeton, NJ, 1998.
- [ZIN 03] ZINKEVICH M., “Online convex programming and generalized infinitesimal gradient ascent”, *Proceedings of the 20th International Conference on Machine Learning (ICML'03)*, Washington DC, pp. 928–936, 2003.
- [ZIN 06] ZINKEVICH M., GREENWALD A. and LITTMAN M. L., “Cyclic equilibria in Markov games”, *Advances in Neural Information Processing Systems 18 (NIPS'05)*, MIT Press, Cambridge, MA, pp. 1641–1648, 2006.

Chapter 9

DEC-MDP/POMDP

9.1. Introduction

MDPs and POMDPs are both mathematical models that have been successfully used to formalize sequential decision-theoretic problems under uncertainty. They allow an agent to decide how to act in a stochastic environment in order to maximize a given performance measure. The agent's decision is based on a complete (MDPs) or partial (POMDPs) observability of the environment. It has been proved that these models are efficient tools for solving problems of mono-agent control and they have been successfully applied to many domains such as mobile robots [BER 01], spoken dialog managers [ROY 00] or inventory management [PUT 94]. This encourages us to consider the extension of these models to cooperative multiagent systems.

EXAMPLE 9.1. Let us go back to the car maintenance example introduced before (see Chapter 1, section 1.1). We now consider several garage employees, who do not always coordinate, who are responsible for repairing and maintaining cars. We can imagine a futuristic scenario where the garage is equipped with a set of robots, each one dedicated to a specific task: one for brakes, one for tires, another one for oil, etc. Thus, several agents may decide to act simultaneously on the same car (or on the same part of the car) while possibly having different and partial knowledge about the underlying state of the car. One key question then is: how can the agents cooperate in order to optimize the outcome of their joint action (which results from the simultaneous execution of the agents' individual actions)? The current chapter presents extensions of the MDP formalism that allow for modeling such a decision problem.

Chapter written by Aurélie BEYNIER, François CHARPILLET, Daniel SZER and Abdel-Illah MOUADDIB.

The models that are developed in this chapter rely on different types of hypotheses that can be classified within: i) each agent has a complete knowledge of the system state; ii) each agent has a partial knowledge of the system state; iii) the agents can communicate; iv) the agents cannot (or can partially) communicate; etc.

These hypotheses have led to several formalisms. Among them, we review the most well-known ones: MMDP, Dec-MDP, Dec-POMDP, MTDP, Dec-MDP-Com, COM-MTDP, ND-POMDP, TI-Dec-MDP, OC-Dec-MDP. This chapter also deals with the complexity of computing optimal solutions for the multiagent decision problems described with these formalisms. Indeed, decentralized decision problems suffer from a high complexity which makes the computation of an optimal solution very difficult. We show that the structure of the problem domain, such as the locality of interactions, can sometimes be exploited to overcome this complexity barrier. We also present approximate algorithms that have been developed to deal with more complex problems. It has been proved that some of them converge to local optima and reach special kinds of equilibrium solutions.

9.2. Preliminaries

The observability of a system describes the information available to an agent (or a group of agents) for its decision-making. As explained in Chapter 7, a system is said to be *partially observable* if an agent or a group of agents does not have access to all the information required to make perfect decisions. The problem is then how agents can act optimally in a cooperative manner despite the lack of information. On the other hand, if the agents have full access to all information relevant to optimal decision-making, the system is said to be *fully observable*.

It has been shown that the degree of observability has a big impact on the complexity of decision-making. Therefore, solving a POMDP is much more difficult than solving an MDP. This remains true in the case of multiple decision-makers. In this section, we define several notions related to the observability of a multiagent system that will later be used in this chapter.

In multiagent systems we distinguish between *joint observability* and *local observability* (or individual observability) of the system state (environment + agents). By joint observability we mean the combination of the individual observations of all agents. If joint observability allows the agents to deduce the state of the system, this state is said to be *jointly fully observable*. Otherwise, some features of the system state are not observed by the agents and the system state is *jointly partially observable*.

The system state is said to be *locally (fully) observable* if each agent alone observes all the features of the system state. In such a case, the system state is also jointly fully observable. Similarly, if at least one agent fully observes the system state, this

state is also jointly fully observable. When dealing with blind agents that have no observability of the system, the system is said to be *non-observable* (locally or jointly). In mono-agent settings, local observability of the system state distinguishes MDPs from POMDPs (see Chapter 7). In multiagent settings, more classes of observability exist. Figure 9.1 presents the relationships between the different kinds of observability about the system state in multiagent systems.

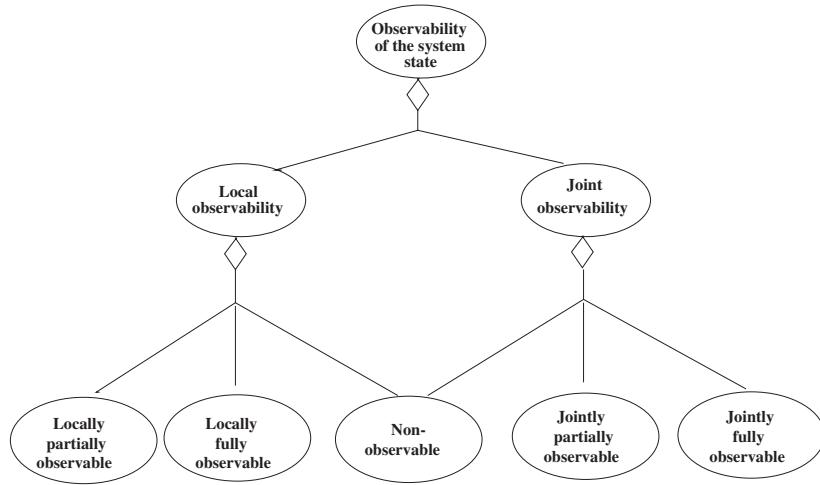


Figure 9.1. Observability of the system state

9.3. Multiagent Markov decision processes

In order to extend MDPs to multiagent settings, Boutilier [BOU 96, BOU 99a, BOU 99b] has introduced Multiagent Markov Decision Processes (MMDPs). MMDPs allow for representing sequential decision-making problems in cooperative multiagent settings. This formalism is very much related to models dealing with stochastic games [SHA 53], as presented in Chapter 8. However, MMDPs (and the approaches that are presented in the present chapter) only model cooperative systems. The reward function is thus a common function the agents must maximize, whereas most stochastic games consider separate reward functions and thus different, possibly conflicting, goals for each agent.

Since MMDPs derive from standard Markov decision processes, an MMDP is defined by a set of states \mathcal{S} , a set of actions \mathcal{A} , a transition function \mathcal{T} and a reward function \mathcal{R} . In order to model the multiagent decision problem, each element a of \mathcal{A} is a “joint action” which is defined as a set of individual actions, one for each agent. Since several agents are considered, a component n , which denotes the number of agents, is added to the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$.

DEFINITION 9.1 (multiagent Markov decision processes). *An MMDP is defined as a tuple $\langle n, \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ where:*

- n is the number of agents \mathcal{A}_i of the system, $i \in \{1, \dots, n\}$;
- \mathcal{S} is the set of states s of the system;
- $\mathcal{A} = \mathcal{A}_1 \times \dots \times \mathcal{A}_n$ denotes the set of joint actions, where \mathcal{A}_i is the set of individual actions of agent \mathcal{A}_i ;
- \mathcal{T} is the transition function; it gives the probability $\mathcal{T}(s, a, s')$ that the system moves to state s' when the agents execute the joint action $a \in \mathcal{A}$ from state s ;
- \mathcal{R} is the reward function; $\mathcal{R}(s, a, s')$ is the reward obtained by the system when the system state changes from s to s' under the influence of joint action a .

An MMDP can then be viewed as a large MDP. The set of agents is effectively represented as a single agent whose aim is to optimize a policy for an MDP where each action is in fact a joint action (see Chapter 1).

Solving an MMDP consists of finding a joint policy $\pi = \langle \pi_1, \dots, \pi_n \rangle$, where π_i is the individual policy of agent \mathcal{A}_i . This is a function $\pi_i : \mathcal{S} \rightarrow \mathcal{A}_i$ which maps each state of the system to an action a_i of the agent \mathcal{A}_i . Since an MMDP can be viewed as a large MDP, techniques for optimally solving MDPs, like policy iteration or value iteration, can be used to solve MMDPs (see Chapter 1).

Note that the MMDP model does not consider individual observations. In order to execute its policy, each agent must therefore know the state of system. In many multiagent systems, this property does not hold. Two approaches can then be considered: i) a central entity knows the system state and makes decisions for all the agents; ii) the agents communicate their observations. The last approach can be used if and only if the system state is jointly fully observable and communication is free and instantaneous. Thus, after each action, each agent communicates to the other agents its own observation. After communication, each agent can therefore deduce the system state and is able to decide which action it must execute. Unfortunately, these assumptions seldom hold in multiagent systems. Most of the time, communication is not free and the agents cannot jointly observe the system state.

9.4. Decentralized control and local observability

Decentralized Partially Observable Markov Decision Processes (DEC-POMDPs) and Decentralized Markov Decision Processes (DEC-MDPs) extend POMDPs and MDPs to multiagent decentralized control. These models allow agents to act in a fully decentralized manner while maximizing the performance of the system as a whole. Unlike MMDPs, DEC-POMDPs and DEC-MDPs do not assume complete individual observability of the system state.

In the following section we introduce the DEC-MDP and DEC-POMDP models. Algorithms for solving problems formalized as DEC-POMDPs or DEC-MDPs are described in section 9.6.

9.4.1. Decentralized Markov decision processes

In a general decentralized control problem, agents may not always be able to observe the true state of the environment, nor the current configuration of the other agents. Often, agents must therefore base their decisions on a partial view of the system. This is why DEC-MDPs and DEC-POMDPs define a set of observations Ω , which consists of the individual observation sets Ω_i of each agent $\mathcal{A}_{\mathcal{G}_i}$. The observation function \mathcal{O} then gives the probability for agent $\mathcal{A}_{\mathcal{G}_i}$ to observe o_i when the system is in state s , the agents execute joint action a and the system moves to the state s' .

DEFINITION 9.2 (decentralized partially observable Markov decision processes). A DEC-POMDP for n agents is defined as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \Omega, \mathcal{O}, \mathcal{R} \rangle$ where:

- \mathcal{S} is a finite set of system states;
- $\mathcal{A} = \langle \mathcal{A}_1, \dots, \mathcal{A}_n \rangle$ is a set of joint actions; \mathcal{A}_i is the set of actions a_i that can be executed by agent $\mathcal{A}_{\mathcal{G}_i}$;
- $\mathcal{P} = \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is a transition function; $\mathcal{P}(s, a, s')$ is the probability of the outcome state s' when the agents execute the joint action a from s ;
- $\Omega = \Omega_1 \times \Omega_2 \times \dots \times \Omega_n$ is a finite set of observations, where Ω_i is $\mathcal{A}_{\mathcal{G}_i}$'s set of observations;
- $\mathcal{O} = \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \Omega \rightarrow [0, 1]$ is the observation function; $\mathcal{O}(s, a, s', o = \langle o_1, \dots, o_n \rangle)$ is the probability that each agent $\mathcal{A}_{\mathcal{G}_i}$ observes o_i when the agents execute the joint action a from state s and the system moves to state s' ;¹
- \mathcal{R} is a reward function; $\mathcal{R}(s, a, s')$ is the reward the system obtains when the agents execute joint action a from state s and the system moves to state s' .

Similarly to POMDPs, the definition of a DEC-POMDP problem includes a problem horizon T , which specifies the number of decision steps that are taken into account. As in single-agent problems, finite-horizon or infinite-horizon problems can be considered. Note that a single-agent DEC-POMDP is a POMDP.

If the state of the system is jointly fully observable, the DEC-POMDP is a DEC-MDP.

1. This definition allows for modeling individual observations which are correlated to each other.

DEFINITION 9.3 (decentralized Markov decision processes). A DEC-MDP is a special kind of DEC-POMDP where the system state is jointly observable. This property can be formalized as follows:

$$\text{If } \mathcal{O}(s, a, s', o = \langle o_1, \dots, o_n \rangle) > 0, \text{ then } \Pr(s' | \langle o_1, \dots, o_n \rangle) = 1.$$

Note that this property does not imply that each agent separately can observe the full system state. Thus, a DEC-MDP has the same components as a DEC-POMDP.

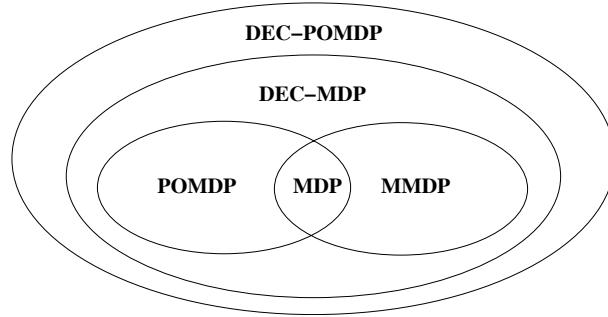


Figure 9.2. Relationships between the different types of Markovian processes

For each possible transition, the reward function \mathcal{R} of a DEC-POMDP (or a DEC-MDP) gives the reward which is obtained by the system. The goal is to maximize the accumulated rewards. Recent works have focused on developing offline planning algorithms to solve problems formalized as DEC-POMDPs and DEC-MDPs. They consist of computing a set of individual policies, one per agent, defining the agents' behaviors. Each individual policy maps the agent's information (its state, its observations or its belief state) to an action.

DEFINITION 9.4 (individual policy for a DEC-POMDP (or a DEC-MDP)). An individual policy π_i for an agent \mathcal{A}_i in a DEC-POMDP (or a DEC-MDP) is a mapping from the agents' information to an action $a_i \in \mathcal{A}_i$.

Usually, an individual policy for a DEC-POMDP (or a DEC-MDP) is defined under the “perfect recall assumption” which assumes that each agent always has access to all its received information [SEU 08]. An individual policy π_i for an agent \mathcal{A}_i in a DEC-POMDP (or a DEC-MDP) is then defined as a mapping from the agent's local history of observations $\bar{o}_i = o_i^1, \dots, o_i^t$ to an action $a_i \in \mathcal{A}_i$. Since there provably exists an optimal deterministic policy to every DEC-POMDP, the agents' local history of observations is a sufficient information to make an optimal decision. When the optimal policy is probabilistic, each agent's has to consider the history of its observations and the history of its actions.

However, an individual policy π_i for an agent Ag_i in a DEC-POMDP (or a DEC-MDP) can also be defined as a mapping from the agents' belief states to an action $a_i \in \mathcal{A}_i$, like it has been done in the POMDP setting (See Chapter 7, section 7.1.4).

DEFINITION 9.5 (joint policy for a DEC-POMDP (or a DEC-MDP)). *A joint policy π in an n -agent DEC-POMDP (or a DEC-MDP) is a set of individual policies $\langle \pi_1, \dots, \pi_n \rangle$, where π_i is the individual policy of the agent Ag_i .*

Optimally solving a DEC-POMDP consists of finding a joint policy which maximizes the expected total reward of the system. When considering a finite-horizon DEC-POMDP with initial state s_0 , this consists of maximizing the value $V^\pi(s_0)$ defined over the finite horizon T such as

$$V^\pi(s_0) = E \left[\sum_{t=0}^{T-1} R(s_t, a_t, s_{t+1}) \mid s_0, \pi \right].$$

When considering infinite-horizon DEC-POMDPs, a discount factor $\gamma \in [0, 1)$ is used to weight the expected reward:

$$V^\pi(s_0) = E \left[\sum_{t=0}^{T-1} \gamma^t R(s_t, a_t, s_{t+1}) \mid s_0, \pi \right].$$

9.4.2. Multiagent team decision problems

Multiagent Team Decision Problems (MTDP), introduced by Pynadath and Tambe [PYN 02], are very similar to DEC-POMDPs. Like DEC-POMDPs, MTDPs allow for formalizing problems of decentralized control in stochastic and cooperative domains. An MTDP has almost the same components as a DEC-POMDP. In order to easily formalize the beliefs each agent has about the system state at step t , a set of belief states is added to the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \Omega, \mathcal{O}, \mathcal{R} \rangle$. Thus, MTDPs distinguish between observations and beliefs about the state of the system.

DEFINITION 9.6 (multiagent team decision problem). *An MTDP is defined as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, B, \Omega, \mathcal{O}, \mathcal{R} \rangle$ where:*

- $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \Omega, \mathcal{O}, \mathcal{R} \rangle$ is a DEC-POMDP;
- $B = \langle B_1, \dots, B_n \rangle$ is the set of belief states of the agents where B_i is the set of belief states b_i^t of Ag_i ; each agent Ag_i builds its own belief state b_i^t at time t from the observations it has made until t .

Solving an MTDP consists of finding an optimal joint policy $\pi = \langle \pi_1, \dots, \pi_n \rangle$, where each individual policy π_i is a mapping from a belief state b_i^t to an action a_i .

Unlike Seuken and Zilberstein's definition of an individual policy for a DEC-POMDP [SEU 08], the MTDP definition of an individual policy does not assume perfect recall.

DEFINITION 9.7 (individual policy for an MTDP). *An individual policy π_i for an agent A_{g_i} in an MTDP is a mapping from the agents' belief states $b_i^t \in B_i$ to an action $a_i \in \mathcal{A}_i$.*

DEFINITION 9.8 (joint policy for an MTDP). *A joint policy π in an n-agent MTDP is a set of individual policies $\langle \pi_1, \dots, \pi_n \rangle$, where π_i is the individual policy of the agent A_{g_i} .*

Seuken and Zilberstein [SEU 08] have proven that DEC-POMDPs and MTDPs are equivalent if each agent has access to all of its received information (each agent recalls all the observations and the messages it has previously received). Indeed, we have shown in Chapter 7, section 7.1.3, that the belief state can be calculated from the entire history of observations and actions. This assertion remains true in the multiagent case, and the state of information captured by a DEC-POMDP and an MTDP is thus equivalent.

Seuken and Zilberstein have also demonstrated that the policies are equivalent if each agent has access to all the information it has accumulated: a policy with a value k for a given MTDP has the same value k for the corresponding DEC-POMDP. This DEC-POMDP is obtained by extracting the history of observations from each belief state. Thus, an optimal policy for an MTDP is also an optimal policy for the corresponding DEC-POMDP, and an optimal policy for a DEC-POMDP is also an optimal policy for the corresponding MTDP.

Table 9.1 describes the relationships between MTDPs, DEC-POMDPs, DEC-MDPs, POMDPs and MDPs.

Control	Centralized		Decentralized	
Joint observability of the system state	Yes	No	Yes	No
Formalism	MDP MMDP	POMDP	DEC-MDP DEC-POMDP	MTDP

Table 9.1. Relationships between Markovian models and joint observability

9.4.3. Complexity

Solving a DEC-POMDP or a DEC-MDP optimally is a very hard problem because the agents do not individually observe the state of the system – and because their respective beliefs about this state are not synchronized. While the performance of the system relies on the underlying state s and the joint action a that is been executed, the synchronized knowledge about this state and the potential actions of all agents is in general not obtainable in a decentralized setting.

Although optimally solving a POMDP is PSPACE-complete for the finite-horizon case (see Chapter 7, section 7.3) [PAP 87], optimally solving a DEC-POMDP is much more difficult. Bernstein *et al.* [BER 02] proved that solving a finite-horizon DEC-POMDP with 2 agents or more is NEXP-complete. The same complexity holds for DEC-MDPs.

These complexity results are closely related to the observability each agent has about the state of the system. If each agent individually fully observes the state of the system, the multiagent decision problem can be formalized as an MDP whose complexity is P-complete. If the agents have no observations about the system (blind agents), the multiagent decision problem can be formalized as a non-observable MDP (NOMDP) which is known to be NP-complete. Table 9.2 sums up how the observability influences the complexity of the problem.

Observability of the system state	Locally observable	Jointly fully observable	Jointly partially observable	Non-observable
Complexity	P-complete	NEXP-complete	NEXP-complete	NP-complete
Formalism	MMDP, DEC-MDP	DEC-MDP	DEC-POMDP	
			MTDP	

Table 9.2. Observability and time complexity

9.5. Sub-classes of DEC-POMDPs

It has been shown that optimally solving a DEC-POMDP is a hard problem. This is why several attempts have been made to identify problem properties that can help reducing the complexity of computing an optimal policy. We present several of such subclasses in the following sections.

9.5.1. Transition and observation independence

In some problem settings, the actions of each agent have only limited effects on the other agents. This observation led Goldman and Zilberstein [GOL 04] to introduce the

notions of transition independence and observation independence. In order to define transition independence and observation independence, we first introduce factored DEC-MDPs (which relate to the Factored MDPs presented in Chapter 4).

DEFINITION 9.9 (factored DEC-MDPs). *An n -agent factored DEC-MDP is a DEC-MDP where the state of the system \mathcal{S} can be factored into $n + 1$ components $\mathcal{S} = \mathcal{S}_0 \times \mathcal{S}_1 \times \dots \times \mathcal{S}_n$. \mathcal{S}_0 is the features of the system state that may be observed by the agents and that are not modified by their actions of the agents themselves. \mathcal{S}_i ($i \in [1, n]$) denotes those features of the system state that are observed and affected only by agent $\mathcal{A}_{\mathcal{G}_i}$.*

In a factored DEC-MDP, for some agent $\mathcal{A}_{\mathcal{G}_i}$, we refer to $s_i \in \mathcal{S}_i$ as its *sub-state* and $\hat{s}_i \in \mathcal{S}_i \times \mathcal{S}_0$ as its *local state*. When the agent fully observes its local state, we obtain $o_i = \hat{s}_i \forall i$.

DEFINITION 9.10 (transition independence). *A factored DEC-MDP with independent transitions is such that the probability an agent $\mathcal{A}_{\mathcal{G}_i}$ moves from a sub-state s_i to a sub-state s'_i only depends on the action a_i the agent has executed. Formally, a factored DEC-MDP is transition independent if the set of states \mathcal{S} can be decomposed into $n + 1$ components $\mathcal{S}_0, \dots, \mathcal{S}_n$ and the transition function can be decomposed as a product of probabilities such as $\mathcal{P} = \prod_{i=0}^n \mathcal{P}_i$, where*

$$\begin{aligned} \mathcal{P}_i(s'_i | (s_0 \dots s_n), (a_1 \dots a_n), (s'_0 \dots s'_{i-1}, s'_{i+1} \dots s_n)) \\ = \begin{cases} \mathcal{P}_0(s'_0 | s_0) & \text{if } i = 0, \\ \mathcal{P}_i(s'_i | \hat{s}_i, a_i, s'_0) & \text{if } 1 \leq i \leq n. \end{cases} \end{aligned}$$

Given a two-agent factored DEC-MDP, transition independence can be formalized as follows:

$$\begin{aligned} \forall s_0, s'_0 \in \mathcal{S}_0, \forall s_1, s'_1 \in \mathcal{S}_1, \forall s_2, s'_2 \in \mathcal{S}_2, \forall a_1 \in A_1, \forall a_2 \in A_2, \\ \Pr(s'_0 | (s_0, s_1, s_2), a_1, a_2, (s'_1, s'_2)) = \Pr(s'_0 | s_0) = \mathcal{P}_0, \\ \Pr(s'_1 | (s_0, s_1, s_2), a_1, a_2, (s'_0, s'_2)) = \Pr(s'_1 | \hat{s}_1, a_1, s'_0) = \mathcal{P}_1, \\ \Pr(s'_2 | (s_0, s_1, s_2), a_2, a_1, (s'_0, s'_1)) = \Pr(s'_2 | \hat{s}_2, a_2, s'_0) = \mathcal{P}_2. \end{aligned}$$

DEFINITION 9.11 (observation independence). *A factored DEC-MDP with independent observation is such that the set of states \mathcal{S} can be decomposed into $n + 1$ components $\mathcal{S}_0, \dots, \mathcal{S}_n$ and there exists, for each agent $\mathcal{A}_{\mathcal{G}_i}$, an observation function \mathcal{O}_i such that*

$$\begin{aligned} \mathcal{O}_i(\hat{s}_i, a_i, \hat{s}'_i, o_i) &= \Pr(o_i | \hat{s}_i, a_i, \hat{s}'_i) \\ &= \Pr(o_i | \langle s_0, \dots, s_n \rangle, \langle a_1, \dots, a_n \rangle, \langle s'_0, \dots, s'_n \rangle, \langle o_1, \dots, o_{i-1}, o_{i+1}, \dots, o_n \rangle) \end{aligned}$$

and

$$\mathcal{O}(\langle s_0, \dots, s_n \rangle, \langle a_1, \dots, a_n \rangle, \langle s'_0, \dots, s'_n \rangle, \langle o_1, \dots, o_n \rangle) = \prod_{i=1}^n \mathcal{O}_i(\hat{s}_i, a_i, \hat{s}'_i, o_i).$$

Properties such as transition or observation independence allow for identifying classes of problems that are easier to solve than general DEC-MDPs. Goldman and Zilberstein [GOL 04] have proven that a DEC-MDP with independent transitions and observations is NP-complete. In fact, a local policy for an agent $\mathcal{A}g_i$ is a mapping from its last observation o_i to an action a_i . Its policy is thus of size polynomial in $|\mathcal{S}_i|T$, where $|\mathcal{S}_i|$ is the number of states of agent $\mathcal{A}g_i$. Computing the value of such a policy takes polynomial time. There are $|A_i|^{|\mathcal{S}_i|T}$ possible policies to evaluate, so computing an optimal policy for a DEC-MDP with independent transitions and observations is NP-complete.

Based on this observation, an optimal algorithm, the Coverage Set Algorithm (CSA), has been developed to solve DEC-MDPs with independent observations and transitions [BEC 03] (see section 9.6.1.3).

9.5.2. Goal oriented DEC-POMDPs

Goal oriented DEC-MDPs [GOL 04] formalize problems where the agents try to reach particularly identified goal states. For instance, in the car-maintenance example, the garage-agents may try to reach a state where there is no more cars to repair.

DEFINITION 9.12. *Goal oriented DEC-POMDPs* A DEC-POMDP is goal-oriented (GO-DEC-POMDP) if the following conditions hold:

- 1) there exists a special subset \mathcal{G} of \mathcal{S} of global goal states. At least one of the global goal states $g \in \mathcal{G}$ is reachable by some joint policy;
- 2) the process ends at time T (the finite horizon of the problem);
- 3) all actions a_i of agent $\mathcal{A}g_i$ incurs a cost $C(a_i) < 0$;
- 4) the global reward is $R(s, \langle a_1, \dots, a_n \rangle, s') = \sum_{i=1}^n C(a_i)$;
- 5) if at time T the system is in a state $s \in \mathcal{G}$, there is an additional reward that is awarded to the system for reaching a global goal state.

It has been proved that optimally solving a GO-DEC-MDP is NEXP-complete and thus not easier than solving DEC-MDPs in general. However, particular classes of GO-DEC-MDPs indeed have a lower complexity than NEXP.

DEFINITION 9.13. A GO-DEC-POMDP has uniform cost when the cost of all actions is the same.

Goldman and Zilberstein [GOL 04] have proven that optimally solving a GO-DEC-MDP with independent transitions and observations, with a single goal state and uniform cost, is P-complete. In this case, each agent separately calculates a policy that minimizes the cost to the global goal state. Since costs are uniform, computing an optimal policy therefore consists of finding the shortest path to the goal state (this can be done using dynamic programming).

In this section, we have described properties of the multiagent decision problems that allow for identifying sub-classes of DEC-POMDPs and DEC-MDPs with a lower complexity. These properties can then be exploited to develop more efficient algorithms. Such algorithms are presented in section 9.6.2.7. Figure 9.3 sums up the influence of transition independence, observation independence and goal-oriented properties on the complexity.

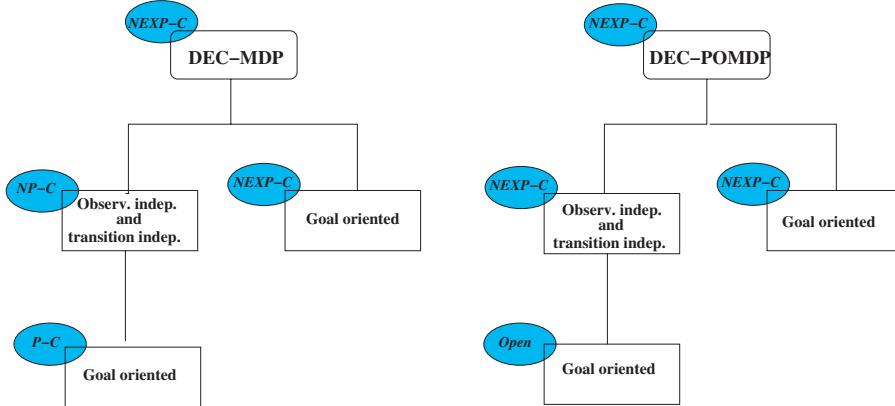


Figure 9.3. Complexity analysis of classes of DEC-MDPs and DEC-POMDPs

9.5.3. DEC-MDPs with constraints

DEC-POMDPs and DEC-MDPs represent powerful models to formalize general multiagent decision problems in stochastic environments, but they are sometimes too general and it may be difficult to formalize some particular structure within this general framework. One such example are constraints between actions. Time or resource dependencies between the actions often arise in practice, but they cannot directly be integrated into the standard DEC-POMDP formalism. Identifying such dependencies, however, can reduce the problem complexity and increase the applicability of DEC-POMDPs to real-world problems. In this section we introduce some classes of DEC-MDPs that allow for explicitly formalizing time and resource constraints between the actions of the agents.

9.5.3.1. Event-driven DEC-MDPs

Event-Driven Decentralized Markov Decision Processes (ED-DEC-MDP) have been defined by Becker *et al.* [BEC 04a]. They introduce time dependencies between the agents such as “enable” or “facilitate” dependencies. It is thus possible to express the fact that an agent \mathcal{A}_i cannot start the execution of an action a_i before another agent \mathcal{A}_j has finished the execution of an action a_j . Quality dependencies such as “ a_i must be executed successfully before a_j starts to produce a higher quality for a_j ” can also be represented.

These dependencies are first expressed using TAEMS [DEC 93] which is a hierarchical task modeling language. The transition function of the ED-DEC-MDP is then defined in order to formalize these constraints.

ED-DEC-MDPs assume that the system state can be factored, and that each agent fully observes its local state. Moreover, ED-DEC-MDPs are reward independent. Thus, the reward function \mathcal{R} can be decomposed as a sum of local reward function such that

$$\mathcal{R}(\langle s_1, \dots, s_n \rangle, \langle a_1, \dots, a_n \rangle, \langle s'_1, \dots, s'_n \rangle) = \sum_{i=1}^n R_i(s_i, a_i, s'_i).$$

Optimally solving an ED-DEC-MDP is NP-complete [BEC 04a]. In fact, an ED-DEC-MDP has a complexity exponential in the number of states $|\mathcal{S}|$ and doubly exponential in the number of dependencies.

9.5.3.2. Opportunity-cost DEC-MDPs

In order to extend the applicability of DEC-POMDP and DEC-MDP models to real-world applications, Beynier and Mouaddib [BEY 05, BEY 06] have introduced the OC-DEC-MDP model. OC-DEC-MDPs are able to represent multiagent decentralized decision problems with time and precedence constraints. They can deal with more complex time and action representations than standard DEC-POMDPs and DEC-MDPs since they allow for different possible durations for each task, and they take into account constraints on task execution. Unlike DEC-POMDPs and DEC-MDPs, Beynier and Mouaddib’s approach do not make the assumption of synchronization between the agents: at each time step, only a subset of the agents have to make a decision while the rest of the agents keep in executing their current tasks.

The following section defines the concept of “mission” and “task” that are used in the OC-DEC-MDP model.

9.5.3.3. Problem statement

Based on the study of real-world multiagent decision problems such as decision making in colonies of robots, Beynier *et al.* have defined a mission \mathcal{X} as a set of agents that must complete a set of tasks.

DEFINITION 9.14 (mission). A **mission** \mathcal{X} is defined as a pair $\langle \mathcal{A}g, \mathcal{T} \rangle$ where:

- $\mathcal{A}g = \{\mathcal{A}g_1, \dots, \mathcal{A}g_n\}$ is a set of n agents;
- $\mathcal{T} = \{t_1, \dots, t_p\}$ is a set of tasks the agents have to execute.

Time and resources are considered as discrete. Thus, each task is assigned a probabilistic set of durations, a probabilistic set of resource consumptions, a set of predecessors and a time window. Each task of the problem must be executed respecting time, precedence and resource constraints.

DEFINITION 9.15. Each **task** $t_i \in \mathcal{T}$ is defined by:

- **an agent** that must execute the task;
- **different possible durations** associated with **probabilities**; $P_i^t(\delta_i)$ is the probability for the execution of task t_i to last δ_i time units;
- **different possible resource consumptions** associated with **probabilities**. $P_i^r(\Delta_r^i)$ is the probability for the execution of task t_i to consume Δ_r^i resources; each agent has an initial amount of resources available for executing its tasks, and this amount of resources must be sufficient to successfully execute the task;
- a **time window** $TC_i = [EST_i, LET_i]$ during which the task t_i must be executed; EST_i stands for the Earliest Start Time of the task and LET_i is its Latest End Time; the time window describes time constraints on the execution of the task;
- a **set of predecessor tasks** $Pred_i$: the tasks that must be finished before t_i can start

$$\forall t_i \in \mathcal{T}, \quad t_i \notin root \iff \exists t_j \in \mathcal{T} : t_j \in Pred(t_i)$$

where **root** are the first tasks to be executed (tasks without predecessors);

- a **reward** \mathcal{R}_i obtained by the agent when the task t_i has been executed respecting time, precedence and resource constraints.

Each time an agent finishes executing a task with respect to constraints, it obtains a reward which depends on the executed task. The essence of the problem is to find a joint policy that maximizes the cumulative reward of the agents. Since the execution time and the resource consumption of the tasks are uncertain, the optimal policy is the one that maximizes the expected reward of the system.

9.5.3.4. The OC-DEC-MDP model

In order to plan the execution of a mission \mathcal{X} and to solve large problems, Beynier and Mouaddib suggest splitting the initial multiagent decision problem into a set of MDPs that are easier to solve. Each MDP stands for a single-agent decision problem.

Thus, individual states and actions are considered and the exponential growth of the joint action set and the state set is avoided. However, precedence constraints between the tasks lead to some dependencies between the agents, and the local MDPs are not independent. Beynier and Mouaddib thus introduce the OC-DEC-MDP framework, which performs such a decomposition and proposes a richer model of time and action formalizing time, precedence and resource constraints.

DEFINITION 9.16. *An n-agent OC-DEC-MDP is a set of n MDPs, one for each agent. The MDP associated with an agent \mathcal{A}_i is defined as a tuple $\langle \mathcal{S}_i, \mathcal{A}_i, \mathcal{P}_i, \mathcal{R}_i \rangle$ where:*

- \mathcal{S}_i is the finite set of states of the agent \mathcal{A}_i ,
- \mathcal{A}_i is the finite set of actions of the agent \mathcal{A}_i ,
- \mathcal{P}_i is the transition function of the agent \mathcal{A}_i ,
- \mathcal{R}_i is the reward function of the agent \mathcal{A}_i .

In order to define the components of the local MDPs, constraints propagation algorithms that use the formal description of the mission are employed.

Each component of the local MDPs is defined to formalize the constraints on the task execution. There exists three kinds of states:

- *success states*: the states resulting from the successful execution of a task (respecting time, precedence and resource constraints);
- *partial failure states*: the states resulting from not respecting precedence constraints (it starts the execution of a task too early, i.e. before the predecessors finish their execution); when an agent starts to execute a task, it immediately observes whether the predecessors have finished their executions or not (this information is assumed to be an observable feature of the environment);²
- *permanent failure states*: the states resulting from violating time or resource constraints are violated (the agent finishes to execute the task after the deadline or it lacks resources).

Each agent \mathcal{A}_i observes its state s_i but does not have access to the other agents' states, nor to the system state.

Since problems formalized by OC-DEC-MDPs are transition dependent, each individual transition function depends on the other agents' transition functions. Beynier and Mouaddib use Bayesian Networks and time propagation algorithms to decompose the joint transition function into individual transition functions.

2. Since the agent could retry to execute the task later and succeed, this failure is not permanent and is called “partial failure”.

It has been proved that optimally solving an OC-DEC-MDP requires an exponential amount of computation time. Note that time, resource and precedence constraints limit the numbers of states and actions. However, these constraints do not reduce the worst-case complexity of OC-DEC-MDPs.

9.5.4. Communication and DEC-POMDPs

Previous sections have discussed the influence of the (partial) observability of the environment on the complexity of multiagent decision problems. Studies of multiagent systems show that communication is an obvious way to increase each agent's knowledge about the system: if an agent communicates its observations, it increases the other agents' knowledge about the system state.³ The agents can therefore better coordinate and may achieve a higher performance as a team.

In section 9.4.1, we show that the individual observability of the system state can be obtained if the system state is jointly fully observable and communication between the agents is free and instantaneous. Unfortunately, things are usually not that easy: communicating is costly (it takes time and it consumes resources such as energy or bandwidth) and the system state is not jointly fully observable. Even if an agent is able to communicate, it must therefore trade-off the cost of communicating against the utility of communicated information [BEC 05].

Communication can be expressed directly within the ordinary DEC-POMDP framework; it is sufficient to define actions that represent the transmission of a message, observations that represent the reception of a message, and to adapt the transition and observation functions accordingly. However, it may sometimes be helpful to introduce explicit communication into the DEC-POMDP framework. Common approaches that address this issue can be divided into two categories. The first category [XUA 01, GOL 04, PYN 02] assumes that the agents alternate the execution of a domain action⁴ and the execution of a communication action (see Figure 9.4). During each communication period, the agents choose whether to communicate or not. If an agent decides not to communicate, it waits until the next domain action phase where it could execute a domain action. These approaches do not consider that the agents could execute domain actions instead of communicating.

Nair *et al.* [NAI 04] describe another kind of approach that does not assume a separate communication phase. Since spending time and resources on communication

3. Research on communication distinguishes between direct and indirect communication. Direct communication consists of information sharing using direct message exchange. Indirect communication consists of acting upon the environment to modify features that could be observed and interpreted by the other agents as communicated information.

4. A domain action is an action that does not consist of communicating.

influences the future opportunities to execute domain actions, there exists an associated decision problem at each time step in order to determine whether sharing information or executing a domain action is more promising.

9.5.4.1. DEC-POMDPs with communication

Goldman and Zilberstein [GOL 03] have extended DEC-POMDPs to include particular communication actions. They assume separate communication and action phases. At each decision step, each agent first decides whether it communicates or not. If an agent chooses to communicate, it sends its message to the other agents. The agent then decides which domain action to perform and it executes this action. As shown in Figure 9.4, each decision step consists of two stages: the communication stage and the domain action stage.

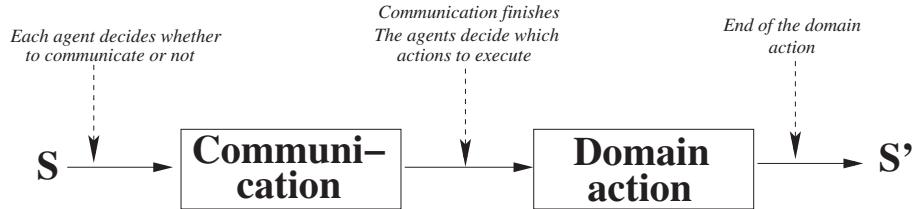


Figure 9.4. Communication and action stages

Each agent's policy π_i is then composed of two distinct policies:

- the communication policy π_i^Σ which maps histories of observations and histories of messages to a communication action,
- the action policy π_i^a which maps histories of observations and histories of messages to a domain action.

The DEC-POMDP-COM model proposed by Goldman and Zilberstein [GOL 03] adds two components to DEC-POMDPs: an alphabet of messages Σ and a cost function C_Σ associated with the transmission of a message.

DEFINITION 9.17 (decentralized partially observable Markov decision processes with communication). A DEC-POMDP-COM is defined as a tuple $\langle \mathcal{S}, \mathcal{A}, \Sigma, C_\Sigma, \mathcal{T}, \Omega, \mathcal{O}, \mathcal{R} \rangle$ where:

- $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \Omega, \mathcal{O}, \mathcal{R} \rangle$ is a DEC-POMDP;
- Σ denotes the alphabet of messages; $\sigma_i \in \Sigma$ is a message that can be sent by agent Ag_i ;
- C_Σ describes the cost of a message; this is a function: $C_\Sigma : \Sigma \rightarrow \mathcal{R}$.

Note that an empty message with zero-cost can be considered.

Communication can also be introduced in DEC-MDPs [XUA 01] in a similar way. MTDPs have also been extended to model communication actions [PYN 02]. A COM-MTDP (*Communicative Multiagent Team Decision Problem*) is thus defined as a tuple $\langle \mathcal{S}, \mathcal{A}, \Sigma, \mathcal{T}, \Omega, \mathcal{O}, \mathcal{B}, \mathcal{R} \rangle$, where the reward function \mathcal{R} is extended to take into account the cost of sending a message.

9.5.4.2. Complexity results

It has been proved [SEU 08] that DEC-POMDPs and DEC-POMDP-COMs are equivalent. In fact, a DEC-POMDP-COM can be mapped to an equivalent DEC-POMDP: communication actions are added to the set of actions \mathcal{A}_i of each agent Ag_i . The state space, the transition function, the observation set, the observation function and the reward function are also adapted to consider explicit communication actions as domain actions. Seuken and Zilberstein have proven that a policy with a value k in the original DEC-POMDP-COM has the same value k in the corresponding DEC-POMDP. Furthermore, a DEC-POMDP can be mapped to an equivalent DEC-POMDP-COM with an empty set of communication messages and no cost function. It can be deduced that DEC-POMDPs and DEC-POMDP-COMs are equivalent.

It has also been demonstrated that COM-MTDPs and DEC-POMDP-COMs are equivalent if each agent has access to all of its received information (perfect recall assumption). When this property holds, all the aforementioned models (DEC-POMDP, MTDP, DEC-POMDP-COM and COM-MTDP) are equivalent [SEU 08]. Since optimally solving a DEC-POMDP with 2 agents or more is NEXP-complete, it follows that MTDPs, DEC-POMDP-COMs and COM-MTDPs are also NEXP-complete.

This complexity result assumes a general model of explicit communication where sending a message has a cost. Under the assumption of free and instantaneous communication, the problem complexity is reduced. If the state of the system is jointly fully observable, the problem can be reduced to an MMDP [BOU 99a] which is known to be P-complete [PAP 87]. If the system state is not jointly fully observable, the problem can be formalized as a large POMDP which is PSPACE-complete [PAP 87]. If the agents have no observability of the system (blind agents), the problem is reduced to an NOMDP which is NP-complete. Table 9.3 sums up how the observability and the communication model influence the problem complexity.

9.5.4.3. Discussion

The communication decision problem consists of deciding when to communicate, which information to communicate, and to which agents. Although some approaches, such as the DEC-POMDP-COM model, define an alphabet of messages which allows for considering different message content, most experiments dealing with these approaches assume that the agents always communicate their last observation.

Observability of the system state	Locally fully observable	Jointly fully observable	Jointly partially observable	Non-observable
Free communication	P-complete	P-complete	PSPACE-complete	NP-complete
General communication	P-complete	NEXP-complete	NEXP-complete	NP-complete

Table 9.3. Observability and complexity results

Moreover, communicative DEC-POMDPs have mainly been studied from the point of view of synchronizing communications [GOL 04, NAI 04, SHE 06]. This kind of explicit communication assumes that if an agent decides to communicate, then all agents must exchange their information so as to unify their world view. Synchronizing communications fit nicely with problems where initializing communication is very costly compared to the cost of the information transfer. Nonetheless, most of the time, such communication is inappropriate and may lead to undesirable extra cost. Moreover, it assumes that an agent cannot execute a domain action while other agents communicate.

Recently, Spaan *et al.* [SPA 08] have described an approach to deal with more realistic forms of communication. Instead of assuming that communication never fails and is instantaneous, they consider probabilities on the success of communications within Δt units of time. Even if this approach allows for representing different durations of communication actions, it considers synchronizing communications.

9.6. Algorithms for solving DEC-POMDPs

The following sections deal with solving problems formalized as DEC-POMDPs. We will essentially cover three classes of algorithms: optimal algorithms for finite-horizon DEC-POMDPs, approximate algorithms for finite-horizon DEC-POMDPs and approximate algorithms for infinite-horizon DEC-POMDPs. Approaches for solving general DEC-POMDPs will be described as well as approaches for solving specific sub-classes of DEC-POMDPs.

In this chapter, we are interested in decentralized control so we consider approaches where the policies are executed in a decentralized way. However, most of the following algorithms calculate an optimal or an approximate policy in a centralized way: a central entity calculates the individual policies of the agents which are then sent to the agents for execution.

9.6.1. Optimal algorithms

A policy for a finite-horizon POMDP can always be represented as a decision tree [KAE 98] denoted by q (see Chapter 7) where nodes are labeled with actions and arcs are labeled with observations. The action associated with the root of the tree is denoted by $\alpha(q)$ and the sub-tree related to the observation o is $q(o)$. $\lambda(q, i)$ is the i th leaf node of tree q .

Let q_i^T be a tree of horizon T . Solving a DEC-POMDP consists of finding a joint policy $\mathbf{q}^T = \langle q_1^T, q_2^T, \dots, q_n^T \rangle$ of horizon T that maximizes the expected amount of reward over the horizon:

$$E\left(\sum_{t=0}^{T-1} \mathcal{R}(s_t, \langle a_1, a_2, \dots, a_n \rangle_t, s_{t+1})\right).$$

The value of the joint policy \mathbf{q}^T for the initial state s can be calculated by dynamic programming using the following equation:

$$V(s, \mathbf{q}^T) = \sum_{s' \in \mathcal{S}} \underbrace{P(s' | s, \mathbf{q}^T)}_{\mathcal{P}(s, \mathbf{q}^T(s), s')} \left[\sum_{o \in \Omega} \underbrace{P(o | s, \mathbf{q}^T, s')}_{\mathcal{O}(s, \mathbf{q}^T(s), s', o)} V_{\mathbf{q}^T}(s') \right],$$

where $\mathbf{o} = \langle o_1, \dots, o_n \rangle$ denotes the joint observation and $\mathbf{q}^T(\mathbf{o})$ is the joint policy for horizon $(T - 1)$ after observing \mathbf{o} .

One way to optimally solve a DEC-POMDP is to perform an exhaustive policy search (see Algorithm 9.1). The number of policies that have to be considered is here

$$\left(|\mathcal{A}|^{\frac{1-|\Omega|^T}{1-|\Omega|}} \right)^n$$

Two planning approaches have been recently proposed to make the exhaustive policy search more efficient. The next subsections describe these approaches.

9.6.1.1. Dynamic programming for DEC-POMDPs

Hansen *et al.* [HAN 04] introduced the first algorithm to generalize the dynamic programming approach for optimally solving general finite-horizon DEC-POMDPs. Their algorithm is based on the incremental exhaustive enumeration of possible policies and on the elimination of dominated strategies. This work extends the

Algorithm 9.1: Exhaustive generation

Input: A set of policies Q_i^t for horizon t
 $k = |\Omega_i|$
for all action $a_i \in \mathcal{A}_i$ **do**

for $p = 1$ **to** $|Q_i^t|^k$ **do**

Let $\langle q_i^{o_1}, \dots, q_i^{o_k} \rangle_p$ be the p th selection of k policies in Q_i^t , build an
new policy q_i^{t+1} with:

for $j = 1$ **to** k **do**

$q_i^{t+1}(o_j) := q_i^{o_j}$

$\alpha(q_i^{t+1}) := a_i$

Output: A set of policies Q_i^{t+1} for horizon $(t + 1)$

concept of belief states (see Chapter 7) to the decentralized control in multiagent settings. A belief state describes the probabilistic knowledge available to an agent \mathcal{A}_{g_i} at execution step t . It can be seen as the summary of the estimate about the state of the system s_t and the future strategies of the other agents. Let Q_i^t be the set of possible policies of agent \mathcal{A}_{g_i} at time t , and let $\mathbf{Q}_{-i}^t = Q_1^t \times \dots \times Q_{i-1}^t \times Q_{i+1}^t \times \dots \times Q_n^t$ be the sets of possible future policies for the agents \mathcal{A}_{g_j} , $j \neq i$, then a multiagent belief state for agent \mathcal{A}_{g_i} is a distribution over \mathcal{S} and \mathbf{Q}_{-i}^t :

DEFINITION 9.18 (multiagent belief state). *A multiagent belief state b_i for an agent \mathcal{A}_{g_i} is a probability distribution over underlying states and future policies for all agents: $b_i \in \Delta(\mathcal{S} \times \mathbf{Q}_{-i})$.*

Contrary to single-agent belief states, which are of constant dimensionality, namely the dimensionality of the state space, multiagent belief states are of variable dimensionality: the belief over some other agent's policies effectively depends on the policy space of that agent. It should also be noted that the multiagent belief state reduces to a single-agent belief state in the case of a POMDP (see Chapter 7).

The multiagent belief state completely captures an agent's local partial view of the multiagent environment. It therefore enables the definition of a local value function, defined over the agent's local belief space. If we denote by V_i the value function for agent \mathcal{A}_{g_i} , \mathbf{q}_{-i} the joint policy for all other agents but agent \mathcal{A}_{g_i} , $\langle \mathbf{q}_{-i}, q_i \rangle$ a complete joint policy, then the value of policy q_i in belief state \mathbf{b}_i can be written as follows:

$$V_i(\mathbf{b}_i, q_i) = \sum_{s \in \mathcal{S}} \sum_{\mathbf{q}_{-i} \in \mathbf{Q}_{-i}} \mathbf{b}_i(s, \mathbf{q}_{-i}) V(s, \langle \mathbf{q}_{-i}, q_i \rangle).$$

It is important to note that a local policy cannot be evaluated by itself and without taking into account the dynamics of the other agents. This is true for competitive as well as for cooperative settings. In this sense, the decentralized control problem can be interpreted as a special case of a cooperative game. In game theory, the notion of best response (see section 8.2.2.4) captures the fact that a policy is the best choice, given that all other players have already made their choice.

DEFINITION 9.19 (best response policy). *We say that $B_i(\mathbf{b}_i)$ is the best response policy of the agent $\mathcal{A}g_i$ among the set of candidate policies Q_i and for belief state \mathbf{b}_i if*

$$B_i(\mathbf{b}_i) = \arg \max_{q_i \in Q_i} V_i(\mathbf{b}_i, q_i).$$

The best response policy is the policy that optimally completes the behavior of the group, given that all agents $\mathcal{A}g_j$, $j \neq i$, already know which policy they should execute. Determining agent $\mathcal{A}g_j$'s optimal policy however requires that all agents – and thus agent $\mathcal{A}g_j$ – have made a decision about their optimal policy. This obviously leads to circular dependencies, and determining the optimal joint policy through n individual optimizations is usually not possible.

We will see that the dynamic programming approach proceeds the other way around: instead of determining the best response policies, it removes all those policies that can never constitute a best response in any case. We thus extend the concept of *useful vector* (introduced in Chapter 7) to multiagent settings and we introduce the notion of *useful policy*.

A policy is said to be useful if it constitutes a best response for at least one belief state. Otherwise, the policy is said to be dominated.

DEFINITION 9.20 (dominated policy). *A policy $q_i \in Q_i$ is said to be dominated if it is a suboptimal policy over the entire belief space, which means that for each belief state \mathbf{b}_i , there is at least one other policy \tilde{q}_i that is at least as efficient:*

$$(\forall \mathbf{b}_i) (\exists \tilde{q}_i \in Q_i \setminus \{q_i\}) \quad \text{with} \quad V_i(\mathbf{b}_i, \tilde{q}_i) \geq V_i(\mathbf{b}_i, q_i).$$

A policy that is not completely dominated is a useful policy.

Instead of computing the best response policies for each agent, Hansen *et al.* [HAN 04] proposed an approach that consists of identifying and pruning all dominated policies. The pruning step consists of solving the following linear program:

$$\begin{aligned} & \text{maximize} \quad \epsilon \in \mathbb{R} \\ & \text{subject to} \quad V_i(\mathbf{b}_i, \tilde{q}_i) + \epsilon \leq V_i(\mathbf{b}_i, q_i) \quad (\forall \tilde{q}_i \neq q_i) \\ & \text{with} \quad \sum_{s \in S} \sum_{\mathbf{q}_{-i} \in \mathbf{Q}_{-i}} \mathbf{b}_i(s, \mathbf{q}_{-i}) = 1 \\ & \quad \mathbf{b}_i(s, \mathbf{q}_{-i}) \geq 0 \quad (\forall s \in \mathcal{S}) (\forall \mathbf{q}_{-i} \in \mathbf{Q}_{-i}). \end{aligned}$$

If the result of the linear program is non-positive ($\epsilon \leq 0$), then the policy q_i is dominated and can be removed.

Algorithm 9.2: Dynamic programming for DEC-POMDPs

Input: A DEC-POMDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \Omega, \mathcal{O}, \mathcal{R} \rangle$ and a horizon T

for all agent \mathcal{A}_i : **do**

- | Initialize $Q_i^0 \leftarrow \emptyset$
- for** $t = 1$ **to** $t = T$ **do**

 - | **for** all agent \mathcal{A}_i : **do**

 - | $Q_i^t \leftarrow \text{ExhaustiveGeneration}(Q_i^{t-1})$
 - | **repeat**

 - | Find an agent \mathcal{A}_i and a policy $q_i \in Q_i^t$ with
 - | $\forall \mathbf{b}_i, \exists \tilde{q}_i \in Q_i^t \setminus \{q_i\}$ such that $V_i(\mathbf{b}_i, \tilde{q}_i) \geq V_i(\mathbf{b}_i, q_i)$
 - | $Q_i^t \leftarrow Q_i^t \setminus \{q_i\}$
 - | **until** there is no more possible pruning

Output: A set of useful policies for each agent

In the multiagent dynamic programming algorithm proposed by Hansen *et al.*, a generation step takes a set of policies of horizon t as input, and builds the exhaustive set of policies of horizon $t+1$ such that each policy of the new set contains a policy that appears in the original set. A pruning step then traverses the new policy set, examines whether each one of these policies is indeed useful and eliminates dominated policies.

This process is repeated for each horizon, starting from horizon 1. Algorithm 9.2 sums up the dynamic programming approach for DEC-POMDPs. Choosing an optimal joint policy for the initial state s_0 is then obtained by maximizing:

$$\mathbf{q}_{s_0}^* = \arg \max_{\mathbf{q}^T \in Q_1^T \times \dots \times Q_n^T} V(s_0, \mathbf{q}^T).$$

9.6.1.2. Heuristic search for DEC-POMDPs

Szer *et al.* [SZE 05] proposed another approach for optimally solving finite-horizon DEC-POMDPs. This approach, called MAA* (multiagent A*), extends the heuristic search algorithm A* to multiagent settings.⁵

5. A variant of A*, called LAO*, has already been described in Chapter 6, section 6.2.3.3. It extends A* to solve mono-agent MDPs.

While dynamic programming algorithms build policies from the last decision step to the first decision step and prune dominated strategies at each step, MAA* builds policies from the first decision step to the last one and makes use of a heuristic function to eliminate dominated strategies. As proved by Szer *et al.* [SZE 05], MAA* is both complete and optimal for finite horizon DEC-POMDPs.

The algorithm incrementally develops the search tree of policies: the leaf nodes of the tree describe partial solutions and at each iteration the solution that seems to be the most promising is expanded for the next step. A leaf node of the policy tree stands for a joint policy up to horizon $t < T$. Expanding the joint policy q^t at horizon t consists of building all the children of q^t , i.e. the joint policies q^{t+1} at horizon $t+1$ that consists of applying q^t up to horizon t and then executing one of the possible actions at horizon $t+1$. Figure 9.5 describes a piece of the search tree for the expansion process from horizon $t = 2$ to horizon $t + 1 = 3$.

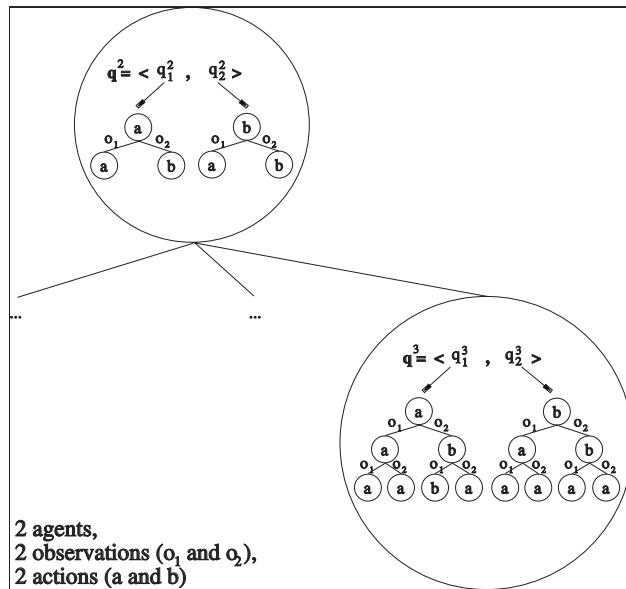


Figure 9.5. A piece of the search tree. One of the children of the joint policy of horizon 2 is expanded to build a joint policy of horizon 3

In order to decide which leaf node is the most promising, the algorithm calculates the exact value of the joint policy q^t that has been developed up to horizon t and a heuristic function H^{T-t} is then used to estimate the expected value of the policy to come. Δ^{T-t} is introduced as a completion of an arbitrary depth- t joint policy, which means a set of depth-($T-t$) policy trees that can be attached at the leaf nodes of a joint policy q^t such that $\langle q^t, \Delta^{T-t} \rangle$ constitutes a complete joint policy of depth T .

The difference between the values $V(s_0, \mathbf{q}^t)$ and $V(s_0, \langle \mathbf{q}^t, \Delta^{T-t} \rangle)$ defines the value of the completion Δ^{T-t} :

$$V(\Delta^{T-t} | s_0, \mathbf{q}^t) := V(s_0, \langle \mathbf{q}^t, \Delta^{T-t} \rangle) - V(s_0, \mathbf{q}^t).$$

Similarly, the value of a joint policy can be calculated as the sum of the value of a root and the value of the completion:

$$V(s_0, \langle \mathbf{q}^t, \Delta^{T-t} \rangle) = V(s_0, \mathbf{q}^t) + V(\Delta^{T-t} | s_0, \mathbf{q}^t).$$

Given a partially built policy \mathbf{q}^t , the upper bound on the value of the joint policy, based on \mathbf{q}^t and defined up to horizon T , can be calculated using the value of the best possible completion:

$$\bar{V}^{*T}(s_0, \mathbf{q}^t) = V(s_0, \mathbf{q}^t) + \max_{\Delta^{T-t}} V(\Delta^{T-t} | s_0, \mathbf{q}^t).$$

An exact computation of this value requires an exhaustive search in the policy space and quickly becomes intractable. This is why MAA* makes use of the heuristic function H^{T-t} which estimates the value of the best completion.

DEFINITION 9.21 (multiagent heuristic function). *The heuristic value function that guides the search of the best completion must overestimate the actual expected reward for any completion of a joint policy \mathbf{q}^t :*

$$H^{T-t}(s_0, \mathbf{q}^t) \geq \max_{\Delta^{T-t}} V(\Delta^{T-t} | s_0, \mathbf{q}^t).$$

Such a heuristic function is said to be an admissible heuristic.

Given an admissible heuristic H , the value function of the algorithm MAA* can then be defined as follows:

$$\bar{V}^T(s_0, \mathbf{q}^t) = V(s_0, \mathbf{q}^t) + H^{T-t}(s_0, \mathbf{q}^t) \geq \bar{V}^{*T}(s_0, \mathbf{q}^t).$$

The core part of the search algorithm is the definition of an admissible heuristic function. As pointed out by [LIT 95] and later by [HAU 00] for the single-agent case, an upper bound for the value function of a POMDP can be obtained through the underlying completely observable MDP. The value function of a POMDP is defined over the belief states (see Chapter 7). The optimal value for a belief state of a POMDP can then be approximated as follows:

$$V_{\text{POMDP}}^*(\mathbf{b}) \leq \sum_{s \in \mathcal{S}} \mathbf{b}(s) V_{\text{MDP}}^*(s).$$

Although it is currently not known how to evaluate a value function over belief states in the multiagent case, a similar upper bound property can still be stated for decentralized POMDPs. Let $P(s | s_0, \mathbf{q})$ be the probability that the system is in state s after executing the joint policy \mathbf{q} from s_0 . A heuristic h that overestimates the optimal value of the DEC-POMDP can then be defined such that

$$h^t(s) \geq V^{*t}(s).$$

This allows for defining the following class of heuristic functions:

$$H^{T-t}(s_0, \mathbf{q}^t) := \sum_{s \in \mathcal{S}} P(s | s_0, \mathbf{q}^t) h^{T-t}(s). \quad (9.1)$$

Intuitively, any such heuristic is optimistic because it effectively simulates the situation where the real underlying state is revealed to the agents after execution of the joint policy \mathbf{q}^t . It has been proved [SZE 05] that any heuristic function H as defined in 9.1 is admissible if h is admissible.

The computation of a good heuristic function is in general much easier than the computation of the exact value. Choosing a heuristic function among admissible heuristics results from a trade-off between the quality of the estimate and the computation cost. Nonetheless, it is often difficult to decide *a priori* which heuristic would be the best one. Szer *et al.* described several admissible heuristics for DEC-POMDPs:

- *The MDP heuristic:* An easy way to calculate a value function heuristic is to use the solution of the underlying centralized MDP with the remaining finite horizon $T - t$. Solving an MDP can be done using dynamic programming or search technique and requires only polynomial time (see Chapter 1). Using the underlying MDP as an admissible heuristic for search in POMDPs has already been applied to the single-agent case as described in [WAS 96] and later in [GEF 98].

- *The POMDP heuristic:* A tighter yet more complex value function heuristic consists of using the solution to the underlying partially observable MDP. Solving POMDPs is PSPACE-complete [PAP 87] and usually involves linear programming. Although this heuristic is more difficult to calculate, it leads to better performance of MAA*.

- *The DEC-POMDP heuristic:* an important special case of a value function heuristic consists of the optimal value itself. One way to calculate this value efficiently is to apply MAA* again:

$$h^{T-t}(s) := V_{\text{DEC-POMDP}}^{T-t}(s) = MAA^{*T-t}(s). \quad (9.2)$$

This leads to a recursive approach, where a call to MAA^{*T} invokes several calls to MAA^{*T-1} and where each of them triggers new sub-searches.

9.6.1.3. Optimal algorithms for sub-classes of DEC-POMDPs

Based on the work of Goldman and Zilberstein about the influence of problem properties on DEC-POMDP's complexity, Becker *et al.* have developed an optimal algorithm for solving specific sub-classes of DEC-POMDPs. CSA (*coverage set algorithm*) allows for solving event-driven DEC-MDPs [BEC 04a] or DEC-MDPs with independent transitions and independent observations [BEC 03, BEC 04b].

Although CSA has been presented in the two-agent case, it can be used to solve larger problems involving more agents. CSA consists of three steps (which are detailed below): 1) create *augmented MDPs*, 2) find an *optimal coverage set* for the augmented MDPs, 3) find the optimal joint policy. If we consider two agents $\mathcal{A}g_i$ and $\mathcal{A}g_j$, an *augmented MDP* formalizes the decision problem in which agent $\mathcal{A}g_i$ must calculate its optimal policy given a fixed policy for agent $\mathcal{A}g_j$.

The first step of CSA thus consists of defining, for each policy of $\mathcal{A}g_j$, an augmented MDP. The second step of the algorithm calculates the optimal policy for each augmented MDP: the optimal policy of $\mathcal{A}g_i$ for each possible policy of $\mathcal{A}g_j$ is calculated. This set of policies defines the optimal *coverage set*. Computing this set aims at minimizing the space of possible policies while performing an exhaustive search: for each possible policy π_j of $\mathcal{A}g_j$, the best response π_i^* of $\mathcal{A}g_i$ is calculated and only the joint policy $\pi = \langle \pi_j, \pi_i^* \rangle$ is added to the coverage set. Joint policies such as $\pi = \langle \pi_j, \pi_i \neq \pi_i^* \rangle$ are ignored. Finally, the third and last step of the algorithm consists of searching for the best joint policy in the coverage set.

This algorithm has a complexity exponential in the number of states of the system. While solving ED-DEC-MDPs, it has been proved that the complexity is doubly exponential in the number of dependencies. The worst-case complexity of the CSA for solving ED-DEC-MDPs is equivalent to the complexity of the exhaustive search. This worst-case complexity explains why the scalability of CSA remains limited. Indeed, even if the complexity of the problems that can be solved by CSA is less important than the complexity of general DEC-POMDPs, solving large sub-classes of DEC-POMDPs remains a hard problem. Experimental results showed that CSA can only solve ED-DEC-MDPs involving few agents few dependencies (results are presented for 2 agents and 4 dependencies). While increasing the number of agents or the number of dependencies, the number of states of the ED-DEC-MDPs quickly becomes untractable.

9.6.2. Approximate algorithms

The approaches that have just been described all calculate an optimal solution to DEC-POMDPs or DEC-MDPs. Because of the high complexity of optimally solving DEC-POMDPs and DEC-MDPs, these approaches quickly become untractable. This is why approximate algorithms have recently been developed. In the following section,

we present some of these approximate solutions for both finite and infinite horizon problems.

9.6.2.1. Heuristics and approximate dynamic programming

As explained in the previous section, all exact algorithms suffer from high complexity, and using a heuristic value function to guide the policy search remains insufficient to obtain more scalable algorithms. Heuristic approximation methods have been proposed to reduce the complexity. Up to now, such heuristics mainly apply to dynamic programming approaches [SZE 06, SEU 08, SEU 07] and heuristic search [OLI 07b].

Szer *et al.* [SZE 06], for example, address two major drawbacks of the exact dynamic programming approach, namely the computationally expensive linear programming part necessary to identify dominated policies, and the difficulty to exclude those regions of the belief space that are never reachable. They introduce a point-based multiagent dynamic programming algorithm that allows us to reduce the space of belief states by determining those belief states that are likely to be visited in a real-world scenario. Moreover, Szer *et al.* propose to limit the set of possible joint policies by sampling from the set of possible prior policies. Only those policies that are most likely to be spread out far away from each other are retained. The Manhattan distance can be used as a simple metric between policy trees.

Szer *et al.* have thus been able to solve the multi-access channel problem up to horizon 8 whereas previous approaches were only able to solve this problem up to horizon 4. Other efficient heuristics [SEU 08, SEU 07] even enable to solve the same problems up to much larger horizons (horizon 100,000). Nonetheless, further experiments are necessary to estimate the effect of these heuristics on the solution quality.

In the context of heuristic search approaches, different kinds of heuristics can be used in order to speed up the search. They make use of approximations such as:

- performing incomplete searches (by limiting the branching factor for instance);
- estimating the value of a policy using simulation methods (Monte Carlo approach) instead of computing their exact value;
- computing the heuristic value function for the most likely belief states.

Recently, Dibangoye *et al.* [DIB 09a] have presented a version of the dynamic programming value iteration algorithm that provides an error bound on the quality of the returned solution. The Point-Based Value Iteration (PBVI) algorithm proposed by Dibangoye *et al.* is the first scalable algorithm for infinite-horizon DEC-POMDPs with provable error bound. Joint policies are represented as joint deterministic finite-state controllers (DJFSC). As it is the case for the mono-agent value iteration algorithm,

at each iteration step the PBVI algorithm first updates the value function and then updates the current DJFSC.

A modified version of the policy iteration algorithm has also been proposed: the Point-Based Policy Iteration algorithm (PBPI) [DIB 09b]. In order to speed up the algorithm and to improve its scalability, Dibangoye *et al.* make use of some adapted heuristic search methods: the point-based heuristic and the point-based heuristic with branch and bound.

Experimental results on PBPI and PBVI algorithms show some orders of magnitude of improvements in their performance compared to other existing algorithms dealing with dynamic programming and DEC-POMDPs. It has also been shown that the PBPI algorithm scales up to larger sizes of problems and is able to successfully solve many DEC-POMDP benchmarks.

9.6.2.2. *Bounded memory*

The major limitation of the general dynamic programming algorithm for DEC-POMDPs is the explosion in memory requirements: for each additional horizon, a super-exponential number of new value functions has to be considered [HAN 04]. Instead of fixing the problem horizon and searching for an optimal policy, Bernstein *et al.* [BER 05] therefore propose an alternative approach based on first bounding the size of the policy and then trying to optimize its parameters. In this case, policies are based on finite state controllers, which can be executed for an infinite amount of time. This algorithm thus constitutes one way of solving infinite-horizon DEC-POMDPs. Each agent's policy is represented as a stochastic finite state controller and a correlation device is used to allow the agents to coordinate their policies.

The policy iteration algorithm described in [BER 05] updates, at each iteration step, the parameters of a controller so as to improve the value function of the system. Convergence to a global optimum is not guaranteed but the backup applied to a local controller provably produces a joint policy with value at least as high as before the backup. The algorithm thus reaches a local optimum.

Experiments showed that increasing the number of nodes of each controller leads to higher quality but also increases the memory requirements for computing the solution. Since each iteration takes polynomial time and the algorithm uses a bounded amount of memory, this approach solves larger problems than exact dynamic programming algorithms. Nonetheless, this approach remains limited to problems involving a small number of agents since the time complexity of each iteration is exponential in the number of agents.

9.6.2.3. *Co-evolutive algorithms*

Some approximate algorithms for solving decentralized MDPs are based on a co-evolutive iterative approach. These algorithms consist of iteratively improving the

policy of a single agent by fixing the policies of the other agents. For an n -agent DEC-POMDP, a typical co-evolutive algorithm selects at each step an agent $\mathcal{A}g_k$ while the policies of the agents $\mathcal{A}g_i$, $i \neq k$, remain fixed. The co-evolutive algorithm then calculates an optimal policy for $\mathcal{A}g_k$ in response to these fixed policies. The process is repeated until none of the policies can be improved.

Chadès *et al.* [CHA 02, SCH 02] introduced the first co-evolutive algorithm for solving DEC-POMDPs. They first described two co-evolutive algorithms that calculate a Nash equilibrium for problems formalized as MMDPs.

Chadès *et al.* extended their approach to partially observable systems under decentralized control (i.e. problems formalized by DEC-POMDPs). In order to take into account partial observability of the system state, they introduced subjective-MDP. A subjective-MDP is an MDP where states are defined by the agent's perceptions. This model is similar to a POMDP where the decision is based on local perceptions and we do not consider probabilities on the underlying state neither histories of observations.

At each step of the co-evolutive algorithm, an agent $\mathcal{A}g_k$ is selected and the subjective-MDP of this agent is defined given the policies of the other agents. The subjective-MDP is then solved and a new policy is obtained for the agent $\mathcal{A}g_k$. Because a subjective-MDP is not Markovian, this policy is not optimal. In decentralized settings with partial observability, the co-evolutive algorithm proposed by Chadès *et al.* is thus not guaranteed to converge to a Nash equilibrium. Convergence to a Nash equilibrium is only obtained under full observability or centralized control (the problem is then formalized as an MMDP).

Nair *et al.* [NAI 03] have also proposed an algorithm based on co-evolutive principles. JESP (*Joint Equilibrium Based Search for Policies*) enables the solving of sequential multiagent decision problems formalized by MTDPS with independent observations.

At each iteration step, the policy of an agent is modified in order to improve the joint policy. The algorithm provably converges to a Nash equilibrium. This algorithm has the same worst-case complexity as the exhaustive search. It may, however, perform much better in practice.

Nair *et al.* proposed an improvement of JESP in terms of performance by using dynamic programming. DP-JESP (*Dynamic Programming Joint Equilibrium-based Search for Policies*) exploits Bellman's optimality principle to speed up policy computation (each policy is incrementally evaluated and built using Bellman's optimality principle). Thus, on the multiagent tiger problem, experiments show that JESP can solve the problem up to horizon 3 whereas DP-JESP solves the problem up to horizon 7. Nonetheless, since the complexity of the algorithm remains exponential,

larger problems are difficult to solve. Nair *et al.* have therefore been interested in exploiting the structure of the problem to propose another variant of their algorithm.

LID-JESP algorithm (*Locally Interacting Distributed Joint Equilibrium-based Search for Policies*) [NAI 05] combines the JESP algorithm with principles from the field of distributed constraint optimization. LID-JESP solves problems formalized by a Network-Distributed POMDP with independent transitions and independent observations. It exploits the locality of interactions between the agents to speed up the running time of JESP. An interaction graph describes the interaction between the agents. The agents are the vertices and edges describe how the actions of an agent influence the reward of the other agents. A neighborhood is thus defined for each agent $\mathcal{A}g_i$: it describes the set of agents $\mathcal{A}g_j$ whose utility is influenced by $\mathcal{A}g_i$'s actions. This neighborhood is then exploited in order to limit the set of agents to consider while revising the policy of $\mathcal{A}g_i$.

9.6.2.4. Gradient descent for policy search

In Chapter 5, it has been shown that optimization methods can be used to solve sequential decision problems using policy search. These methods can also be used in multiagent settings while searching to optimize a set of controllers instead of optimizing a single controller. Methods based on performing a gradient descent in policy space have been used for finding locally optimal solutions to multiagent sequential decision problems [PES 00, DUT 01, TAO 01]. More recently, the cross-entropy method, a method for combinatorial optimization, has been used to speed up policy search and to find approximate solutions of DEC-POMDPs [OLI 07a].

In cooperative multiagent settings, gradient descent methods can be executed in a centralized way by all the agents or in a decentralized way by each agent (each agent receives the same reward). The controller can be, for example, a finite state automaton [PES 00] or a function approximator (whose input is the current observation or a belief state) [DUT 01, TAO 01]. These learning approaches result in a local optimum. Nonetheless, unlike the JESP algorithm, this local optimum may not be a Nash equilibrium.

9.6.2.5. Bayesian games

Emery-Montemerlo *et al.* [EME 04] proposed a decentralized algorithm for DEC-POMDPs that approximates the problem with a series of Bayesian games.⁶ These Bayesian games together approximate the initial multiagent decision problem.

The approach alternates planning and policy execution. Each planning step consists of solving a one-step Bayesian game for the next execution step. The

6. A Bayesian game is a strategic form game with incomplete information.

calculated policy is then executed and a new planning phase starts for the next execution step. Each Bayesian game is solved using a heuristic that allows for estimating the utility function. The performance of the approach is closely related to the kind of problems and the heuristics that are considered.

9.6.2.6. Heuristics for communicating agents

Goldman and Zilberstein [GOL 03] described a myopic greedy approach to solve the “meeting under uncertainty” problem formalized as a DEC-POMDP-COM. This problem involves two agents that have to meet at some location on a grid as soon as possible. The moves of the agents are uncertain and each agent cannot observe the location of the other agent. Nevertheless, the agents are able to communicate information about their position and to revise the meeting point consequently. Communication can therefore allow the agents to meet faster.

Since communicating is costly, a part of the decision problem consists of deciding when to communicate. If the communication cost is prohibitive, the optimal policy consists of never communicating. If communication is free, the optimal policy consists of communicating at each time step. In general, the optimal policy is between these two extremes. Goldman and Zilberstein described a myopic approach to approximate this optimal policy. At each time step, the agents optimize the choice of when to communicate assuming that they can communicate only once. This myopic approach allows the agents to exchange information only when communicating is particularly beneficial.

Other kinds of communication heuristics have been proposed by Xuan *et al.* [XUA 01] for the meeting under uncertainty problem. Although, these heuristics perform well on the meeting under uncertainty problem, they may be inappropriate for other kinds of problems. Thus, there is a need for approaches that are able to solve the general communication decision problem.

9.6.2.7. Approximate solutions for OC-DEC-MDPs

Based on the OC-DEC-MDP model described in section 9.5.3.2, Beynier *et al.* [BEY 05, BEY 06] developed some efficient algorithms for computing an approximation of the optimal solution even for large multiagent decision problems.

9.6.2.7.1. Opportunity cost and coordination

Given the problem decomposition which is part of the OC-DEC-MDP model, Beynier *et al.* proposed to simultaneously solve each local MDP and to deduce each agent’s policy. Since they consider cooperative multiagent systems where the interactions between the agents arise from precedence and time constraints, maximizing a common reward requires the agents to coordinate – the local MDPs cannot be solved independently.

For the purpose of coordinating the agents, Beynier *et al.* introduced the notion of opportunity cost. The Opportunity Cost is borrowed from economics where it refers to hidden indirect costs associated with a decision. In their work, Beynier *et al.* use the opportunity cost to measure the indirect effect of an agent's decision on other agents. In order to obtain cooperative behaviors, the best action an agent $\mathcal{A}g_i$ can execute from a state s_i then results from a trade-off between:

- the expected utility Q of the agent, and
- the opportunity cost induced on the other agents.

Thus, the policy of an agent $\mathcal{A}g_i$ from a state s_i is calculated using the following equation:

$$\pi_i(s_i) = \arg \max_{a_i \in A_i} (Q_i(s_i, a_i) - OC(s_i, a_i)), \quad (9.3)$$

where A_i stands for the set of actions a_i the agent $\mathcal{A}g_i$ can execute from s_i , $Q_i(s_i, a_i)$ is the expected utility of the agent $\mathcal{A}g_i$ while executing a_i from s_i and $OC(s_i, a_i)$ is the opportunity cost induced on the other agents.

In fact, the opportunity cost stands for a difference in expected utility. It measures the loss in expected utility incurred by the other agents when they deviate from their best policy. Several opportunity cost evaluation algorithms have been proposed. Especially, Beynier *et al.* introduced the expected opportunity cost that accurately measures the influence of one agent on the other agents. Approximations of opportunity costs have also been presented.

9.6.2.7.2. Policy improvement using opportunity costs

In order to efficiently solve problems formalized by OC-DEC-MDPs, Beynier *et al.* described a revision algorithm that calculates an approximate solution respecting time, precedence and resource constraints on task execution. This algorithm starts with an initial policy set and tries to improve it using equation 9.3. Two versions of the algorithm have been developed. A centralized one allows a central entity to revise all agents' policies. It passes through the set of tasks and improves the agent's execution policy of each task. A decentralized version of the algorithm has also been proposed. Thus, each agent improves its own policy and individual policies are simultaneously considered. This second version requires offline communication of opportunity cost values.

Beynier *et al.*'s revision algorithm does not guarantee the optimality of the solution unless some properties hold such as: resources are unlimited, time constraints do not restrict execution tasks, the graph is a two-agent tree with one-way precedence constraints, etc. In general, an approximate solution is obtained. Despite other existing approaches whose complexity is exponential, Beynier *et al.* proved that

both versions of their algorithm have a time complexity polynomial in the number of states and actions. Moreover, precedence, time and resources constraints are exploited to limit the state and the action spaces. Thus, large problem sizes can be handled. Experiments demonstrated that missions involving more than ten agents and hundreds of tasks can be solved. This approach shows that exploiting the properties of the problem and computing an approximate solution is a promising approach to develop more scalable approaches. Such an approach has also been adopted by Nair *et al.* [NAI 05] to efficiently solve DEC-POMDPs in distributed networks (ND-POMDPs, Networked-Distributed POMDPs).

9.6.2.7.3. Iterative policy improvement

The revision algorithm described above consists of improving an initial policy. When it stops, each task has been revised once and a new local policy is available for each agent. In order to obtain better solutions, Beynier *et al.* proposed to re-execute the revision algorithm considering that the initial policy is the policy that has just been calculated [BEY 06].

At each iteration step, the agents improve their initial local policy at the same time. The outcome policies of iteration $N - 1$ are the initial policies of iteration N . This process is repeated until no changes are made on the policies during one iteration. Two versions of the iterative algorithm – which are based on the versions (centralized and decentralized) of the revision algorithm – have been proposed.

The complexity of one iteration is the same as the complexity of the revision algorithms (polynomial in the number of states and actions). The time complexity of this iterative algorithm is thus mainly influenced by the number of iterations. It has been shown that convergence guarantees rely on the method that is used to calculate the opportunity cost. The algorithm converges if we use an estimation of the opportunity cost that accurately measures the influence of a decision on the other agents. In this case, experiments show that convergence is reached within a small number of iterations (most of the time less than 4 iteration steps).

Finally, experiments proved that the iterative algorithm can also be used to solve large problems (more than 10 agents and 100 tasks). Moreover, as explained in the next section, it has been shown that this approach is suitable to solve real-world multi-robot decision problems.

9.7. Applicative scenario: multirobot exploration

In order to prove the applicability of their approach to real-world multiagent decision problems, Beynier *et al.* [BEY 05, BEY 06] applied OC-DEC-MDPs to multirobot exploration scenarios. They considered variants of the Mars rover exploration scenarios (Chapter 14 introduces a similar application but it considers different difficulties from the ones considered by Beynier *et al.*).

A scenario, similar to Mars rover missions, was therefore implemented. It involves two robots that have to explore a set of 8 interesting places (Figure 9.6). The first robot (robot $\mathcal{A}g_1$) can take pictures and the second one (robot $\mathcal{A}g_2$) can collect and analyze ground samples. Robot $\mathcal{A}g_1$ must take picture of sites A, B, D, E, F, H and J , and robot $\mathcal{A}g_2$ must analyze sites C, D, F, H and I . Sites are ordered so as to minimize traveling resource consumptions. Furthermore, precedence constraints have to be taken into account. As taking samples of the ground may change the topology of the site, pictures of a site must be taken before the other robot starts to analyze it. Moreover, robot $\mathcal{A}g_1$ must have left a site before robot $\mathcal{A}g_2$ can start to analyze it. Thus, robot $\mathcal{A}g_1$ must have taken a picture of site D before robot $\mathcal{A}g_1$ enters this site. Time constraints have also to be considered: visiting earliest start times and latest end times are associated with each site.

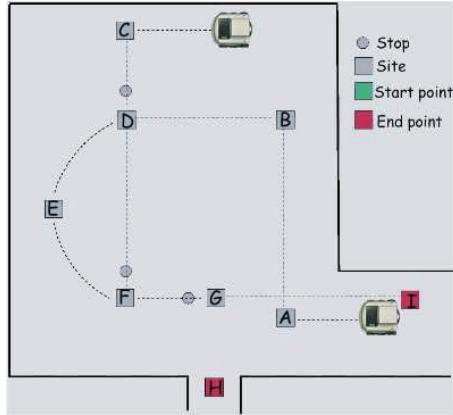


Figure 9.6. Mission scenario

The mission was represented using a mission graph. Then, the corresponding OC-DEC-MDP was automatically built and solved by the iterative algorithm. Finally, resulting policies were implemented on Koala robots. During task execution, robots only have to execute their policies which map each state to an action and thus limit the computational resources needed to make a decision. Coordination is performed without communication between the agents during the execution, and time and precedence constraints are respected. As shown in Figure 9.7 for the crossing point D , while deciding when to start its action, the first robot takes into account the fact that the other robot waits for him (thanks to the OC). The decision of the second robot is based on the probability that robot $\mathcal{A}g_1$ has left the site, the cost of a partial failure, and the robot's own expected value. Thus, robot 1 enters site D , completes its task (Picture 2) and leaves the site (Picture 3). As robot 1 does not know the other robot's actions, it may try to enter the site and fails because the other robot has not



Figure 9.7. Execution of the mission by 2 Koala robots (crossing D)

finished to take the picture. The second robot realizes that it fails when it tries to enter the site. If precedence constraints are not respected, the robot returns to its last position. If time constraints are respected, the robot enters the site (Picture 4). These experiments show that the policies calculated by the OC-DEC-MDP approach can be used by physical robots which are thus able to successfully and cooperatively complete their mission.

Experiments dealing with larger scenarios have been performed on simulators [BEY 05, BEY 06]. They proved that multirobot missions composed of about ten agents and hundreds of tasks can be planned and executed by the OC-DEC-MDP approach (problems up to 20 agents and 800 tasks were solved) with high performance. This approach is thus scalable enough to formalize and solve problems like the ones considered in Mars exploration researches (regarding Mars exploration, the mission to execute is sent to the robots once a day and may involve about ten robots that have to complete hundreds of tasks).

9.8. Conclusion and outlook

In this chapter, we introduced several extensions of MDPs and POMDPs to formalize decision problems in cooperative multiagent systems acting in stochastic environments. We first described the MMDP model to formalize decision problems where each agent observes the system state. We then turned to models that allow for formalizing problems where each agent has a partial observability of the system. We thus introduced DEC-POMDPs and DEC-MDPs, and we reviewed several variants of these models.

In the second part of this chapter, we described algorithms for solving the problems that were formalized in section 9.4. We described optimal algorithms that solve general classes of DEC-POMDPs. Given the high complexity of finding an optimal solution, these algorithms can only solve very small problems. Two kinds of approaches have been proposed to tackle this high complexity. The first set of approaches tries to identify specific properties of the problems (such as transition independence or observation independence) that reduce the complexity. Other approaches develop approximate algorithms that find an approximation of the optimal policy instead of searching for an exact optimal policy. Several approximate algorithms have been described in section 9.6.2.

In this chapter we also showed that some difficulties arise while formalizing real-world multiagent decision problems with DEC-POMDPs. Indeed, DEC-POMDPs suffer from limited representation of time and actions: it is assumed that all the actions have the same duration and they do not take into account constraints on action execution. This issue has been tackled by ED-DEC-MDP and OC-DEC-MDP models described in section 9.5. Nevertheless, these models allow for representing limited sets of constraints. In order to widely apply these models to real-world decision problems, it would thus be necessary to formalize richer sets of constraints.

Although recent works dealing with solving DEC-MDPs have demonstrated the efficiency of approximate algorithms, quality guarantees are lacking. Indeed, existing approximate approaches evaluate the quality of calculated policies in terms of equilibria. For instance, JESP algorithms returns a Nash equilibrium. Nonetheless, for a given problem there may exist several solutions that are Nash equilibria with different qualities. Nash equilibria being local optima, such solutions can lead to high or poor performance depending on the problems and of the equilibrium that is selected. Thus, characterizing the solution in terms of equilibria does not give real guarantees on the quality of the solutions and comparing the performance of different approaches is very difficult. At the moment, few approaches provide an error bound on the solution quality whose evaluation is an important issue. There is thus a need for developing approximate algorithms that provide an upper bound on the distance between the solutions that are calculated and the optimal solution (ϵ -optimal algorithms).

9.9. Bibliography

- [BEC 03] BECKER R., ZILBERSTEIN S., LESSER V. and GOLDMAN C., “Transition-independent decentralized Markov decision processes”, *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'03)*, ACM Press, New York, pp. 41–48, 2003.
- [BEC 04a] BECKER R., LESSER V. and ZILBERSTEIN S., “Decentralized Markov decision processes with event-driven interactions”, *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'04)*, New York, pp. 302–309, 2004.

- [BEC 04b] BECKER R., ZILBERSTEIN S., LESSER V. and GOLDMAN C., “Solving transition independent decentralized Markov decision processes”, *Journal of Artificial Intelligence Research*, vol. 22, pp. 423–455, 2004.
- [BEC 05] BECKER R., LESSER V. and ZILBERSTEIN S., “Analyzing myopic approaches for multi-agent communication”, *Proceedings of the International Conference on Intelligent Agent Technology (IAT)*, Compiègne, France, pp. 550–557, 2005.
- [BER 01] BERNSTEIN D., ZILBERSTEIN S., WASHINGTON R. and BRESINA J., “Planetary rover control as a Markov decision process”, *Proceedings of the 6th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, Montreal, Canada, 2001.
- [BER 02] BERNSTEIN D. S., GIVAN R., IMMERMANN N. and ZILBERSTEIN S., “The complexity of decentralized control of Markov decision processes”, *Mathematics of Operations Research*, vol. 27, no. 4, pp. 819–840, 2002.
- [BER 05] BERNSTEIN D., HANSEN E. A. and ZILBERSTEIN S., “Bounded policy iteration for decentralized POMDPs”, *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI'05)*, Edinburgh, Scotland, 2005.
- [BEY 05] BEYNIER A. and MOUADDIB A. I., “A polynomial algorithm for decentralized Markov decision processes with temporal constraints”, *Proceedings of the 4th International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS'05)*, The Netherlands, pp. 963–969, 2005.
- [BEY 06] BEYNIER A. and MOUADDIB A. I., “An iterative algorithm for solving constrained decentralized Markov decision processes”, *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI'06)*, Boston, MA, pp. 1089–1094, 2006.
- [BOU 96] BOUTILIER C., “Planning, learning and coordination in multiagent decision processes”, *Proceedings of the 6th Conference on Theoretical Aspects of Rationality and Knowledge (TARK'96)*, Morgan Kaufmann, San Francisco, CA, pp. 195–201, 1996.
- [BOU 99a] BOUTILIER C., DEAN T. and HANKS S., “Decision-theoretic planning: structural assumptions and computational leverage”, *Journal of Artificial Intelligence Research*, vol. 11, pp. 1–94, 1999.
- [BOU 99b] BOUTILIER G., “Sequential optimality and coordination in multiagent systems”, *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, Stockholm, Sweden, pp. 478–485, 1999.
- [CHA 02] CHADÈS I., SCHERRER B. and CHARPILLETT F., “A heuristic approach for solving decentralized-POMDP: assessment on the pursuit problem”, *Proceedings of the 2002 ACM symposium on Applied computing (SAC'02)*, Madrid, Spain, pp. 57–62, 2002.
- [DEC 93] DECKER K. and LESSER V., “Quantitative modeling of complex computational task environments”, *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI'93)*, pp. 217–224, 1993.
- [DIB 09a] DIBANGOYE J. S., MOUADDIB A. I. and CHAIB-DRAA B., “Point-based incremental pruning heuristic for solving finite horizon DEC-POMDPs”, *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, Budapest, Hungary, pp. 569–576, 2009.

- [DIB 09b] DIBANGOYE J. S., MOUADDIB A. I. and CHAIB-DRAA B., “Policy iteration algorithms for DEC-POMDPs with discounted rewards”, *AAMAS 2009 Workshop on Multi-agent Sequential Decision-Making in Uncertain Domains*, Budapest, Hungary, 2009.
- [DUT 01] DUTECH A., BUFFET O. and CHARPILLETT F., “Multi-agent systems by incremental gradient reinforcement learning”, *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI'01)*, Seattle, WA, pp. 833–838, 2001.
- [EME 04] EMERY-MONTEMERLO R., GORDON G., SCHNEIDER J. and THRUN S., “Approximate solutions for partially observable stochastic games with common payoffs”, *Proceedings of the 3rd Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'04)*, New York, pp. 136–143, 2004.
- [GEF 98] GEFFNER H. and BONET B., “Solving large POMDPs by real time dynamic programming”, *Working Notes – Fall AAAI Symposium on POMDPs*, Orlando, FL, 1998.
- [GOL 03] GOLDMAN C. and ZILBERSTEIN S., “Optimizing information exchange in cooperative multi-agent systems”, *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'03)*, Melbourne, Australia, pp. 137–144, 2003.
- [GOL 04] GOLDMAN C. and ZILBERSTEIN S., “Decentralized control of cooperative systems: categorization and complexity analysis”, *Journal of Artificial Intelligence Research*, vol. 22, pp. 143–174, 2004.
- [HAN 04] HANSEN E. A., BERNSTEIN D. S. and ZILBERSTEIN S., “Dynamic programming for partially observable stochastic games”, *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI'04)*, San Jose, CA, pp. 709–715, 2004.
- [HAU 00] HAUSKRECHT M., “Value-function approximations for partially observable Markov decision processes”, *Journal of Artificial Intelligence Research*, vol. 13, pp. 33–94, 2000.
- [KAE 98] Kaelbling L. P., Littman M. L. and Cassandra A., “Planning and acting in partially observable stochastic domains”, *Artificial Intelligence*, vol. 101, pp. 99–134, 1998.
- [LIT 95] LITTMAN M. L., CASSANDRA A. R. and KAELBLING L. P., “Learning policies for partially observable environments: scaling up”, *Proceedings of the 12th International Conference on Machine Learning (ICML'95)*, Morgan Kaufmann, San Francisco, CA, pp. 362–370, 1995.
- [NAI 03] NAIR R., TAMBE M., YOKOO M., MARSELLA S. and PYNADATH D. V., “Taming decentralized POMDPs: towards efficient policy computation for multiagent settings”, *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*, Acapulco, Mexico, pp. 705–711, 2003.
- [NAI 04] NAIR R., ROTH M., YOKOO M. and TAMBE M., “Communication for improving policy computation in distributed POMDPs”, *Proceedings of the 3rd International Joint Conference on Agents and Multiagent Systems (AAMAS'04)*, New York, pp. 1098–1105, 2004.

- [NAI 05] NAIR R., PRADEEP V., MILIND T. and MAKOTO Y., “Networked distributed POMDPs: a synthesis of distributed constraint optimization and POMDPs”, *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI'05)*, Pittsburgh, PA, pp. 133–139, 2005.
- [OLI 07a] OLIEHOEK F., KOOIJ J. and VLASSIS N., “A cross-entropy approach to solving Dec-POMDPs”, *Proceedings of the 1st International Symposium on Intelligent and Distributed Computing*, Craiova, Romania, pp. 145–154, 2007.
- [OLI 07b] OLIEHOEK F. and VLASSIS N., “Q-value functions for decentralized POMDPs”, *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'08)*, Honolulu, Hawaii, pp. 833–840, 2007.
- [PAP 87] PAPADIMITRIOU C. H. and TSITSIKLIS J. N., “The complexity of Markov decision processes”, *Mathematics of Operations Research*, vol. 12, no. 3, pp. 441–450, 1987.
- [PES 00] PESHKIN L., KIM K., MEULEAU N. and KAEUBLING L., “Learning to cooperate via policy search”, *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI'00)*, Morgan Kaufman, San Francisco, CA, pp. 489–496, 2000.
- [PUT 94] PUTERMAN M. L., *Markov decision processes: Discrete stochastic dynamic programming*, John Wiley & Sons, New York, 1994.
- [PYN 02] PYNADATH D. V. and TAMBE M., “The communicative multiagent team decision problem: analyzing teamwork theories and models”, *Journal of Artificial Intelligence Research*, vol. 16, pp. 389–423, 2002.
- [ROY 00] ROY N., PINEAU J. and THRUN S., “Spoken dialogue management using probabilistic reasoning”, *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL-2000)*, Hong Kong, pp. 93–100, 2000.
- [SCH 02] SCHERRER B. and CHARPILLETT F., “Cooperative co-learning: a model-based approach for solving multi-agent reinforcement problems”, *Proceedings of the 14th International Conference on Tools with Artificial Intelligence (ICTAI'02)*, pp. 463–468, 2002.
- [SEU 07] SEUKEN S. and ZILBERSTEIN S., “Improved memory-bounded dynamic programming for decentralized POMDPs”, *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence (UAI'07)*, Vancouver, British Columbia, Canada, 2007.
- [SEU 08] SEUKEN S. and ZILBERSTEIN S., “Formal models and algorithms for decentralized decision making under uncertainty”, *Autonomous Agents and Multi-Agent Systems*, vol. 17, no. 2, pp. 190–250, 2008.
- [SHA 53] SHAPLEY L. S., “Stochastic games”, *Proceedings of the National Academy of Sciences of the United States of America (PNAS)*, vol. 39, pp. 1095–1100, 1953.
- [SHE 06] SHEN J., BECKER R. and LESSER V., “Agent interaction in distributed MDPs and its implications on complexity”, *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multi-Agent Systems*, ACM Press, New York, pp. 529–536, 2006.
- [SPA 08] SPAAN M. T. J., OLIEHOEK F. A. and VLASSIS N. A., “Multiagent planning under uncertainty with stochastic communication delays”, *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS'08)*, Sydney, Australia, pp. 338–345, 2008.

- [SZE 05] SZER D., CHARPILLET F. and ZILBERSTEIN S., “MAA*: A heuristic search algorithm for solving decentralized POMDPs”, *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI'05)*, pp. 576–583, 2005.
- [SZE 06] SZER D. and CHARPILLET F., “Point-based dynamic programming for DEC-POMDPs”, *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI'06)*, Boston, MA, pp. 1233–1238, 2006.
- [TAO 01] TAO N., BAXTER J. and WEAVER L., “A multi-agent, policy-gradient approach to network routing”, *Proceedings of the 18th International Conference on Machine Learning (ICML'01)*, Morgan Kaufmann, San Francisco, CA, pp. 553–560, 2001.
- [WAS 96] WASHINGTON R., “Incremental Markov-model planning”, *Proceedings of the 8th International Conference on Tools with Artificial Intelligence (ICTAI'96)*, pp. 41–47, 1996.
- [XUA 01] XUAN P., LESSER V. and ZILBERSTEIN S., “Communication decisions in multi-agent cooperation: model and experiments”, *Proceedings of the 5th International Conference on Autonomous Agents (Agents'01)*, Montreal, Canada, pp. 616–623, 2001.

Chapter 10

Non-Standard Criteria

10.1. Introduction

Modeling and solving a sequential decision problem with an MDP require some strong assumptions: mono-criterion preference representation, complete and precise knowledge of the environment at each step, knowledge of the model itself, well-defined probabilistic uncertainty representation, etc.

Among those hypotheses, we have seen that some could be relaxed. POMDPs allow for taking into account a partial knowledge of the environment. Reinforcement learning methods make it possible to go without the knowledge of the model itself. In this chapter, we are more particularly interested in the two other limitations, those of a mono-criterion preference representation and of a well-defined probabilistic uncertainty representation.

More specifically, we start by describing the formalism of *multicriteria or vector-valued* MDPs [FUR 80, WHI 82, WAK 01] extending the MDP framework to multicriteria decision-making. In this framework, we present an algorithm [MOU 04] allowing the heuristic computation of *satisfying* policies that try to be close to an ideal solution.

Then we describe a first approach for solving an MDP whose model is ill-known. In this approach, called *robust*, [GIV 00, BAG 01, NIL 04, NIL 05] we stay in the standard probabilistic framework. However, we do not assume that the transition

Chapter written by Matthieu BOUSSARD, Maroua BOUZID, Abdel-Illah MOUADDIB, Régis SABBADIN and Paul WENG.

function and the reward function are perfectly known. Thanks to certain assumptions (knowledge of an interval for the probabilities and the rewards), a robust version of the value iteration algorithm can be proposed.

Yet, even the specification of intervals of probabilities for the transition function could be impossible in some situations. Similarly it could be arbitrary to specify numeric additive rewards for certain MDPs. In such cases, the knowledge and the preferences of an agent could be expressed more faithfully with a preorder on the likelihoods of transitions and on his preferences on those transitions. *Possibility theory* [DUB 88] offers a framework allowing the representation of such qualitative knowledge and preferences. Consequently, we present in this chapter an MDP approach based on possibility theory [SAB 98]. We show that this approach also allows for partial observability of the states (or of the model) to be taken into account in *possibilistic POMDPs* and that it allows for the resolution of compactly represented problems.

Finally we present the framework of *algebraic MDPs* [PER 05, WEN 06b] generalizing both multicriteria MDPs and possibilistic MDPs in the finite-horizon case. This framework not only unifies those approaches, but also highlights some algebraic conditions guaranteeing the validity of a dynamic programming algorithm. Those conditions can then be utilized to test if new preference representations and/or new uncertainty representations can be exploited together in planning under uncertainty.

EXAMPLE 10.1. If we come back to the recurrent example of the maintenance of a car (see section 1.1), the method described in this chapter makes it possible, among others, to solve the two following problems:

- it seems more realistic to estimate the consequences of an action by stating “if this oil leak is not taken care of, it is almost sure that the engine will break” than by claiming to know that “the probability that the engine breaks is 87.72%”. Possibility theory could be used to express the first statement in an MDP framework;

- instead of being only interested in the cost of an action, it is possible to take into account several criteria. For instance, we could want to optimize at the same time the cost, the car immobilization duration, the environmental impact and the garage clutter. This is exactly what multicriteria MDPs allow to model.

10.2. Multicriteria approaches

In this section, we present decision models making it possible to weaken one of the assumptions made in standard MDPs, which concerns the additive scalar reward function. Indeed, many problems are rather formalized by MDPs whose reward function is multi-dimensional, representing different criteria to be optimized. To deal with such problems, the formalism of multicriteria Markov decision

processes (2V-MDP¹) has been proposed. Section 10.2.1 recalls some basic notions in multicriteria decision-making, then section 10.2.2 presents the 2V-MDP formalism as well as the resolution algorithm. Then an illustrative example will be detailed to show how the algorithm works.

10.2.1. Multicriteria decision-making

Multicriteria decision-making [VIN 89, KEE 76] aims at selecting a preferred solution in an alternative set, taking into account several aspects (called attributes or criteria), such as price, quality, appearance, etc. Those criteria are generally contradictory as, for instance, it is difficult to maximize quality and minimize price in the same time. A classic way to compare alternatives, which are represented by vectors, is to use *Pareto-dominance*.

DEFINITION 10.1. A point $x = (c_1, c_2, \dots, c_i, \dots, c_n)$ of \mathbb{R}^n Pareto-dominates (or simply dominates) another point $x' = (c'_1, c'_2, \dots, c'_i, \dots, c'_n)$ of \mathbb{R}^n if and only if

$$\forall i, c_i \geq c'_i \quad \text{and} \quad \exists i, c_i > c'_i.$$

Pareto-dominance implies that an alternative can be “bad” with respect to one criterion without being considered a “bad” choice if it is “good” with respect to another criterion.

In Figure 10.1, the gray area corresponds to the points dominating point D_0 . Of course the solution of a multicriteria decision problem must not be dominated. Thus potential solutions must belong to the *Pareto-optimal set*.

DEFINITION 10.2. The Pareto-optimal set is made of non-dominated realizable solutions.

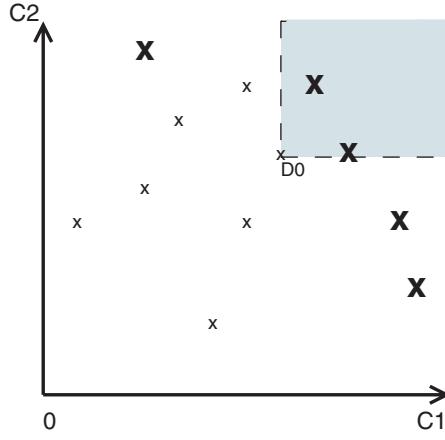
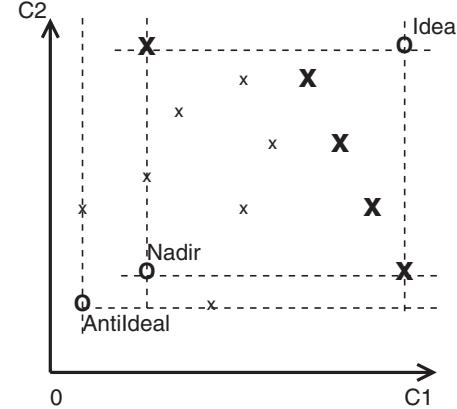
As the Pareto-dominance relation is a partial relation, it could be not very discriminating. Indeed there exist multicriteria problems for which all realizable solutions are in the Pareto-optimal set [HAN 80]. Therefore, it is interesting to consider relations refining Pareto-dominance.

For that matter, we introduce two reference points, the *ideal point* and the *anti-ideal point* [BRI 04].

DEFINITION 10.3. For maximization problems, the ideal point is defined as the point maximizing all criteria simultaneously. Formally, for a realizable solution set $E \subset \mathbb{R}^n$, it is defined by

$$\text{ideal}_i = \max_{x \in E} x_i,$$

1. Vector-valued Markov decision process.

**Figure 10.1.** Dominance with two criteria**Figure 10.2.** Ideal, anti-Ideal and Nadir point

where $ideal_i$ and x_i are the values on the i th criterion of the ideal point and the realizable solution x . Note that this reference point does not generally belong to the set of realizable solutions. The anti-ideal point is its opposite, that is to say it minimizes the criteria. It is used as a reference point for the worst choice:

$$antiIdeal_i = \min_{x \in E} x_i.$$

Those two points define a frame in which we can focus its search for a “good” solution (see Figure 10.2). For more details, the reader should refer to [GRA 02]. Note that, instead of using the anti-ideal point, a refinement would be to use the Nadir point. It is defined as the point minimizing all the criteria over the alternatives belonging to the Pareto-optimal set. As the set of Pareto-optima is difficult to calculate, so is the Nadir point. We will therefore not use this refinement here. In the following section, the sequential decision model is introduced.

10.2.2. Multicriteria MDPs

2V-MDPs [MOU 04] are an extension of MDPs where the decision-making process depends on several criteria. Those criteria are introduced through the reward function that becomes vector-valued. This change will have an impact over the rest of the formalism.

Let $Z = (z_1, z_2, \dots, z_n)$ be a vector of criteria, where each z_i represents the value of the i th criterion. An action a_j , taken in an action set $\mathcal{A} = \{a_1, a_2, \dots, a_m\}$, operates on a certain number of criteria (some or all) changing vector Z in

$Z' = (z'_1, z'_2, \dots, z'_n)$. We then have to modify the definition of a standard MDP in order to take into account those multiple criteria. Thus, a multicriteria MDP is defined by:

- a set of states S ,
- a set of actions A ,
- a transition function $p(s, a, s')$, $\forall s, s' \in S, a \in A$,
- a reward function $\overrightarrow{r(s)} = (r_1(s), r_2(s), \dots, r_i(s), \dots, r_n(s))$, where each $r_i(s)$ represents the reward obtained in state s for criterion i (in the same manner as in mono-criterion MDPs, except that here the reward does not depend on actions).

Now let us rewrite the Bellman equation that allows the determination of the optimal policy by computing the expectation of the sum of rewards $V(s)$ in each state s . In the mono-criterion formulation, it is written as follows:

$$\forall s \in S, \quad V(s) = R(s) + \max_a \sum_{s'} p(s, a, s') V(s').$$

In the multicriteria case, the reward function is vector-valued and thus so is the value function. Translating directly the previous equations, we would get $\forall s \in S$,

$$\vec{V}(s) = \begin{pmatrix} v_1(s) \\ v_2(s) \\ \dots \\ v_n(s) \end{pmatrix} = \begin{pmatrix} r_1(s) \\ r_2(s) \\ \dots \\ r_n(s) \end{pmatrix} + \max_a \sum_{s'} p(s, a, s') * \begin{pmatrix} v_1(s') \\ v_2(s') \\ \dots \\ v_n(s') \end{pmatrix},$$

where the v_i s, $i \in 1, \dots, n$, are the values taken by the different criteria. However, in general, the max operator cannot be directly used as an action does not necessarily maximize all criteria simultaneously. We then have to redefine the concept of convergence to a fixed point in this setting. Many approaches have been developed to solve this problem [WHI 82, FUR 80]. They generally rely on Pareto-dominance like the works of [WAK 01], which searches for all non-dominated paths. These approaches suffer from the fact that there could be an exponential number (in the number of states) of non-dominated policies [HAN 80].

10.2.2.1. Operators for multicriteria decision-making

Therefore, instead of enumerating all non-dominated policies, we could try to search for a “satisfying” policy. To that end, an approach could be to redefine the max operator used in the Bellman equation in order to obtain a unique policy at the end of the value iteration algorithm. The Chebyshev norm [BRI 04] could be used for that purpose.

DEFINITION 10.4. *Weighted Chebyshev norm:*

$$\forall p, q \in \mathbb{R}^n, \quad s_{\omega, q}^\infty(p) = \max_{i \in \{1, \dots, n\}} \omega_i \|p_i - q_i\|.$$

This norm defines a distance from any point p to a reference point q . Point q is taken here as the ideal point (Definition 10.3). Weights ω_i of this norm are used to normalize all the criteria. They are defined by

$$\omega_i = \frac{\alpha_i}{\|ideal_i - antiIdeal_i\|},$$

where α_i parameters allow to prioritize criteria [BOU 07]. The motivation to use this norm (with the normalization) is to be able to express, for each action, the regret that the agent has for having chosen an action instead of the ideal action. With this norm, a decision operator can be constructed, replacing the max operator of the Bellman equations in 2V-MDPs. However, it is known that the weighted Chebyshev norm is not compatible with the Bellman optimality principle [GAL 06] that states that sub-policies of optimal policies are optimal. Yet, after many experiments, the agents following this optimality criterion appear to show “good” behaviors.

Decision operator: LexDiff

For a criterion i , let v_i be the value of the current policy and v_i^* the value of the optimal policy (for criterion i).

A new vector is defined, called utility vector and denoted by \vec{V}^u , constructed from the weighted Chebyshev norm, representing for each criterion the normalized (by the ω_i s) distance to ideal point $q = (v_1^*, v_2^*, \dots, v_n^*)$ (weighted if necessary by the α_i s present in the ω_i s). Thus, for any state $s \in S$,

$$\vec{V}^u(s) = \begin{cases} v_0^u(s) = \omega_0 \cdot \|v_0(s) - v_0^*(s)\|, \\ v_1^u(s) = \omega_1 \cdot \|v_1(s) - v_1^*(s)\|, \\ \vdots \\ v_n^u(s) = \omega_n \cdot \|v_n(s) - v_n^*(s)\|. \end{cases}$$

The LexDiff operator over an alternative set consists of two stages. The first one is the computation of vectors V^u , and the second one selects the best choice with respect to a lexicographic order (leximin), from the greatest regret to the weakest. This operator guarantees a Pareto-optimal solution while preserving an egalitarian society. That is to say this operator will not choose a solution whose value on a criterion is too low even if, to that end, it has to decrease a little bit the global utility. A dynamic programming algorithm is used in order to reach a “good” policy (with regard to this

operator). A similar algorithm has been proposed by [DAE 80]. Algorithm 10.1 allows the computation of a policy with the LexDiff operator.

To determine the vector $\vec{V}^u(s)$, it is necessary to calculate the *ideal* and *antiIdeal* points. For the *ideal* point, n mono-criterion optimizations have to be performed. Each of these optimal values is calculated by value iteration for instance. For n criteria, we need to determine n optimal value functions. This does not necessarily increase the global computation time. Indeed, as all these optimizations are independent, it is easy to parallelize them. It is also possible to speed up this computation by optimizing one criterion while storing the values of the other criteria, which could be used to initialize the computation for these other criteria. The use of the *antiIdeal* point could lead to a bad normalization as its value can be lower on each criterion than those reachable by the Pareto-optimal solutions. In that sense, the *nadir* point would be fairer. However, it is difficult to calculate. That is why a classic heuristic can be used in order to get a close approximation of the *nadir* point: we will use the lowest value encountered during each optimization [EHR 03].

Algorithm 10.1: Solving 2V-MDP

```

for each criterion  $i$  do
    | Calculate  $V_i^*(s)$ 
    |  $V' \leftarrow 0$ 
for  $t = 0 \dots T - 1$  do
    |  $V \leftarrow V'$ 
    | for each  $s \in S$  do
        |   for each  $a \in A$  do
            |        $\vec{Q}^{tmp}(a) \leftarrow \vec{R}(s) + \sum_{s' \in S} P(s, a, s') \vec{V}(s')$ 
            |       for each criterion  $i$  do
            |           |    $ideal_i \leftarrow \max_a Q_i^{tmp}(a)$ 
            |           |    $antiIdeal_i \leftarrow \min_a Q_i^{tmp}(a)$ 
            |           |    $\omega_i \leftarrow \frac{\alpha_i}{ideal_i - antiIdeal_i}$ 
            |           |    $Q_i^u(a) \leftarrow \omega_i \cdot \|Q_i^{tmp}(a) - V_i^*(s)\|$ 
            |       ActionOpt  $\leftarrow \text{leximin}_a \vec{Q}^u(a)$ 
            |        $\vec{V}'(s) \leftarrow \vec{Q}^{tmp}(\text{ActionOpt})$ 
    | return  $V'$ 

```

EXAMPLE 10.2. Numerical application: Figure 10.3 presents an MDP with three actions $\{a, b, c\}$. The transition probabilities are displayed along the arcs and the

rewards are shown under the leaves. This MDP involves two criteria. The different decision-making steps are listed below:

- Computation of the *ideal* and *antiIdeal* points:

$$\text{ideal} = (-23; -3)$$

$$\text{nadir} = \text{antiIdeal} = (-30; -11)$$

- Computation of the ω_i s:

$$\omega_1 = \frac{1}{\| -23 - (-29) \|} = \frac{1}{6}$$

$$\omega_2 = \frac{1}{\| -3 - (-10) \|} = \frac{1}{7}$$

- Computation of V^u :

$$\vec{Q}^u(a) = (1; 0)$$

$$\vec{Q}^u(b) = (0; 1)$$

$$\vec{Q}^u(c) = \left(\frac{1}{3}; \frac{2}{7} \right)$$

- Sorting by lexicographic order:

$$\vec{Q}^u(a) = (1; 0)$$

$$\vec{Q}^u(b) = (0; 1)$$

$$\vec{Q}^u(c) = \left(\frac{2}{7}; \frac{1}{3} \right)$$

- Selection by leximin:

$$\text{LexDiff}(Q^u) = c$$

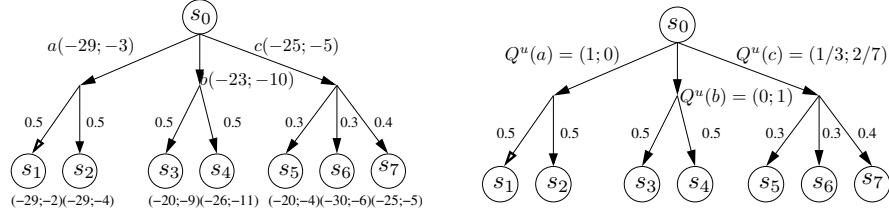


Figure 10.3. 2V-MDP: application of the LexDiff operator

To conclude, we have to keep in mind that in such a formalism, the real optimal policy – based on the Chebyshev norm – is difficult to calculate as the Bellman optimality principle is violated, the approach presented here then focuses on finding a “satisfying” policy. To that end, the LexDiff operator has been proposed. It should provide balanced solutions while not degrading too much the global utility. The proposed algorithm is one possible algorithm for solving multicriteria MDPs. It has advantages and drawbacks. To solve multicriteria MDPs, we therefore have to choose an algorithm in agreement with the type of policies that one wants to obtain.

10.3. Robustness in MDPs

It is common that the model of an MDP (transition and reward functions) is only known in an uncertain manner. The reasons for that are related to how the model is obtained: via a human expert or by statistical estimations. If we want to take into account this uncertainty, a first difficulty is to decide how to model it. But this difficulty is directly related to the problem that we want to solve, which could be:

- which part of the model is the most useful to improve to take the best decisions?
- how our actions should be planned taking into account that uncertainty?

The first problem is that encountered when searching for the best approximation of a model, discretizing states or actions or using approximating functions (see, e.g. [MUN 02]). We are interested here in the second problem, which can be solved with *robust planning*, i.e. by reformulating the planning goal as that of finding an optimal policy for the worst possible model. We face here a two-player zero-sum game (the “planner” vs. the “modeler”), which can be summarized by the following optimization problem:

$$\arg \max_{\pi \in \Pi} \min_{m \in M} V(\pi, m),$$

where Π is the policy set, M the set of possible models and $V(\pi, m)$ the value of policy π for model m (different criteria are possible).

Robust planning leads us to make two remarks about the reward function:

- When a reward can be related to a transition $(s, a) \rightarrow s': r(s, a, s')$, it is common to make rewards depend only on s and a : $r(s, a) = E_{s'}[r(s, a, s')]$. Here, since the true model is unknown, this expectation cannot be calculated. It is then better to keep the $r(s, a, s')$ formulation.
- $r(s, a, s')$ is an uncertain value. But among its possible values, the worst is always the smallest. We therefore assume that $r(s, a, s')$ takes its worst value.

Notice that robust planning is interested in the possible models, not in the probability distribution over those models. Thus we only need to specify the set of

possible models. A simple and practical choice is to measure uncertainty about a quantity (here $p(s' | s, a)$) with an interval:

$$p(s' | s, a) \in [P^{\min}(s' | s, a), P^{\max}(s' | s, a)].$$

Figure 10.4 illustrates this modeling of transition uncertainty from a state-action pair (s, a) . Here, a triangle is a simplex representing all the possible probability distributions for a transition with three reachable states ($P(s'_i) = 1$ in state s'_i). The trapezium on the left-hand side triangle corresponds to the constraint provided by the probability interval for s'_1 . On the right-hand side triangle, the possible models are at the intersection of the three constraints.

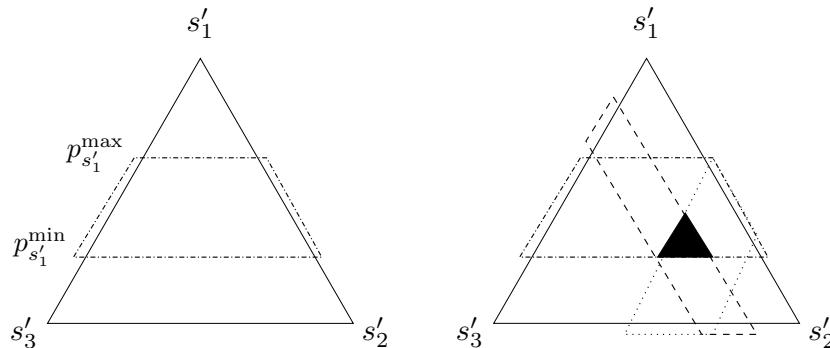


Figure 10.4. Modeling of transition uncertainty from a state-action pair (s, a) with intervals

Notice that on this figure the set of possible models for a state-action pair forms a convex polygon (when modeling with intervals). This convexity is a property often making the search for an optimum easier: if the function to be optimized is also convex, there is only one global optimum.

Robust planning as described previously allows for the use of algorithms optimizing decisions locally (i.e. at the level of a state) if the following condition holds.

PROPERTY 10.1. *The probability distribution $p(\cdot | s, a)$ is independent from a state-action pair to another.*

Assuming this hypothesis increases the number of possible models and allows the problem to be boiled down to an alternate game, where the planner and the modeler play in turn. We can then reformulate the value iteration algorithm under a robust form by alternating a minimizing step and a maximizing step, as shown in Algorithm 10.2 with the γ -weighted criterion.

Algorithm 10.2: Robust value iteration

```

Initialize  $V_0 \in \mathcal{V}$ 
 $n \leftarrow 0$ 
repeat
  for each  $s \in S$  do
    for each  $a \in A$  do
       $Q_{n+1}(s, a) \leftarrow$ 
       $\min_{m_s^a \in \mathcal{M}_s^a} \sum_{s' \in S} p_{m_s^a}(s' | s, a) [r(s, a, s') + \gamma V_n(s')]$ 
       $V_{n+1}(s) \leftarrow \max_{a \in A} Q_{n+1}(s, a)$ 
     $n \leftarrow n + 1$ 
  until  $\|V_{n+1} - V_n\| < \epsilon$ 
  for each  $s \in S$  do
     $\pi(s) \in \arg \max_{a \in A} Q_n(s, a)$ 
return  $V_n, \pi$ 

```

Like in a standard MDP, there always exists an optimal deterministic policy for the planner. Moreover, if the set of possible models is convex, a worst model exists at one of the extremities of this set (a polygon vertex when using intervals). This allows the definition of simple procedures for finding the worst local model.

The reader interested in robust planning can find more details in the following literature [GIV 00, BAG 01, NIL 04, NIL 05, TRE 07]. The main difficulty is related to Property 10.1. There are many situations in which it does not hold, like in planning with concurrent tasks (see Chapter 15). In such cases, dynamic programming cannot be used anymore. We could then resort to direct policy search approaches for instance [BUF 05].

10.4. Possibilistic MDPs

One of the contributions of artificial intelligence to decision theory in general has been the proposition and the study of decision criteria alternative to the standard criterion of expected utility. Among those alternative decision criteria, “qualitative” criteria, more adapted to the preoccupations of artificial intelligence (knowledge/preference elicitation, man/machine communication), have been used in the framework of sequential decision-making under uncertainty. In particular, a qualitative counterpart of MDPs/POMDPs [FAR 98, SAB 01a] has been proposed recently. We describe this model in the present section.

10.4.1. Possibilistic counterpart of expected utility

A possibility distribution describes the knowledge that we have of the value taken by one or several ill-known attributes describing the state of the system. For instance, the age of a man, the height of a building, etc. In our case, a possibility distribution is used to model the imperfect knowledge that we have about the world in a decision-making problem under uncertainty, distinguishing “plausible” or normal states from unlikely or surprising states.

More formally, a possibility distribution π over a set S of states is an application from S to $(L, <)$, a finite or bounded ordered scale. This scale is assumed to be endowed with an order-reversing map n , bijection from L to L such that if $\alpha > \beta \in L$, then $n(\beta) > n(\alpha)$. 1_L and 0_L , respectively, represent the greatest and the smallest elements of L , and $n(0_L) = 1_L$ and $n(1_L) = 0_L$. If $L = [0, 1]$, we generally have $n(\cdot) = 1 - \cdot$.

The function $\pi : S \rightarrow L$ models imperfect knowledge with the following conventions:

- $\pi(s) = 0_L$ means that s is considered impossible;
- $\pi(s) = 1_L$ means that s is a “normal” or totally possible state;
- $\pi(s) > \pi(s')$ means that s is more likely than s' .

Notice that several states could have a possibility of 1_L : this would mean that several states are equally plausible and that they are more plausible than all other states. The extreme case of ignorance is that when all the states share a possibility of 1_L : all the states are possible and nothing can discriminate between them. On the contrary, if only one state has a possibility of 1_L and all the others are impossible (possibility of 0_L), then we are in a state of perfect knowledge. This knowledge description in terms of possibility distribution is quite flexible in so far as all the degrees of scale L included between 0_L and 1_L are usable to model the degrees of possibility $\pi(s)$ of the various states. In general, and it will be so in this chapter, the only constraint on distribution π is that there exists a state s of possibility 1_L (normalization): whatever our knowledge on the state of the world, there exists at least one state accepted as “normal”.

A possibility distribution can be used to model incomplete knowledge about the real state of the world. However a different interpretation of this function can be given in terms of *preferences* on the state of the world: In this case, $\pi(s)$ represents the degree at which s is a desirable situation for an agent (see [DUB 96] for a detailed discussion about the interpretation of a possibility distribution in terms of preferences). We now examine the case where two possibility distributions are jointly used to model knowledge and preferences in problems of qualitative decision-making under uncertainty.

The authors of [DUB 95] proposed an ordinal counterpart of expected utility theory based on possibility theory. In the framework of one-shot decision-making, S and X are, respectively, the (finite) sets of possible states of the world and possible consequences of actions. Assuming that the pieces of information about the decision-maker's knowledge and preferences are qualitative, it is reasonable to represent both the incomplete knowledge about the state of the world by a possibility distribution π over S and the gradual preferences over consequences by another possibility distribution μ over X , those two distributions taking their values on the same finite totally ordered scale L whose smallest and greatest elements are, respectively, 0_L and 1_L . The existence of this shared scale can be naturally justified with an axiomatization of the probabilistic decision criteria according to Savage, as proposed in [DUB 98].

The agent's uncertainty about the effect of an action a performed in the state of the world s is represented by possibility distribution $\pi(\cdot | s, a) : X \rightarrow L$. The distribution $\pi(x | s, a)$ measures to which extent x is a plausible consequence of action a applied in s . $\pi(x | s, a) = 1_L$ means that x is a completely plausible consequence, whereas $\pi(x | s, a) = 0_L$ means x is impossible.

Similarly, the consequences are also ordered in terms of satisfaction level by a qualitative utility function $\mu : S \times A \times X \rightarrow L$. $\mu(s, a, x) = 1_L$ means that x is a completely satisfying consequence of a in s , whereas $\mu(s, a, x) = 0_L$ means that x is absolutely not satisfying. Notice that we always assume that π is normalized, but μ could very well not be normalized (nothing guarantees, for a given decision problem, that a totally satisfying consequence can be reached).

[DUB 95, DUB 98] proposed and axiomatized the two following criteria:

$$u^*(a, s) = \max_{x \in X} \min \{ \pi(x | s, a), \mu(s, a, x) \}, \quad (10.1)$$

$$u_*(a, s) = \min_{x \in X} \max \{ n(\pi(x | s, a)), \mu(s, a, x) \}, \quad (10.2)$$

where n is an order-reversing map of L .

u^* can be interpreted as an extension of the *maximax* criterion, which values the state-action pairs with the utility of the best consequence possible, whereas u_* is an extension of the *maximin* criterion, which values them with the worst consequence possible. Using u^* corresponds to an optimistic attitude (we focus on the best possible consequences of an action, ignoring the worst ones), whereas u_* corresponds to the cautious attitude (we focus on the worst possible consequences of an action, ignoring the best ones).

Qualitative possibilistic utilities, though very different from expected utility, are not totally distinct from it. As a matter of fact, it is possible to prove that preference relations based on possibilistic optimistic and pessimistic utilities can always be refined by a preference relation based on expected utility, which itself could be qualitatively and uniquely expressed with possibility distributions and qualitative utility [FAR 03, FAR 05].

EXAMPLE 10.3. Consider the example proposed by [SAV 54, pages 13–15] to illustrate the expected utility criterion: the problem is about cooking an omelet. We have already broken five eggs in a bowl and we have in hand a sixth egg whose freshness is uncertain. Three actions are available: break the egg in the omelet (BO), break it apart in a cup (BC) or directly throw it away (T). Assume that the finite scale $L = T = \{0, a, b, c, d, 1\}$ (where $0 < a < b < c < d < 1$), endowed with an order-reversing map n , is sufficient to conjointly express the uncertainty about the state of the world and the preferences about consequences. In particular, the consequences are ordered in terms of preferences in Table 10.1.

Actions/States	Fresh egg (F)	Rotten egg (R)
BO	6-egg omelet (1)	Nothing to eat (0)
BC	6-egg omelet, cup to wash (d)	5-egg omelet, cup to wash (b)
T	5-egg omelet, 1 spoiled egg (a)	5-egg omelet (c)

Table 10.1. States, actions and consequences in Savage's omelet example

The degrees between braces represent an intuitive encoding of the preference order between consequences. Two states of the world are possible (fresh (F), rotten (R)), of possibility, respectively, $\pi(F)$ and $\pi(P)$, with $\max(\pi(F), \pi(P)) = 1$ (normalization of the possibility distribution representing uncertainty). The computation of the two criteria yields

$$\begin{aligned} u_*(BO) &= \min(1, \max(n(\pi(R)), 0)) = n(\pi(R)), \\ u^*(BO) &= \pi(F), \\ u_*(BC) &= \min(d, \max(n(\pi(R)), b)), \\ u^*(BC) &= \max(\min(\pi(F), d), b), \\ u_*(T) &= \min(\max(n(\pi(F)), a), c), \\ u^*(T) &= \max(a, \min(\pi(R), c)). \end{aligned}$$

Criterion u_* recommends cautiousness (BC) as soon as we are ignorant about the state of the egg ($\pi(R)$ and $\pi(F)$ greater than or equal to $n(a) = d$), which seems to be more realistic than the qualitative theories suggesting to forget the states that are not the most plausible (like in [BOU 94]). Those theories, focusing only on either state (F) or state (R), never recommend action (BC), which seems to be the most “intuitive” one. Notice that, in this example, the optimistic attitude is less “intuitive” since it recommends action (BO) in case of uncertainty.

10.4.2. Possibilistic dynamic programming

10.4.2.1. Finite horizon

In [FAR 98], possibilistic decision theory has been extended to the sequential case in finite horizon N . In this framework, the utility of a policy δ in an initial state s_0 is defined depending on the case (pessimistic or optimistic) by a qualitative utility criterion applied to the possible trajectories (and not to the possible states/consequences):

$$u_*(\delta, s_0) = \min_{\tau} \max \{n(\pi(\tau | s_0, \delta)), \mu(\tau, \delta)\}, \quad (10.3)$$

$$u^*(\delta, s_0) = \max_{\tau} \min \{\pi(\tau | s_0, \delta), \mu(\tau, \delta)\}, \quad (10.4)$$

where, if $\tau = \{s_0, \dots, s_N\}$ and $\delta = (d_0, \dots, d_{N-1})$ ($\forall i = 0 \dots N-1, d_i \in A^S$),

$$\mu(\tau, \delta) = \underset{i=0 \dots N-1}{*} \mu(s_i, \delta(s_i), s_{i+1}),$$

$$\pi(\tau | s_0, \delta) = \min_{i=0 \dots N-1} \pi(s_{i+1} | s_i, d_i(s_i)).$$

$*$ is an operator aggregating the preference degrees associated with each transition. In practice, in the finite horizon case, either $*_{i=0 \dots N-1} \mu(s_i, \delta(s_i), s_{i+1}) = \min_{i=0 \dots N} \mu(s_i)$, or $*_{i=0 \dots N-1} \mu(s_i, \delta(s_i), s_{i+1}) = \mu(s_N)$.

To have an intuitive idea of those criteria in sequential decision-making, consider the following simplified cases:

1) the transition possibilities take only degrees 0_L or 1_L , just like the utility degrees, which are only associated with the final state: in this case, the pessimistic criterion yields the maximal utility to any strategy that only induces trajectories whose final state is satisfying; the optimistic criterion selects strategies that induce at least one trajectory leading to a satisfying state;

2) same case, but the aggregating operator $*$ is the *min* operator: satisfying strategies are either those that only induce trajectories whose transitions are all satisfying (pessimistic case) or those that induce at least one trajectory whose transitions are all satisfying (optimistic case);

3) transition possibilities 0_L or 1_L , but preferences can take any value in scale L : the satisfaction degree of a strategy is either the satisfaction degree of the worst possible trajectory (pessimistic case) or that of the best possible trajectory (optimistic case) where the satisfaction degree of a trajectory is either the satisfaction degree of its final state or the satisfaction degree of its worst transition;

4) general case: the possibilistic one-shot decision criteria are used but the state space is replaced by the trajectory set and the consequence space is replaced either by S_N , or by the N -tuple space of transitions and the action space is replaced by the policy set.

The possibilistic (pessimistic and optimistic) counterparts of the Bellman equation are:

– in the pessimistic case (for $*$ $\equiv \min$):

$$\begin{aligned} u_*^t(s) &= \max_{a \in A_s} \min_{s' \in S_{t+1}} \min \{ \mu(s, a, s'), \max \{ n(\pi(s' | s, a)), u_*^{t+1}(s') \} \}, \\ u_*^N(s) &= \mu(s); \end{aligned} \quad (10.5)$$

– in the optimistic case:

$$\begin{aligned} u^{*t}(s) &= \max_{a \in A_s} \max_{s' \in S_{t+1}} \min \{ \mu(s, a, s'), \pi(s' | s, a), u^{*t+1}(s') \}, \\ u^{*N}(s) &= \mu(s). \end{aligned} \quad (10.6)$$

[FAR 98] shows that policies calculated by backward induction by successive applications of (10.5) (resp., (10.6)) optimize criterion u_* (resp., u^*).

Notice that, due to the idempotency of operator \min , backward induction only yields a subset of the set of policies maximizing u_* (resp., u^*). However, those policies satisfy the property of *dynamic consistency* and more specifically the Bellman optimality principle: any subpolicy (from t' to N) of an optimal policy from t to N (with $t' \geq t$) is optimal for the chosen criterion (see [FAR 98]).

10.4.2.2. Value iteration

We now consider, in the qualitative (possibilistic) framework, stationary problems in the infinite horizon case. To be more exact, given that there does not exist a correspondence in the possibilistic case with the γ -weighted criterion, that allows the association of a preference degree with a trajectory of infinite length, we limit ourselves in the possibilistic case to the case of problems of indefinite horizon: we assume the existence of a possibilistic utility function μ on terminal states of trajectories, an arbitrary action *no-op* leaving the system in its current state and we

look for a strategy allowing the system to be brought in a satisfying final state (surely or “possibly” depending on our pessimistic or optimistic attitudes), if necessary performing only satisfying transitions.

A possibilistic version of the value iteration can be defined to solve this kind of problem. This algorithm [SAB 01a] uses a possibilistic version $\tilde{Q}(s, a)$ of the Q function used in reinforcement learning. $\tilde{Q}(s, a)$ yields the (pessimistic or optimistic) utility of action a in state s .

As in the stochastic case, optimal possibilistic strategies can be determined by iterating the following updating equations:

– pessimistic case:

$$\tilde{Q}_{t+1}(s, a) = \min_{s' \in S} \min \left\{ \mu(s, a, s'), \max \left\{ n(\pi(s' | s, a)), u_{*t}(s') \right\} \right\}, \quad (10.7)$$

where $u_{*t}(s) = \max_a \tilde{Q}_t^*(s, a)$ and $\tilde{Q}_t^*(s, no-op) = \mu(s)$;

– optimistic case:

$$\tilde{Q}_{t+1}^*(s, a) = \max_{s' \in S} \min \left\{ \mu(s, a, s'), \pi(s' | s, a), u_t^*(s') \right\}, \quad (10.8)$$

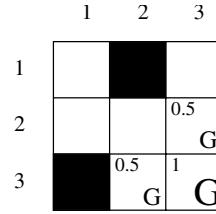
where $u_t^*(s) = \max_a \tilde{Q}_t^*(s, a)$ and $\tilde{Q}_t^*(s, no-op) = \mu(s)$.

This algorithm converges in a finite number of iterations (the algorithm stops as soon as $\tilde{Q}_{t+1} = \tilde{Q}_t$). This can be easily proved by noticing that the sequence of functions $(\tilde{Q}_t^*)_t$ is non-decreasing and takes its values in the finite set L .

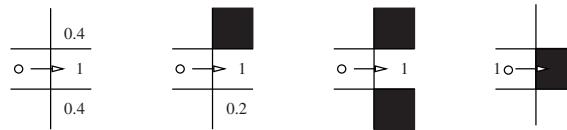
By the way, notice that the number of iterations is bounded by the size of the set of Q functions: $|A| \times |S| \times |L|$. Since an iteration of the algorithm requires $|S| \times |A|$ updates, the complexity of searching for an optimal policy is $O(|S|^2 \times |A|^2 \times |L|)$.

Also remark that, contrary to the stochastic value iteration algorithm, the initialization of u_* (or u^*) cannot be arbitrary (function μ over S is used for the initialization).

EXAMPLE 10.4. Consider the example of Figure 10.5, in which a robot has to reach the bottom right corner of the figure. A policy leading to one of the squares next to the bottom right corner would be partially satisfying. The black squares of the figure represent obstacles. The utility function μ associated with the problem (satisfaction degrees are only associated with final states of the system) also represented in Figure 10.5, is defined by $\mu(s_{33}) = 1$, $\mu(s_{23}) = \mu(s_{32}) = 0,5$ and $\mu(s) = 0$ for the other states.

**Figure 10.5.** State space and utility function

The available actions are those moving the robot towards the (T)op, (B)ottom, (R)ight and (L)eft or (S)tay in place. If the robot chooses to (S)tay in place, its position remains identical with certainty. On the contrary, if it chooses one of the other actions, it would move towards the desired square with maximal possibility ($\pi = 1$), but it could eventually drift to one of the neighboring squares with possibility degrees given in Figure 10.6 (for action (R), the others being obtained by symmetry).

**Figure 10.6.** Transition possibilities for action (R)

If the chosen destination square is an obstacle, the robot's position does not change (as if action (S) had been chosen).

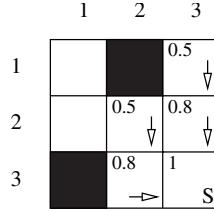
Let us calculate now the (pessimistic) policy obtained after one updating iteration (equation (10.7)). For any pair (s, a) , we get

$$\tilde{Q}^1(s, a) = \min_{s' \in S} \max \left(1 - \pi(s' | s, a), \mu(s') \right)$$

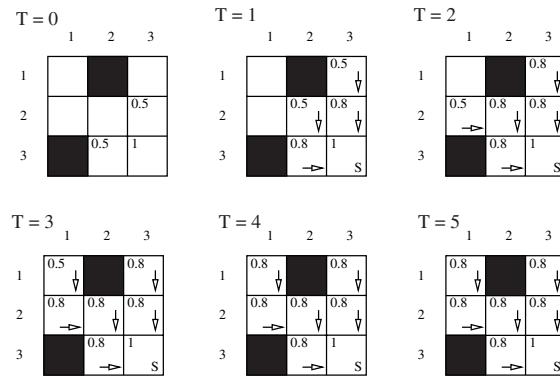
(π does not depend on the computation step) and

$$u_*(s) = \max_{a \in \{H, B, G, D, R\}} \tilde{Q}^1(s, a).$$

Figure 10.7 describes the pessimistic utility of each action after one iteration, as well as the current policy calculated for each state whose utility is non-null. The action returned for each state is unique, except for states s_{33} and s_{22} for which (B) and (R) could be chosen.

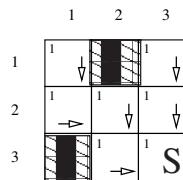
**Figure 10.7.** Policy calculated after one iteration

It is sufficient now to iterate again the updates until the convergence of the value function. The process is described in Figure 10.8, where we notice that the convergence is obtained after five iterations.

**Figure 10.8.** Iterative computation of an optimal pessimistic policy

The number of iterations required to calculate an optimal policy is of the order of the length of the longest deterministic path linking a start state to the goal state. This number is always lower than the size of the state space.

In this example, the optimal optimistic policy is identical to the optimal pessimistic policy. Only the associated value function (represented in Figure 10.9) differs.

**Figure 10.9.** Optimal optimistic policy

10.4.2.3. Policy iteration

A possibilistic version of the policy iteration algorithm can also be defined. This algorithm (here in the case where there is no intermediate utility) alternates, like the standard algorithm for MDPs, *evaluation* phases and phases *improving* the current policy:

- evaluation:
repeat until convergence of u_*^δ :

$$\forall s \in S, \quad u_*^\delta(s) \leftarrow \min_{s' \in S} \max \{ n(\pi(s' | s, \delta(s))), u_*^\delta(s') \}; \quad (10.9)$$

- improvement:

$$\forall s \in S, \quad \delta(s) \leftarrow \operatorname{argmax}_{a \in A} \min_{s' \in S} \max \{ n(\pi(s' | s, a)), u_*^\delta(s') \}. \quad (10.10)$$

Exactly like the value iteration algorithm, the initialization of the value function cannot be arbitrary (the function is initialized by the utility function over goals). An “optimistic” version of the policy iteration algorithm is obtained in the same manner as for the pessimistic case.

10.4.3. Extensions of possibilistic MDPs

Possibilistic MDPs have been extended to face the limitations similar to those of the standard MDP framework. More specifically, the three following extensions have been proposed:

– *Reinforcement learning*. Some qualitative decision problems under uncertainty combine both a qualitative preference representation (preorder on preferences) and a partial uncertainty representation, only accessible via transition simulations, or their experimentations. *Reinforcement learning* like methods have been proposed to solve those problems.

– *Possibilistic POMDPs*. The assumption of complete or partial observability of the state of the world is not related to the model used to represent uncertainty. Possibilistic MDPs have thus been extended to take into account the partial observability inherent to some problems.

– *Possibilistic influence diagrams*. The possibilistic uncertainty representation is often better adapted than the probabilistic one to reasoning about structured knowledge, due to the operators (min and max) used in the reasoning tools. It was therefore natural to extend possibilistic MDPs to structured representations of knowledge and preferences. Recently, a possibilistic counterpart of influence diagrams has thus been proposed, including algorithmic tools for solving those problems.

In this section we briefly describe the results obtained on these three points.

10.4.3.1. Possibilistic reinforcement learning

[SAB 01b] has proposed possibilistic versions of *indirect* methods for reinforcement learning: *certainty equivalent* and *prioritized sweeping*. The mathematical properties of the qualitative utility operators do not allow the definition of *direct* algorithms for reinforcement learning (TD-lambda, Q-learning) and therefore only indirect methods have been developed.

The problem of possibilistic reinforcement learning is to define an estimator $\hat{\pi}_t(s' | s, a)$ of $\pi_t(s' | s, a)$, where π and $\hat{\pi}$ belong to a finite ordinal scale L . In the literature, many transformation operators between probabilities and possibilities exist. Those operators can be classified into two categories:

- the first family [DAR 94, HEN 99] is based on an interpretation of possibility degrees in terms of “infinitesimal probabilities”;
- the second family [GIA 99, DUB 93] is based on the principle of *consistent* transformations between probability and possibility; a transformation is consistent as soon as $\forall A, B \subseteq S, P(A) \leq P(B) \Rightarrow \Pi(A) \leq \Pi(B)$.²

The simplest (but also the less efficient) learning method of optimal possibilistic policies is the *certainty equivalent* method that consists of learning $\hat{\pi}$ and $\hat{\mu}$ by exhaustive exploration of $S \times A$ before applying a possibilistic value or policy iteration algorithm. This method is inefficient as, like in the stochastic case, it dedicates the same effort to the whole state space, although some little plausible states have a low influence on the global value of a policy and some state-action pairs can be very quickly considered “bad”. It can be improved by alternating phases for learning the model and phases for updating the possibilistic value function, thus constituting a kind of possibilistic *prioritized sweeping* algorithm.

The *possibilistic prioritized sweeping* algorithm (PPS) is similar to the probabilistic algorithm, except that (like for the possibilistic value iteration algorithm) the current policy is saved in memory with the current value function. Each time that an action a is applied in a state s and that variations of $\hat{\pi}$ and $\hat{\mu}$ are observed which are sufficient to change the current value of $\hat{Q}(s, a)$, those variations are propagated to the *predecessors* of s . If the value of predecessors are modified, those modifications are also propagated and so on. The propagation is made thanks to a FIFO queue containing the predecessors to be modified. The size of the queue is bounded, as well as the number of updates by observed transition.

2. Notice that transformations based on infinitesimal probabilities are not necessarily consistent, except when $\epsilon \rightarrow 0$, in which case the obtained possibility distributions tend to be “all or nothing” distributions.

Unfortunately the policy returned by the PPS algorithm is not always optimal. This is due to the process of action allocation: a new action $\hat{a}^*(s_{\text{loc}})$ is associated by the algorithm to the current state s_{loc} each time the current value $\hat{u}^*(s_{\text{loc}})$ is modified by a trial. Now, while learning, the current value $\hat{u}^*(s_{\text{loc}})$ could possibly become equal to the optimal value, whereas the current model $\hat{\pi}, \hat{\mu}$ is not correct yet, which implies that current action $\hat{a}^*(s_{\text{loc}})$ is not necessarily optimal. If afterwards the current value function does not change anymore whereas the model keeps changing, the current action will not be modified anymore. The mean used in [SAB 01b] to solve this problem consists of using the PPS algorithm to calculate a sub-optimal policy, then running a probabilistic policy iteration algorithm starting with this policy and using the current estimations $\hat{\pi}$ and $\hat{\mu}$. This allows obtaining the policy optimality when the number of trials allocated to PPS raises. In practice, we notice that the policy calculated by PPS is “nearly” optimal and that a very small number of iterations are then necessary to obtain an optimal policy. PPS+policy iteration allows an optimal policy to be calculated faster than by the probabilistic *certainty equivalent* algorithm.

10.4.3.2. Probabilistic partially observable MDPs

The notion of conditioning has been studied in the framework of possibility theory (see [DUB 94] for a complete presentation). Conditioning relative to an event takes a similar form as that of Bayesian conditioning:

$$\forall A, B, \quad \Pi(A \cap B) = \min \{ \Pi(B | A), \Pi(A) \}. \quad (10.11)$$

Contrary to the Bayesian conditioning case, equation (10.11) does not have a unique solution $\Pi(B|A)$. Therefore, in general, the least specific³ solution of equation (10.11) is

$$\Pi(B|A) = \begin{cases} 1_L & \text{if } \Pi(A \cap B) = \Pi(A) > 0_L, \\ \Pi(A \cap B) & \text{otherwise.} \end{cases} \quad (10.12)$$

Once the choice is made for the conditioning of the possibility measure, the conditioning $\pi(\cdot | o)$ of a possibility distribution by an observation $o \in \Omega$ is immediately defined by

$$\pi(s | o) = \begin{cases} 1_L & \text{if } \pi(s, o) = \Pi(o), \\ \pi(s, o) & \text{otherwise,} \end{cases} \quad (10.13)$$

where $\Pi(o) = \max_s \pi(s, o)$ and $\pi(\cdot, \cdot)$ is the joint possibility distribution on $S \times \Omega$.

3. If $\Pi(A \cap B) = \Pi(A) < 1_L$, then $\forall \alpha \geq \Pi(A \cap B)$, $\Pi(B | A) = \alpha$ satisfies equation (10.11) (when $*$ = min).

DEFINITION 10.5 (possibilistic partially observable MDP). *From this definition of possibilistic conditioning, a possibilistic POMDP can be easily defined [SAB 99], in the same manner as in the stochastic framework. In the possibilistic framework, a possibilistic POMDP (Π -POMDP) can be turned into a possibilistic MDP, like in the probabilistic case. However, in this case, the state space remains finite, allowing the application of the previously described iterative algorithms: a possibilistic belief state β is a possibility distribution over state space S . Contrary to the stochastic case, the set of possibilistic belief states is finite as soon as the scale L used to measure possibility degrees is finite. The cardinal of B , the set of possibilistic belief states, is bounded from above by $|L|^{|S|}$.*

Assume now, as in the probabilistic case, that the transition possibilities $\pi(s' | s, a)$ are given, as well as the observation possibilities, $\pi(o | s, a)$. Then we can define $\beta_a(s')$, the possibility of reaching s' starting from a knowledge on the initial state defined by β and applying action a :

$$\beta_a(s') = \max_{s \in S} \min \{ \pi(s' | s, a), \beta(s) \}. \quad (10.14)$$

We then calculate the possibility of observing $o \in \Omega$ after having applied a in β :

$$\beta_a(o) = \max_{s \in S} \min \{ \pi(o | s, a), \beta_a(s) \}. \quad (10.15)$$

Now β_a^o is the revised possibilistic belief state after having applied a in β and observed o :

$$\beta_a^o(s) = \begin{cases} 0_L & \text{if } \pi(o | s, a) = 0_L, \\ 1_L & \text{if } \pi(o | s, a) = \beta_a(o) > 0_L, \\ \beta_a(s) & \text{in all other cases.} \end{cases} \quad (10.16)$$

All the elements of the new possibilistic MDP on the space of belief states are defined in equations (10.14), (10.15) and (10.16). Intuitively, the evolution of the system is defined as follows: If the system is in state β , then applying action a could lead to one of the $|\Omega|$ possible successor states β_a^o , the possibility of reaching state β_a^o being $\beta_a(o) = \pi(o | \beta, a)$.

From this point, the possibilistic Bellman equation can be extended to the partially observable case (here in the pessimistic case)⁴:

$$u_*^t(\beta) = \max_{a \in A_s} \min \{ \mu(\beta), \min_{o \in O} \max \{ n(\beta_a(o)), u_*^{t+1}(\beta_a^o) \} \}, \quad (10.17)$$

where $\mu(\beta) = \min_{s \in S} \max \{ n(\beta(s)), \mu(s) \}$ and $u_*^0(\beta)$ is initialized to $\mu(\beta)$.

4. For the sake of simplifying the notations, we limit ourselves to a utility function μ on states and not on transitions. Obviously equation (10.17) can be extended to take into account preferences over transitions.

10.4.3.3. Possibilistic influence diagrams (PID)

The framework of *possibilistic influence diagrams* (PID), the possibilistic counterpart of *influence diagrams*, has been recently defined [GAR 06]. The graphical part of a PID is exactly the same as that of usual influence diagrams but the semantic differs. The transition likelihoods are expressed by possibility distributions and rewards are considered here as satisfaction degrees attached to partial goals. The expected utility is then replaced by one of the two previously presented possibilistic qualitative utility criteria.

The possibilistic dynamic programming algorithms (*backward induction*, since the horizon is finite) are applicable to solve a problem expressed in the form of a PID. However, they require exponential time resources to calculate the utility of an optimal possibilistic strategy.⁵ [GAR 07, GAR 08] have proven that the computation of the utility of an optimal strategy for PID is NP-complete in the optimistic case⁶ and PSPACE-complete in the pessimistic case. They have proposed two classes of algorithms, respectively, based on the exploration of a decision tree or on variable elimination, allowing problems expressed as a DIP to be solved. Independently, a more general algebraic model for (structured) decision-making under uncertainty has been proposed [PRA 06], also presenting algorithms of the same family for a larger class of problems.

10.5. Algebraic MDPs

We now present a framework extending both that of multicriteria MDPs and that of possibilistic MDPs. In a view to study planning problems using a non-probabilistic uncertainty representation and/or a non-standard preference representation over actions (non-necessarily scalar additive rewards), we introduce the general framework of algebraic MDPs proposed by [PER 05, WEN 06b].

Before presenting this formalism, we briefly recall the required tools: semirings, plausibility measures and generalized expected utility. Then we formally present algebraic MDPs. Under certain conditions that we specify, an algorithm based on backward induction can be used to determine non-dominated solutions. This preliminary study is restricted to the finite-horizon case (finite number of decision steps).

5. And an exponential space to represent it.

6. As a matter of fact, it is the decision problem associated with this optimization problem that is NP-complete.

10.5.1. Background

10.5.1.1. Semirings

For the definition of an algebraic MDP (AMDP), we introduce two valuation scales V and P , for measuring, respectively, rewards and uncertainty. They are assumed to be endowed with *semiring* structure (see [GON 01] for a more thorough presentation).

DEFINITION 10.6. A semiring $(X, \oplus_X, \otimes_X, 0_X, 1_X)$ is a set X endowed with two operators \oplus_X and \otimes_X that satisfy the following conditions:

- $(X, \oplus_X, 0_X)$ is a commutative monoid with 0_X as a neutral element, i.e.

$$a \oplus_X b = b \oplus_X a,$$

$$(a \oplus_X b) \oplus_X c = a \oplus_X (b \oplus_X c),$$

$$a \oplus_X 0_X = a;$$

- $(X, \otimes_X, 1_X)$ is a monoid with 1_X as a neutral element and 0_X as an absorbing element, i.e.

$$(a \otimes_X b) \otimes_X c = a \otimes_X (b \otimes_X c),$$

$$1_X \otimes_X a = a \otimes_X 1_X = a,$$

$$0_X \otimes_X a = a \otimes_X 0_X = 0_X;$$

- \otimes_X is distributive over \oplus_X , i.e.

$$(a \oplus_X b) \otimes_X c = (a \otimes_X c) \oplus_X (b \otimes_X c), \quad (10.18)$$

$$a \otimes_X (b \oplus_X c) = (a \otimes_X b) \oplus_X (a \otimes_X c). \quad (10.19)$$

Operator \oplus_X induces a (non-necessarily total) preorder \geq_X , named *canonic preorder* as follows:

$$\forall x, y \in X, \quad x \geq_X y \iff \exists z \in X, \quad x = z \oplus_X y.$$

The semiring is called *idempotent* when operator \oplus_X is idempotent (i.e. $\forall x \in X, x \oplus_X x = x$). In that case, the canonic preorder \geq_X associated with X is an order.

Therefore the valuation scale for rewards V is supposed to be endowed with a structure of idempotent semiring $(V, \oplus_V, \otimes_V, 0_V, 1_V)$. Intuitively operator \oplus_V is used for the selection of preferred elements and operator \otimes_V allows the combination of elements. The valuation scale for uncertainty P is supposed to be endowed with the structure of semiring $(P, \oplus, \otimes, 0_P, 1_P)$. The interpretation of

operators \oplus and \otimes is given in the next section. Besides, in order to simplify the exposition we assume that canonic preorder over P is an order. As an example, in standard MDPs, we have $(V, \oplus_V, \otimes_V, 0_V, 1_V) = (\mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0)$ and $(P, \oplus, \otimes, 0_P, 1_P) = ([0, +\infty[, +, \times, 0, 1)$.

10.5.1.2. Plausibility measures

To model uncertainty related to consequences of an action, we resort to plausibility measures⁷ proposed by [FRI 95] generalizing most uncertainty representations.

DEFINITION 10.7. *Let X be a finite set. A plausibility measure Pl over 2^X is an application from 2^X to P satisfying*

- $Pl(\emptyset) = 0_P$,
- $Pl(X) = 1_P$,
- $\forall A, B \in X, A \subseteq B \Rightarrow Pl(A) \leq Pl(B)$.

Those three conditions can be simply interpreted. The last condition states a certain consistency in the plausibilities: an event is always more plausible (in a weak sense) than any event that implies it. It entails with the first condition that the impossible event is the less plausible event and with the second condition that the sure event is the most plausible event.

A plausibility measure is said to be *decomposable* if $Pl(A \cup B) = Pl(A) \oplus Pl(B)$ for any pair A, B of disjoint events and $Pl(A \cap B) = Pl(A) \otimes Pl(B)$ for any pair A, B of independent events in the sense of plausibility [HAL 01].

The restriction of a decomposable plausibility measure to singletons of 2^X is called *plausibility distribution* and completely determines the decomposable plausibility measure. In AMDPs, we assume that uncertainty is represented by plausibility distributions.

Thus we can notice that the two operators of semiring $(P, \oplus, \otimes, 0_P, 1_P)$ allow, respectively, the combination of disjoint events and the combination of independent events. Moreover notice that the hypothesis that $(P, \oplus, \otimes, 0_P, 1_P)$ is a semiring is not restrictive as the authors of [DAR 92], who used similar properties to define symbolic probabilities, showed that those properties are satisfied by many uncertainty representations, such as probability theory, possibility theory and other calculus systems used in artificial intelligence.

7. Plausibility measures should not be mistaken with the plausibility functions of Dempster and Shafer [DEM 67, SHA 76].

10.5.1.3. Generalized expected utility

We now present the formalism of *generalized expected utility* (GEU) proposed by [CHU 03] to offer a general framework for the study of decision criteria. In this framework, we assume that utilities are measured on a scale V and uncertainty concerning the consequences of an action is represented by a plausibility measure valued on a scale P . The reader interested in an axiomatic justification of this criterion when plausibility measures are assumed decomposable could refer to [WEN 06a].

As in the standard criteria (total, weighted total, mean), GEU combines plausibility and utility to define a decision criterion. In this aim, we introduce operators $\oplus^g : V \times V \rightarrow V$ and $\otimes^g : P \times V \rightarrow V$ which are the analogs of $+$ and \times over real numbers used by the standard criteria. We assume that those two operators satisfy three requirements:

$$\mathbf{GEU1} \quad (x \oplus^g y) \oplus^g z = x \oplus^g (y \oplus^g z),$$

$$\mathbf{GEU2} \quad x \oplus^g y = y \oplus^g x,$$

$$\mathbf{GEU3} \quad 1_P \otimes^g x = x.$$

Criterion GEU is then defined by

$$\text{GEU}(Pl) = \bigoplus_{x \in V}^g Pl(x) \otimes^g x,$$

where Pl is a plausibility measure.

Finally we say that Pl is preferred to Pl' if and only if $\text{GEU}(Pl) \geq_V \text{GEU}(Pl')$.

10.5.2. Definition of an algebraic MDP

An algebraic MDP (AMDP) is described as a quintuple $(\mathcal{S}, \mathcal{A}, p, r, T)$, where p and r are redefined as follows:

– $p: \mathcal{S} \times \mathcal{A} \rightarrow \mathbf{Pl}(\mathcal{S})$ is a transition function, where $\mathbf{Pl}(\mathcal{S})$ is the set of all plausibility distributions over \mathcal{S} valued in P ;

– $r: \mathcal{S} \times \mathcal{A} \rightarrow V$ is a reward function valued in V , giving the immediate reward of an action.

In a consistent manner with the Markov assumption, the following state and reward depend only on the current state and the chosen action. In particular, plausibility distributions $p(s, a)$ are independent (in a plausibilistic sense) of past states and actions.

EXAMPLE 10.5. Most MDPs previously introduced in the literature are instances of AMDPs. In standard MDPs (see section 1.2.1), uncertainty is probabilistic. Thus the underlying algebraic structure used for plausibilities is $(P, \oplus, \otimes, 0_P, 1_P) = ([0, +\infty[, +, \times, 0, 1])$. The valuation criterion of the application of a policy in a state relies on the expected utility model, which means that operators \oplus^g and \otimes^g are, respectively, $+$ and \times . When rewards are defined on $(V, \oplus_V, \otimes_V, 0_V, 1_V) = (\overline{\mathbb{R}}, \max, +, -\infty, 0)$, where $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty\}$, we can recognize the total-reward criterion (see section 1.2.3). With $(\overline{\mathbb{R}}, \max, +_\gamma, -\infty, 0)$ (where $x +_\gamma y = x + \gamma y$), we can recognize the discounted criterion. Last, with $(\overline{\mathbb{R}}, \max, +_h, -\infty, 0)$, where $a +_h b = \frac{1}{\gamma}a + b$, we can recognize the mean criterion assuming there is a final dummy state with null reward.

Possibilistic MDPs (presented in section 10.4) introduced by [SAB 98] are also AMDPs in which uncertainty is valued on a qualitative scale L endowed with the semiring structure $(L, \vee, \wedge, O_L, 1_L)$, where \vee and \wedge are, respectively, the maximum and the minimum operators on L . When optimistic utility is used (see section 10.4.1), rewards are valued on the same scale endowed with the structure $(L, \vee, *, O_L, e^*)$, where $*$ is the combination operator on rewards (\wedge for instance), with neutral element e^* .

The \oplus^g and \otimes^g operators are, respectively, \vee and \wedge . In return when pessimistic utility is used, valuation scale L has to be inverted and its values have to be minimized because of

$$U^-(\pi) = n \left(\bigvee_{x \in X} (\pi(x) \wedge n(u(x))) \right),$$

where n is an order-inversing map of L .

Qualitative MDPs , introduced by [BON 02], are AMDPs where plausibility measures are defined on the semiring of power series $(\Sigma(\epsilon), +, \times, 0, 1)$, where $\Sigma(\epsilon)$ is defined as the set of convergent infinite series in ϵ :

$$\Sigma(\epsilon) = \left\{ \sum_{k=-\infty}^{\infty} a_k \epsilon^k : \forall k, a_k \in \mathbb{R}, \sum_{k=-\infty}^{\infty} |a_k| \epsilon^k < \infty \right\}.$$

The $+$ and \times operators are the sum and the product on series. They are well defined as the series are convergent. Rewards are defined on $(\Sigma(\epsilon) \cup \{-\infty\}, \max, +, -\infty, 0)$. Finally operators for the computation of expectation are simply $+$ and \times .

Multicriteria MDPs (presented in section 10.2.2) are also instances of AMDPs. Uncertainty is probabilistic as in the standard case. We do not present here the semiring used for the rewards. The interested reader can refer to [WEN 06b] for more details.

Out of those already known instances of AMDPs, we present in section 10.5.5 other examples which have not been studied yet, to the best of our knowledge, in order to show the interest of our algebraic approach.

10.5.3. Value function of a policy

We can now proceed in a similar manner as in standard MDPs and define a value function for policies. To that end, we define the value of a history $\gamma_t = (s_t, a_t, s_{t-1}, \dots, a_1, s_0)$ by

$$r(\gamma_t) = \bigotimes_{i=1}^t r(s_i, a_i).$$

For an initial state s , a policy $\pi = (\delta_t, \dots, \delta_1)$ induces a plausibility distribution $Pl_t^\pi(s, \cdot)$ over histories. Thus the plausibility that the application of policy π in state s yields history γ_t is given by $Pl_t^\pi(s, \gamma_t)$. The value function of policy π , calculated due to this plausibility distribution using GEU is written

$$\forall s \in \mathcal{S}, \quad v_t^\pi(s) = \bigoplus_{\gamma \in \Gamma_t(s)}^g Pl_t^\pi(s, \gamma) \otimes^g r(\gamma).$$

This value function can be viewed as a vector of V^n , where n is the number of states. Policies can therefore be compared using the dominance relation \succeq_{V^n} between vectors of V^n :

$$x \succeq_{V^n} y \iff (\forall i = 1, \dots, n, x_i \geq_V y_i) \quad (10.20)$$

for any $x = (x_1, \dots, x_n), y = (y_1, \dots, y_n) \in V^n$. Indeed at a given horizon t , a policy π is preferred over a policy π' if and only if, in each state, the value of policy π is better than that of π' , which is written

$$\pi \succsim \pi' \iff \forall s \in \mathcal{S}, v_t^\pi(s) \geq_V v_t^{\pi'}(s).$$

More compactly, using dominance relation \succeq_{V^n} , we can then write

$$\pi \succsim \pi' \iff v_t^\pi \succeq_{V^n} v_t^{\pi'}.$$

A *non-dominated* policy is a policy for which there does not exist any other policy that is preferred over it.

Solving an AMDP boils down to determining non-dominated policies. The latter can be obtained by the computation of the following equations:

$$\begin{aligned} \forall s \quad v_0^*(s) &= 1_V, \\ \forall s, \forall t \quad v_t^*(s) &= \bigoplus_{\pi \in \Pi_t} v_t^\pi(s). \end{aligned} \quad (10.21)$$

10.5.4. Conditions

Equations (10.21) are hardly directly computable when the time horizon is long. For that reason, we introduce a set of conditions on the algebraic operators that guarantees those equations can be calculated iteratively:

$$\text{AMDP1 } p \otimes^g (x \oplus_{\mathcal{V}} y) = (p \otimes^g x) \oplus_{\mathcal{V}} (p \otimes^g y),$$

$$\text{AMDP2 } x \oplus^g (y \oplus_{\mathcal{V}} z) = (x \oplus^g y) \oplus_{\mathcal{V}} (x \oplus^g z),$$

$$\text{AMDP3 } p \otimes^g (q \otimes^g x) = (p \otimes q) \otimes^g x,$$

$$\text{AMDP4 } \bigoplus_i^g p_i \otimes^g (x \otimes_{\mathcal{V}} y_i) = x \otimes_{\mathcal{V}} (\bigoplus_i^g p_i \otimes^g y_i),$$

$$\text{AMDP5 } p \otimes^g (x \oplus^g y) = (p \otimes^g x) \oplus^g (p \otimes^g y)$$

for any $p, q, p_i \in P, x, y, z, y_i \in V$.

Conditions AMDP1 and AMDP2 are two distributivity properties implying a certain form of additivity of $\geq_{\mathcal{V}}$ relative to \otimes^g and \oplus^g (i.e. $x \geq_{\mathcal{V}} y \Rightarrow (z * x) \geq_{\mathcal{V}} (z * y)$ for $* \in \{\otimes^g, \oplus^g\}$). Condition AMDP3 indicates that operators \otimes and \otimes^g are “compatible”. Condition AMDP4 allows a sure gain to be isolated in the computation of the value of an action. Notice that this is similar to a distributivity axiom introduced by [LUC 03], which states that an action, represented as a probability distribution l with a finite support over consequences, is equivalent to receiving conjointly a sure gain x and another action obtained from l by subtracting x to all its consequences. Lastly, condition AMDP5 is a distributivity condition similar to that in the standard expectation.

When those conditions are satisfied, we can write an algebraic version of the Bellman equation (equation (1.1)):

$$\begin{aligned} \forall s, \quad v_0^*(s) &= 1_{\mathcal{V}}, \\ \forall s, \forall t, \quad v_t^*(s) &= \bigoplus_{a \in \mathcal{A}} r(s, a) \otimes_{\mathcal{V}} \bigoplus_{s' \in \mathcal{S}}^g p(s, a, s') \otimes^g v_{t-1}^*(s'). \end{aligned} \tag{10.22}$$

The authors of [PER 05] have proven the following proposition, which indicates that equations (10.21) and (10.22) are equivalent.

PROPOSITION 10.1 [PER 05]. *If conditions AMDP1 to AMDP5 are satisfied, then the policies obtained through the Bellman equations (10.22) are non-dominated policies.*

Proposition 10.1 then justifies the use of an algebraic version of the backward induction algorithm (Algorithm 10.3) and factorizes in one unique result different works on standard MDPs, multicriteria MDPs, possibilistic MDPs, etc. Due to its generality, this result allows a certain number of known works developed in different contexts to be explained, but it also allows the backward induction algorithm to be justified in advance in contexts that have not been studied yet. We present in section 10.5.5 some original and potentially useful instances of AMDPs.

Algorithm 10.3: Algebraic version of the backward induction for AMDPs

```

 $v_0 \leftarrow 1$ 
 $t \leftarrow 0$ 
repeat
   $t \leftarrow t + 1$ 
  for  $i = 1 \dots n$  do
    for  $j = 1 \dots m$  do
       $q_t(s^i, a^j) \leftarrow r(s^i, a^j) \otimes_{\mathcal{V}} \bigoplus_{k=1 \dots n}^g p(s^i, a^j, s^k) \otimes^g v_{t-1}(s^k)$ 
     $v_t(s^i) \leftarrow q_t(s^i, a^1) \oplus_{\mathcal{V}} \dots \oplus_{\mathcal{V}} q_t(s^i, a^m)$ 
until  $t = h$ 

```

10.5.5. Examples of AMDPs

To show the generality of the algebraic approach, we provide a few examples of AMDPs that, to the best of our knowledge, have not been studied yet: a first example with a partial preference structure, a second example with a qualitative uncertainty representation and a last example where the rewards are non-decreasing functions over a valuation semiring.

10.5.5.1. Probabilistic multicriteria AMDP

In probabilistic multicriteria MDPs, Pareto-dominance is used to discriminate between reward vectors. The order induced by Pareto-dominance is partial. Therefore it would be possible that it is not discriminating enough and that too many non-dominated policies are obtained. We propose to refine the order induced by Pareto-dominance by introducing a priority in the comparison between criteria.

Let Q be the set of criteria ($|Q|$ criteria in total) and \succeq_Q a (possibly partial) order relation over Q (reflecting the importance of the criteria). Following [GRO 91] and [JUN 02], we use a strict order relation \succ_G between vectors, which is characterized

by, for any $x = (x_1, \dots, x_{|Q|})$ and $y = (y_1, \dots, y_{|Q|})$ in $\mathbb{R}^{|Q|}$:

$$x \succ_G y \iff \begin{cases} \exists i = 1 \dots |Q| \quad x_i \neq y_i, \\ \forall i : x_i \neq y_i \quad ((x_i > y_i) \text{ or } (\exists j \succ_Q i, x_j > y_j)). \end{cases}$$

In a natural manner, we define \succeq_G , for any x, y in $\mathbb{R}^{|Q|}$, by

$$x \succeq_G y \iff x = y \text{ or } x \succ_G y.$$

The Pareto-dominance relation is a particular case of \succeq_G when \succeq_Q is the empty relation, i.e. each criterion has the same importance. When \succeq_Q is linear, \succeq_G becomes the lexicographic order relation. Preference relation \succeq_G thus allows the Pareto-dominance relation to be refined by introducing a priority when taking into account the criteria.

[PER 05] showed that this example is an instance of AMDPS and that it is possible to calculate non-dominated policies by backward induction.

10.5.5.2. Possibilistic multicriteria AMDPS

Under possibilistic uncertainty, the use of optimistic and pessimistic utilities (section 10.4.1) has been extended to sequential decision-making in the possibilistic MDP framework (section 10.4) by [SAB 98]. We show that the use of binary possibilistic utility [GIA 01], which is a unification of the two previous utilities, is also conceivable.

Before recalling the definition of binary possibilistic utility, let us present the framework. Uncertainty is measured on a totally ordered scale $(P, \wedge, \vee, 0_P, 1_P)$, where \wedge and \vee are, respectively, the minimum and maximum operators on P . We define $(\hat{P}_2 = \{\langle \lambda, \mu \rangle : \lambda, \mu \in P\}, \oplus_{\hat{P}_2}, \otimes_{\hat{P}_2}, \langle 0_P, 1_P \rangle, \langle 1_P, 0_P \rangle)$ where for any $\langle \alpha, \beta \rangle, \langle \lambda, \mu \rangle$ in \hat{P}_2 , we have

$$\langle \alpha, \beta \rangle \oplus_{\hat{P}_2} \langle \lambda, \mu \rangle = \langle \alpha \vee \lambda, \beta \wedge \mu \rangle$$

and

$$\langle \alpha, \beta \rangle \otimes_{\hat{P}_2} \langle \lambda, \mu \rangle = \langle \alpha \wedge \lambda, \beta \vee \mu \rangle.$$

These two structures are semirings thanks to the properties of \vee and \wedge (in particular their distributivities over each other). Operator $\oplus_{\hat{P}_2}$ induces a partial order $\geq_{\hat{P}_2}$ over \hat{P}_2 :

$$\forall \langle \lambda, \mu \rangle, \langle \lambda', \mu' \rangle \in \hat{P}_2, \quad (\langle \lambda, \mu \rangle \geq_{\hat{P}_2} \langle \lambda', \mu' \rangle) \iff (\lambda \geq \lambda' \text{ and } \mu \leq \mu'). \quad (10.23)$$

Rewards are measured over $P_2 = \{\langle \lambda, \mu \rangle \in \hat{P}_2 : \lambda \vee \mu = 1_P\}$, which is a subset of \hat{P}_2 . Notice that relation $\geq_{\hat{P}_2}$ is total when it is restricted to P_2 and operators $\oplus_{\hat{P}_2}$ and $\otimes_{\hat{P}_2}$ are, respectively, the maximum and minimum operators on P_2 .

We now recall the definition of binary possibilistic utility:

$$PU(\pi) = \bigvee_{x \in X} (\pi(x) \wedge u(x)) = \left\langle \bigvee_{x \in X} (\pi(x) \wedge u_1(x)), \bigvee_{x \in X} (\pi(x) \wedge u_2(x)) \right\rangle,$$

where $u: X \rightarrow P_2$ is a utility function valued in P_2 and $\forall x \in X, u(x) = \langle u_1(x), u_2(x) \rangle$. Thus it is a generalized expectation with operator \oplus^g defined as the component-wise operator \vee and operator \otimes^g as the component-wise operator \wedge . Notice that this criterion also takes its values in P_2 .

Thanks to the properties of \vee and \wedge , if policies are valued with this binary criterion, then the algebraic version of backward induction yields the optimal policies.

As in the previous example, we now assume that actions and therefore policies are valued by a vector (of elements of P_2). Moreover, over the criteria set Q , we assume that relation \succeq_Q is defined. Then strict order \succ_G is now characterized by, for any $x = (x_1, \dots, x_{|Q|})$ and any $y = (y_1, \dots, y_{|Q|})$ in $P_2^{|Q|}$:

$$x \succ_G y \Leftrightarrow \begin{cases} \exists i = 1 \dots |Q|, \quad x_i \neq y_i, \\ \forall i : x_i \neq y_i, \quad ((x_i >_{\hat{P}_2} y_i) \text{ or } (\exists j \succ_Q i, x_j >_{\hat{P}_2} y_j)). \end{cases}$$

In the same manner as in the previous example, [PER 05] showed that the backward induction Algorithm 10.3 allows the computation of non-dominated policies.

10.5.5.3. AMDPs whose rewards are non-decreasing functions

Let V be a valuation scale endowed with a semiring structure $(V, \oplus_V, \otimes_V, 0_V, 1_V)$. Idempotent operator \oplus_V is used to maximize and operator \otimes_V is used to combine the values of that scale. The set H of non-decreasing functions (in the sense of \geq_V) over V can be endowed with the following semiring structure $(H, \oplus_V, *, \mathbf{0}, Id)$ [MIN 77] where:

- $\forall x \in V, \forall f, g \in H, (f \oplus_V g)(x) = f(x) \oplus_V g(x);$
- $\forall f, g \in H, f * g = g \circ f$ (composition of functions);
- $\mathbf{0}$ is the constant function equals to 0_V everywhere;
- Id is the identity function.

It is then possible to build an AMDP whose rewards would be non-decreasing functions, i.e. the reward function would be defined by

$$- r : \mathcal{S} \times \mathcal{A} \rightarrow H.$$

In such an AMDP, rewards could then vary. The value of a history $\gamma_t = (s_t, a_t, s_{t-1}, \dots, a_1, s_0)$ is then defined by

$$r(\gamma_t) = r(s_t, a_t)(r(\gamma_{t-1})),$$

where $\gamma_{t-1} = (s_{t-1}, a_{t-1}, \dots, a_1, s_0)$ and $r(\gamma_0)$ is a fixed value depending on the problem to be solved.

Operators \oplus^g and \otimes^g are extended to H by

$$\forall x \in V, \quad (h \oplus^g g)(x) = h(x) \oplus^g g(x) \text{ and } (p \otimes^g h)(x) = p \otimes^g h(x)$$

for any $h, g \in H$ and any $p \in P$. The following proposition indicates that conditions AMDP1 to AMDP5 hold.

PROPOSITION 10.2 [WEN 06b]. *If conditions AMDP1 to AMDP5 are satisfied on structure V , then these conditions are also satisfied on structure H .*

As conditions AMDP1 to AMDP5 hold, our results on AMDP could then be applied. In particular, the algorithm generalized from backward induction (Algorithm 10.3) can be exploited for the search of non-dominated policies.

We now present an illustrative example where it is natural to model rewards as non-decreasing functions.

EXAMPLE 10.6. The illustrative example that we develop in this section is inspired by the problem of transportation of hazardous products presented in [ERK 98] and [SER 06]. Assume that a robot has to transport a hazardous product from point A to a goal. The states of this problem are the different positions where the robot can be. Assume the environment is represented by an $n \times m$ grid (see the left-hand side of Figure 10.10). Then $\mathcal{S} = \{1, 2, \dots, n\} \times \{1, 2, \dots, m\}$. Certain states (walls, obstacles, etc.) cannot be accessed. This information is integrated in the transition function. The actions correspond to the movements that the robot can perform. For instance, the actions could be up, down, left or right. The effects of the actions are modeled by a probabilistic transition function (see the right-hand side of Figure 10.10). The success of an action is not certain for diverse reasons: the robot imperfectly controls its engine, the ground is slippery or for any other external event). Rewards here are costs. They should then be minimized and they model the probability of an accident and its incurred cost. At each movement, the robot risks having an accident incurring a cost c with a probability p . Costs and probabilities

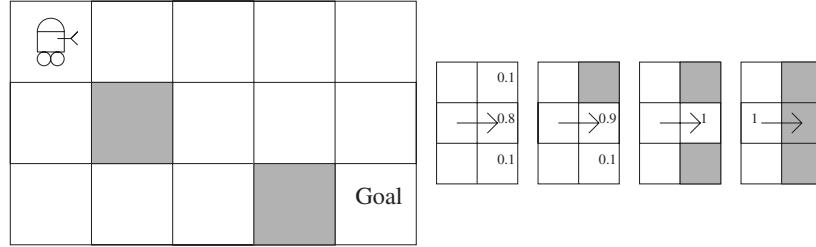


Figure 10.10. Navigation of an autonomous agent

could depend on the position and the performed action (certain spots are more difficult to access than others for instance). They are then denoted by $c(s, a)$ and $p(s, a)$ for any $s \in \mathcal{S}$ and $a \in \mathcal{A}$.

The standard approach in the treatment of this problem would be to look for the policy that minimizes the expectation of the sum of the costs without taking into account the probability of having an accident. The criterion would be at horizon h :

$$\forall s \in \mathcal{S}, \quad v_h^\pi(s) = E_s^\pi \left(\sum_{t=1}^h C_t \right),$$

where E_s^π is the expectation induced by the application of policy π in state s and C_t is the cost incurred at step t . This approach is questionable since the cost of an accident is taken into account even if it has not occurred.

Another approach would be to look for the policy maximizing the probability of not having an accident. The criterion considered at horizon h would be

$$\forall s \in \mathcal{S}, \quad v_h^\pi(s) = E_s^\pi \left(\prod_{t=1}^h (1 - P_t) \right),$$

where P_t is the probability of having an accident at step t . In this approach, the cost of an accident is not taken into account. We would want to make compromises between the cost and the probability of an accident.

In this manner, when both pieces of information (costs and probabilities of an accident) are available, a better approach could be implemented by defining the *risk* of a history.

DEFINITION 10.8. *The risk of a history $\gamma_t = (s_t, a_t, s_{t-1}, a_{t-1}, \dots, a_1, s_0)$ is defined recursively by*

$$r(\gamma_t) = p_t c_t + (1 - p_t) r(\gamma_{t-1}),$$

where $c_t = c(s_t, a_t)$, $p_t = p(s_t, a_t)$ and $\gamma_{t-1} = (s_{t-1}, a_{t-1}, \dots, a_1, s_0)$ by setting the risk of an empty history to 0.

The value function of policies is then defined at horizon h by

$$\forall s \in \mathcal{S}, \quad v_h^\pi(s) = \sum_{\gamma \in \Gamma_h(s)} r(\gamma) Pr_s^\pi(\gamma),$$

where Pr_s^π is the probability distribution over histories induced by the application of policy π in state s .

The problem of searching a policy minimizing the expectation of risk can be modeled with an AMDP whose reward function is defined as follows:

$$-r : \mathcal{S} \times \mathcal{A} \rightarrow H.$$

H is the set of non-decreasing functions defined on the semiring $(\mathbb{R} \cup \{+\infty\}, \min, +\infty)$.

For our problem of hazardous product transportation, we can define for any $s \in \mathcal{S}$ and for any $a \in \mathcal{A}$, $r(s, a)(x) = p(s, a)c(s, a) + (1 - p(s, a))x$. We can notice that the risk of a history $\gamma_t = (s_t, a_t, s_{t-1}, a_{t-1}, \dots, a_1, s_0)$ is then defined by

$$r(\gamma_t) = r(s_t, a_t)(r(\gamma_{t-1})), \text{ where } \gamma_{t-1} = (s_{t-1}, a_{t-1}, \dots, a_1, s_0).$$

The risk of a history is obtained by decomposing successively the rewards (which are functions). In this section, we have shown that the backward induction algorithm could be used to look for policies that minimize the expected risk.

10.6. Conclusion

This chapter has only given a brief, non-exhaustive and therefore biased overview of existing works about taking into account non-standard criteria in the MDP framework. However, it has showed that this research direction is vast (from the integration of multiple criteria to the use of non-standard decision criteria) and active.

Let us cite for the record [ALT 99], which proposed to take into account several criteria in the framework of *constrained* MDPs. A constrained MDP contains several cost functions,⁸ c_0, c_1, \dots, c_k . A set of cost thresholds C_1, \dots, C_k and a set of probability degrees $\varepsilon_1, \dots, \varepsilon_k$ are also defined. Solving a constrained MDP consists of

8. A constrained MDP could be equivalently defined with reward functions but this framework has been defined by an automatician and keeps the “minimization” formulation of costs, usually adopted in that domain.

finding, for a fixed initial state s_0 , a stationary non-deterministic policy minimizing the value function of the MDP defined with c_0 as a cost function, under the constraint that the probability of each value functions defined with $c_i, i \geq 1$, being greater than C_i does not exceed ε_i , i.e.

$$\begin{aligned} \min_{\pi} \quad & V_0^{\pi}(s_0) \\ \text{s.t.} \quad & \forall i \geq 1, \Pr(V_i^{\pi}(s_0) < C_i) < \varepsilon_i, \end{aligned}$$

where V_i^{π} is the value function of policy π defined with c_i .

The qualitative possibilistic utilities are not the only criteria of non-standard utilities to have been extended to sequential decision-making. More generally a certain number of works [SAR 98, JAF 06] aim at exploiting in the sequential decision-making framework richer models in terms of descriptive power. These decision models, such as the RDU criterion [QUI 93], the Choquet integral [CHO 53] or the Sugeno integral [SUG 74] developed in decision theory cannot be expressed under the form of a generalized expectation and cannot be used in the AMDP framework. Indeed they are known for not being dynamically consistent [MAC 89], that is the preferences at one point in time could be different viewed from another point in time. This implies that subpolicies of optimal policies could be non-optimal and that optimal subpolicies could induce non-optimal policies, and forbids the use of dynamic programming resolution methods. Thus, the use of such decision models in sequential decision-making represents a hard problem as the Bellman optimality principle does not hold.

10.7. Bibliography

- [ALT 99] ALTMAN E., *Constrained Markov Decision Processes*, Chapman & Hall/CRC, Boca Raton, FL, 1999.
- [BAG 01] BAGNELL J. A., NG A. Y. and SCHNEIDER J., Solving uncertain Markov decision problems, Report no. CMU-RI-TR-01-25, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, August 2001.
- [BON 02] BONET B. and PEARL J., “Qualitative MDPs and POMDPs: an order-of-magnitude approximation”, *Proceedings of the 18th Annual Conference on Uncertainty in Artificial Intelligence (UAI'02)*, Morgan Kaufmann, San Francisco, CA, pp. 61–68, 2002.
- [BOU 94] BOUTILIER C., “Toward a logic for qualitative decision theory”, *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning (KR'94)*, pp. 75–86, 1994.
- [BOU 07] BOUSSARD M., BOUZID M. and MOUADDIB A.-I., “Multi-criteria decision making for local coordination in multi-agent systems”, *Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'07)*, pp. 87–90, 2007.

- [BRI 04] BRIGUI I., BELLOSTA M., KORNMAN S., PINSON S. and VANDERPOOTEN D., “Un mécanisme de négociation multicritère pour le commerce électronique”, *Reconnaissance des Formes et Intelligence Artificielle (RFIA'04)*, pp. 1009–1016, 2004.
- [BUF 05] BUFFET O. and ABERDEEN D., “A two-teams approach for robust probabilistic temporal planning”, *Proceedings of the ECML'05 Workshop on Reinforcement Learning in Non-Stationary Environments*, Porto, Portugal, 2005.
- [CHO 53] CHOQUET G., “Theory of capacities”, *Annales de l'institut Fourier*, vol. 5, pp. 131–295, 1953.
- [CHU 03] CHU F. C. and HALPERN J. Y., “Great expectations. Part I: On the customizability of generalized expected utility”, *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*, Acapulco, Mexico, pp. 291–296, 2003.
- [DAE 80] DAELLENBACH H. G. and DE KLUYVER C. A., “Note on multiple objective dynamic programming”, *Journal of the Operational Research Society*, vol. 31, no. 7, pp. 591–594, 1980.
- [DAR 92] DARWICHE A. and GINSBERG M. L., “A symbolic generalization of probability theory”, *Proceedings of the National Conference on Artificial Intelligence (AAAI'92)*, pp. 622–627, 1992.
- [DAR 94] DARWICHE A. and GOLDSZMIDT M., “On the relation between kappa calculus and probabilistic reasoning”, *Proceedings of the 10th Conference on Uncertainty on Artificial International (UAI'94)*, Morgan Kaufman, San Francisco, CA, pp. 145–153, 1994.
- [DEM 67] DEMPSTER A. P., “Upper and lower probabilities induced by a multivalued mapping”, *Annals of Mathematical Statistics*, vol. 38, pp. 325–339, 1967.
- [DUB 88] DUBOIS D. and PRADE H., *Possibility Theory*, Plenum Press, New York, 1988.
- [DUB 93] DUBOIS D., PRADE H. and SANDRI S., “Chapter on possibility/probability transformations”, in LOWEN R. and ROUBENS M., Eds., *Fuzzy Logic: State of the Art*, Kluwer Academic Publishers, Boston, MA, pp. 103–112, 1993,
- [DUB 94] DUBOIS D. and PRADE H., “A survey of belief revision and updating rules in various uncertainty models”, *International Journal of Intelligent Systems*, vol. 9, pp. 61–100, 1994.
- [DUB 95] DUBOIS D. and PRADE H., “Possibility theory as a basis for qualitative decision theory”, *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI'95)*, Montreal, Canada, pp. 1925–1930, 1995.
- [DUB 96] DUBOIS D., FARGIER H. and PRADE H., “Possibility theory in constraint satisfaction problems”, *Applied Intelligence*, vol. 6, no. 4, pp. 287–309, 1996.
- [DUB 98] DUBOIS D., PRADE H. and SABBADIN R., “Qualitative decision theory with Sugeno integrals”, COOPER G. F. and MORAL S., Eds., *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI'98)*, Morgan Kaufmann, San Francisco, CA, pp. 121–128, 1998.
- [EHR 03] EHRCOTT M. and TENFELDE-PODEHL D., “Computation of ideal and Nadir values and implications for their use in MCDM methods”, *European Journal of Operational Research*, vol. 151, no. 1, pp. 119–139, 2003.

- [ERK 98] ERKUT E. and VERTER V., "Modeling of transport risk for hazardous materials", *Operations Research*, vol. 48, pp. 624–642, 1998.
- [FAR 98] FARGIER H., LANG J. and SABBADIN R., "Towards qualitative approaches to multi-stage decision making", *International Journal of Approximate Reasoning*, vol. 19, pp. 441–471, 1998.
- [FAR 03] FARGIER H. and SABBADIN R., "Qualitative decision under uncertainty: back to expected utility", *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*, Acapulco, Mexico, pp. 303–308, 2003.
- [FAR 05] FARGIER H. and SABBADIN R., "Qualitative decision under uncertainty: back to expected utility", *Artificial Intelligence*, vol. 164, pp. 245–280, 2005.
- [FRI 95] FRIEDMAN N. and HALPERN J. Y., "Plausibility measures: A user's guide", *Proceedings of the 11th International Conference on Uncertainty in Artificial Intelligence (UAI'95)*, Montreal, Canada, pp. 175–184, 1995.
- [FUR 80] FURUKAWA N., "Characterization of optimal policies in vector-valued Markovian decision processes", *Mathematics of Operations Research*, vol. 5, no. 2, pp. 271–279, 1980.
- [GAL 06] GALAND L. and PERNY P., "Search for compromise solutions in multiobjective state space graphs", *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI'06)*, Riva del Garda, Italy, pp. 93–97, 2006.
- [GAR 06] GARCIA L. and SABBADIN R., "Possibilistic influence diagrams", *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI'06)*, Riva del Garda, Italy, pp. 372–376, 2006.
- [GAR 07] GARCIA L. and SABBADIN R., "Diagrammes d'influence possibilistes", *Revue d'Intelligence Artificielle*, vol. 21, no. 4, pp. 521–554, 2007.
- [GAR 08] GARCIA L. and SABBADIN R., "Complexity results and algorithms for possibilistic influence diagrams", *Artificial Intelligence*, vol. 172, no. 8-9, pp. 1018–1044, 2008.
- [GIA 99] GIANG P. H. and SHENOY P. P., "On transformations between probability and Spohnian disbelief functions", PRADE H., Ed., *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI'99)*, Morgan Kaufmann, San Mateo, CA, pp. 236–244, 1999.
- [GIA 01] GIANG P. H. and SHENOY P. P., "A comparison of axiomatic approaches to qualitative decision making using possibility theory", *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence (UAI'01)*, pp. 162–170, 2001.
- [GIV 00] GIVAN R., LEACH S. and DEAN T., "Bounded-parameter Markov decision processes", *Artificial Intelligence*, vol. 122, no. 1-2, pp. 71–109, 2000.
- [GON 01] GONDTRAN M. and MINOUX M., *Graphes, dioides et semi-anneaux*, Editions Technique et Documentation, 2001.
- [GRA 02] GRABISCH M. and PERNY P., "Agrégation multicritère", *Logique Floue, Principes, Aide à la Décision*, Hermès, Paris, pp. 81–120, 2002.
- [GRO 91] GROSOF B., "Generalizing prioritization", *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pp. 289–300, 1991.

- [HAL 01] HALPERN J. Y., “Conditional plausibility measures and Bayesian networks”, *Journal of Artificial Intelligence Research*, vol. 14, pp. 359–389, 2001.
- [HAN 80] HANSEN P., “Bicriterion path problems”, FANDEL G. and GAL T., Eds., *Multiple Criteria Decision Making: Theory and Applications*, vol. 177 of *Lecture Notes in Econom. and Math. Systems*, Springer-Verlag, Berlin, pp. 109–127, 1980.
- [HEN 99] HENRION M., PROVAN G., DEL FAVEROL B. and SANDERS G., “An experimental comparison of numerical and qualitative probabilistic reasoning”, *Proceedings of the 10th Annual Conference on Uncertainty in Artificial Intelligence (UAI'94)*, Morgan Kaufmann, San Mateo, CA, pp. 319–326, 1999.
- [JAF 06] JAFFRAY J. Y. and NIELSEN T. D., “Dynamic decision making without expected utility: an operational approach”, *European Journal of Operational Research*, vol. 169, pp. 226–246, 2006.
- [JUN 02] JUNKER U., “Preference-based search and multi-criteria optimization”, *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI'02)*, Edmonton, Alberta, Canada, pp. 34–40, 2002.
- [KEE 76] KEENEY R. L. and RAIFFA H., *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*, John Wiley & Sons, New York, 1976.
- [LUC 03] LUCE R. D., “Increasing increment generalizations of rank-dependent theories”, *Theory and Decision*, vol. 55, no. 2, pp. 87–146, 2003.
- [MAC 89] MACHINA M. J., “Dynamic consistency and non-expected utility models of choice under uncertainty”, *Journal of Economic Literature*, vol. 27, no. 4, pp. 1622–1668, 1989.
- [MIN 77] MINOUX M., “Generalized path algebra”, *Surveys of Mathematical Programming*, Publishing House of the Hungarian Academy of Sciences, pp. 359–364, 1977.
- [MOU 04] MOUADDIB A. I., “Multi-objective decision-theoretic path planning”, *Proceedings of the IEEE International Conference on Robotics and Automaton (ICRA'04)*, pp. 2814–2819, 2004.
- [MUN 02] MUNOS R. and MOORE A., “Variable resolution discretization in optimal control”, *Machine Learning*, vol. 49, pp. 291–323, 2002.
- [NIL 04] NILIM A. and EL GHAOUI L., “Robustness in Markov decision problems with uncertain transition matrices”, *Advances in Neural Information Processing Systems 16 (NIPS'03)*, 2003.
- [NIL 05] NILIM A. and EL GHAOUI L., “Robust solutions to Markov decision problems with uncertain transition matrices”, *Operation Research*, vol. 53, no. 5, 2005.
- [PER 05] PERNY P., SPANJAARD O. and WENG P., “Algebraic Markov decision processes”, *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI'05)*, Edinburgh, Scotland, pp. 1372–1377, 2005.
- [PRA 06] PRALET C., VERFAILLIE G. and SCHIEX T., “Decision with uncertainties, feasibilities and utilities: Towards a unified algebraic framework”, *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI'06)*, Riva del Garda, Italy, pp. 427–431, 2006.

- [QUI 93] QUIGGIN J., *Generalized Expected Utility Theory: The Rank-Dependent Model*, Kluwer Academic Publishers, Boston, MA, 1993.
- [SAB 98] SABBADIN R., Une approche ordinaire de la décision dans l'incertain: axiomatisation, représentation logique et application à la décision séquentielle, PhD thesis, université Paul Sabatier de Toulouse, 1998.
- [SAB 99] SABBADIN R., "A possibilistic model for qualitative sequential decision problems under uncertainty in partially observable environments", LASKEY K. and PRADE H., Eds., *Proceedings of the 15th Conference Uncertainty in Artificial Intelligence (UAI'99)*, Morgan Kaufmann, San Mateo, CA, pp. 567–574, 1999.
- [SAB 01a] SABBADIN R., "Possibilistic Markov decision processes", *Engineering Applications of Artificial Intelligence*, vol. 14, pp. 287–300, 2001.
- [SAB 01b] SABBADIN R., "Towards possibilistic reinforcement learning algorithms", *Proceedings of the 10th IEEE International Conference on Fuzzy Systems (FUZZ-IEEE'01)*, vol. 1, pp. 404–407, 2001.
- [SAR 98] SARIN R. and WAKKER P., "Dynamic choice and non-expected utility", *Journal of Risk and Uncertainty*, vol. 17, pp. 87–119, 1998.
- [SAV 54] SAVAGE L. J., *The Foundations of Statistics*, John Wiley & Sons, New York, 1954.
- [SER 06] SERAFINI P., "Dynamic programming and minimum risk paths", *European Journal of Operational Research*, vol. 175, pp. 224–237, 2006.
- [SHA 76] SHAFER G., *A Mathematical Theory of Evidence*, Princeton University Press, Princeton, NJ, 1976.
- [SUG 74] SUGENO M., Theory of fuzzy integrals and its applications, PhD thesis, Tokyo Institute of Technology, 1974.
- [TRE 07] TREVIZAN F. W., COZMAN F. G. and DE BARROS L. N., "Planning under risk and Knightian uncertainty", *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, Hyderabad, India, pp. 2023–2028, 2007.
- [VIN 89] VINCKE P., *L'aide multicritère à la décision*, Statistiques et mathématiques appliquées, édition de l'université de Bruxelles, éditions Ellipses, 1989.
- [WAK 01] WAKUTA K., "A multi-objective shortest path problem", *Mathematical Methods of Operations Research*, vol. 54, no. 3, pp. 445–454, 2001.
- [WEN 06a] WENG P., "Axiomatic foundations for a class of generalized expected utility: algebraic expected utility", *Proceedings of the 22nd Conference on Uncertainty in Artificial Intelligence (UAI'06)*, Cambridge, MA, pp. 520–527, 2006.
- [WEN 06b] WENG P., Modèles qualitatifs et approches algébriques pour la décision dans l'incertain : fondements axiomatiques et application à la décision séquentielle, PhD thesis, université Paris VI, 2006.
- [WHI 82] WHITE D. J., "Multi-objective infinite-horizon discounted Markov decision processes", *Journal of Mathematical Analysis and Applications*, vol. 89, pp. 639–647, 1982.

Part 3
Applications

Chapter 11

Online Learning for Micro-Object Manipulation

11.1. Introduction

This chapter presents the application of a reinforcement learning algorithm to learning online how to manipulate micro-objects. What makes this application original is that the action policy has been learned not thanks to a simulator but by controlling the real process. This work related to reinforcement learning for micro-robotics has been conducted at the *Automatic control and Micro-Mechatronic Systems* department of the FEMTO-ST Institute (Besançon, France). It was first described in [ADD 05].

Micro-robotics has the general objective of designing, realizing and controlling compact robotic systems used to manipulate objects whose dimensions typically range between one micrometer and one millimeter for various applications (instrumentation, micro-assembly, biomedical applications).

Considering the dimensions and the required precision, micro-robotics faces practical difficulties which are quite different from classical robotics:

- at the actuators level: micro-robotics employs new, more compact, actuation principles, for example, based on the use of active materials; such actuators are often strongly non-linear;

Chapter written by Guillaume LAURENT.

- at the sensors level: the reduced volume of the applications makes it difficult to set up a sufficient number of sensors; employing a vision system (via a microscope) is often the main mean to observe and measure;
- at the level of the interactions between the robot and the surrounding objects: at this scale, surface forces become preponderant over volume forces; the inertia of objects is very weak; the friction between objects generates dry friction forces which are important and difficult to quantify; under 100 µm adhesion (capillarity) and van der Waals forces make the objects “stick” to each other; these phenomena make manipulations extremely delicate and hazardous.

The actuators’ non-linearity, the sensors’ imprecision, the complexity of surface forces, make the processes hard to model. Because of the lack of any precise model, the synthesis of controllers through traditional approaches from the control field is difficult. On the contrary, reinforcement learning frees us from using any model of the controlled process and allows us to take into account the uncertainty in this process via a stochastic approach. These control methods are therefore adapted to micro-robotics.

This chapter is structured in three sections. The first section presents the context of micro-manipulation by pushing as well as the device to control. The second section describes the reinforcement learning algorithm employed for controlling the manipulator. Experimental results are presented in the last section.

11.2. Manipulation device

11.2.1. *Micro-positioning by pushing*

In the industry, positioning is an essential function for machining or assembling parts. In the field of micro-robotics, classical solutions as “pick-and-place” are not directly transposable. In these conditions, it is often easier to push a micro-object than to hold it in a gripper. Thus, numerous works use this approach for micro-manipulation [CHE 07, GOR 06, GAU 06, SIT 04, MOL 02, ZES 98] and also for nano-manipulation [KOR 09, BAU 98, HAN 98, RES 00].

If these manipulators are simpler to design, things are different for their control. Indeed, the problem of predicting the movement of an object pushed at a given point is already complex at the macroscopic scale. Under a millimeter, classical friction equations based on the objects’ weights (Coulomb’s law) do not apply, due to the important preponderance of surface forces. The movement of an object being pushed depends on various parameters such as the surface roughness of the stand and of the object, air humidity, the distribution of electrostatic charges, etc. In these conditions, the movement of the object is difficult to predict, which led us to consider a reinforcement learning approach for control. The objective therefore consists of synthesizing – by learning – high-performance control policies for a manipulator performing micro-positioning tasks by pushing.

11.2.2. Manipulation device

The controlled manipulation device is inspired from existing manipulation devices that allow for the pushing and pulling of micro-parts one-by-one thanks to an atomic force microscope's cantilever [GAU 06, SIT 04, ZES 98, HAN 98]. On the contrary, the scale of the device we describe is very different since this is about positioning millimeter-sized objects like watch gears. This device is a testbed whose objective is to demonstrate the feasibility of control by reinforcement learning for micro-positioning by pushing.

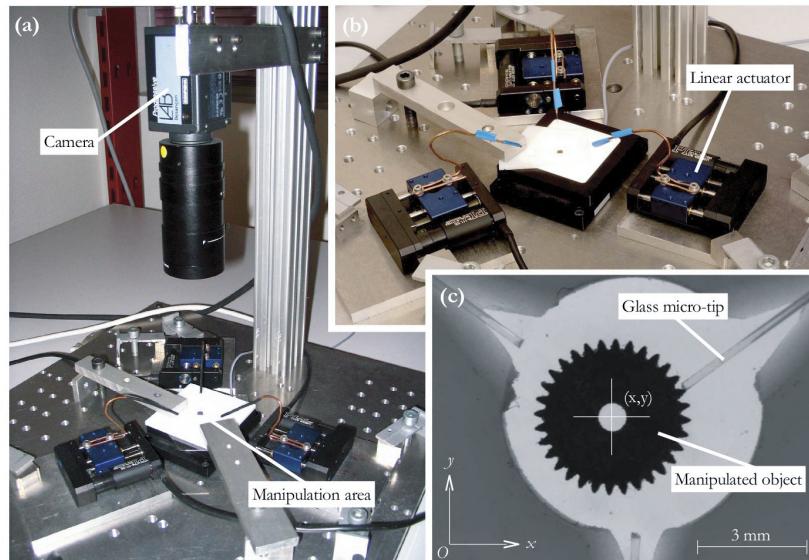


Figure 11.1. Manipulation device: (a) overview, (b) zoom on the manipulation area surrounded by three linear actuators, (c) detail of the manipulation area (camera image).
The object is localized with the vision system

The manipulator is equipped with three glass micro-tips mounted on linear actuators whose position is regulated with a micro-metric precision (see Figure 11.1b). The object to manipulate is laid on a glass slide between three micro-tips (see Figure 11.1c). Each tip can come in contact with the object then exert a thrust over a short distance. The objective is to move this object towards a given position through an adequate sequence of pushes.

The micro-tips' displacement axes are concurrent. The manipulation area is surrounded by a circular wall preventing the manipulated object from escaping. This disposition enables the movement of a (circular) cylindrical object to any position.

The object position is measured via a vision system (see Figure 11.1a). The object is located through its x and y axes in the video image. The camera's resolution being limited, the precision of the localization is of about 23 μm .

11.2.3. Control loop

To control the manipulator, two control levels are used: a low level and a high level (see Figure 11.2). The low level handles the position regulation of the micro-tips (with traditional control methods). This allows for bringing a micro-tip in contact with the object, then pushing it over a distance specified by the high level controller. The high level is the decision stage that plans pushes over the long term to bring the object to a given position.

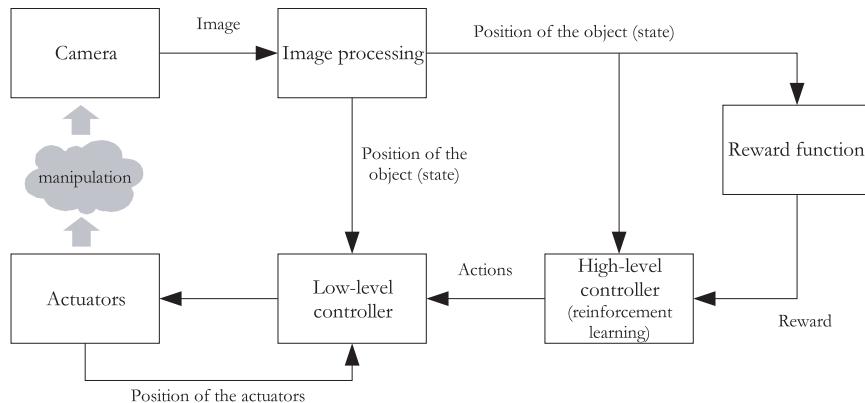


Figure 11.2. Sensory-motor loop

A high level action's duration (i.e. a push) is variable: from one to three seconds. At the end of a push, the object immediately stops because of its very weak inertia. Thus, only the static behavior of the object is taken into account. Its dynamic behavior is neglected.

11.2.4. Representation of the manipulation task as an MDP

11.2.4.1. Definition of the state space

The system being considered as static, its state s is simply defined by the position (x, y) of the object's center in the video image.

The manipulation area having a diameter of 7 mm and the object a diameter of 4 mm, the position of the object's center can evolve within a disc with a diameter of

3 mm. Given the low resolution of the video sensor, the camera allows for localizing the center of the object in a circular area of diameter 131 pixels, i.e. about 13,500 different positions ($\text{card } \mathcal{S} \approx 13,500$).

11.2.4.2. *Definition of the action space*

In the conducted experiments, only 6 distinct actions have been used. Each of the three micro-tips can push the object over two determined distances: a “long” push of 1 mm – used for large displacements – or a “short” push of 100 μm – essential for fine positioning. We therefore have $\text{card } \mathcal{A} = 6$.

This set of 6 actions is the result of a compromise between the quality of the trajectories obtained and the learning time. More variety in these actions would probably allow for a faster manipulation but would considerably increase the size of the search space and therefore the overall learning time.

11.2.4.3. *Definition of the reward function*

The objective of the manipulation is to bring an object to a given position with a given precision. In the conducted experiments, the goal was to position the object at the center of the manipulation area with a 140 μm precision (6 camera pixels). The set of states located in this goal area is denoted by $\mathcal{S}_{\text{goal}}$.

Thus, the goal is to lead the process from any state to one of the states in $\mathcal{S}_{\text{goal}}$. The reward function is therefore defined as follows:

$$r(s, a, s') = \begin{cases} 1 & \text{if } s' \in \mathcal{S}_{\text{goal}}, \\ 0 & \text{otherwise.} \end{cases} \quad (11.1)$$

11.2.4.4. *Definition of an episode*

Before each manipulation, the object is put in a random position in the manipulation area (i.e. the initial state is random). The manipulator is then controlled by the high level control algorithm, in this case a reinforcement learning algorithm, and an episodic task starts. The episode ends when the object has reached a goal state.

11.3. Choice of the reinforcement learning algorithm

11.3.1. *Characteristics of the MDP*

The manipulation system being particularly slow (one push every one to three seconds), it is imperative to reduce to a minimum the number of episodes necessary to obtain a good policy. Furthermore, it is possible to perform many offline computations between two pushes.

The vision-based localization system provides a discrete observation of the system state. Considering the low resolution of the video sensor, there is a notable imprecision on the measure of the process state: it is not fully observable. Yet, the observation is sufficient to consider that the process is discrete, fully observable and slightly stochastic (quantization noise).

For these reasons, a discrete and indirect method (see Chapter 2) has been used so as to exploit as well as possible – during the duration of pushes – all the past interactions between the algorithm and the process. The most classical indirect algorithm is *Dyna-Q* [SUT 90] (see Chapter 2, section 2.6.1). In its original version, *Dyna-Q* handles deterministic processes, which means that the same action in the same state always leads to the same next state. As a consequence, in a deterministic case, the transition model can be represented as a matrix with $|\mathcal{S}|$ lines and $|\mathcal{A}|$ columns, each input containing the next state. An extension of *Dyna-Q* to stochastic processes is immediate, but requires for each (s, a) pair a probability distribution over all next states, so that an $|\mathcal{S}| \times |\mathcal{A}| \times |\mathcal{S}|$ matrix is needed to store the probabilities. In our case, with 13,500 states and 6 actions, such a representation is not possible. Similarly, *Prioritized Sweeping* allows for learning a policy with a stochastic process but requires memorizing antecedence relations in the form of a matrix of $|\mathcal{S}| \times |\mathcal{A}|$ lines by $|\mathcal{S}|$ columns. Considering the number of states in the process to control, this method, although efficient, is not applicable either. For these reasons, a less memory consuming algorithm is used to control the manipulator.

11.3.2. A suitable algorithm: STM-Q

The algorithm being used, called *STM-Q* (*Short-Term Model-based Q-learning*), is an extension of *Dyna-Q* to weakly stochastic systems (see Algorithm 11.1) [ADD 05]. This is an indirect algorithm that looks for an optimal policy vis-à-vis the γ -weighted criterion. The *STM-Q* algorithm uses an intermediate representation between a single-record model, as *Dyna-Q* does, and an exhaustive-record model.

As an indirect method, *STM-Q* builds a model of the transitions and the rewards during the interactions with the process. This model is made of a table of queues (FIFO) that store, for each visited state-action pair, the different outcomes observed in the past: each (s, a) pair is associated with a queue $M(s, a)$ whose maximum size cannot exceed a number defined *a priori*, denoted by n_{\max} . Each element in $M(s, a)$ contains a pair made of the next state and the reward received while performing action a in state s . For example, if $n_{\max} = 4$ and the pair (s, a) has been visited at least 4 times, we can have a queue like

$$M(s, a) = \{(s'_{t_1}, r_{t_1}), (s'_{t_2}, r_{t_2}), (s'_{t_3}, r_{t_3}), (s'_{t_4}, r_{t_4})\} \quad (11.2)$$

with $t_1 < t_2 < t_3 < t_4$. These pairs obviously have different values if the process is not deterministic.

Algorithm 11.1: STM-Q (Short-Term Model-based Q-learning)

```

initialization
  for each  $(s, a) \in \mathcal{S} \times \mathcal{A}$  do
     $Q(s, a) \leftarrow Q_0$ 
     $M(s, a) \leftarrow \emptyset$ 
    /*  $M(s, a)$  is the queue of state-reward pairs
       observed during previous visits of the  $(s, a)$ 
       pair */
```

```

for each episode do
   $s \leftarrow \text{StateChoice}$ 
  while  $s$  is not a terminal state do
     $a \leftarrow \text{ActionChoice}(Q, s)$ 
    Perform action  $a$ , observe new state  $s'$  and new reward  $r$ 

    if  $\text{card } M(s, a) < n_{\max}$  then
      Add observation  $(s', r)$  to queue  $M(s, a)$ 
    otherwise
      Replace oldest observation in queue  $M(s, a)$  with observation
       $(s', r)$ 
     $s \leftarrow s'$ 

    repeat  $N$  times
      /* this part can be performed offline */
```

```

      Sample an already visited  $(s, a)$  pair at random
      
$$Q(s, a) \leftarrow \sum_{\forall(y, r) \in M(s, a)} \frac{1}{\text{card } M(s, a)} \left[ r + \gamma \max_{v \in \mathcal{A}} Q(y, v) \right]$$

```

The resulting model allows for computing an estimate of the transition probabilities of the process:

$$\hat{p}(s' | s, a) = \sum_{\forall(y, r) \in M(s, a) | y = s'} \frac{1}{\text{card } M(s, a)}. \quad (11.3)$$

This estimate is used to optimize the value function $Q(s, a)$ with a process similar to a value iteration algorithm (see Chapter 1). The update equation is then the one appearing at the last line of Algorithm 11.1.

The size n_{\max} of the waiting queue allows for adapting the algorithm to the “degree of determinism” of the process. For the sake of efficiency, n_{\max} has to

remain small (from about 10 to 50), so that *STM-Q* is rather suited to the control of weakly stochastic processes such as physical systems seen through digital measures and having a low-dimensional state space (dimension 2 or 3).

11.4. Experimental results

11.4.1. Experimental setup

As explained in section 11.2, the objective is to select the pushes that will bring the object to a given position as fast as possible. The *STM-Q* algorithm has therefore been implemented in the high level controller (see Figure 11.2). It receives the process state via the vision system and sends instructions to the low level controller (choice of the tip and length of the push).

The experiment consists of learning to position an object at the center of a manipulation area from any position. At the beginning of each episode, the object is placed at random in the area and the episode terminates when the object is correctly positioned.

The exploration method retained for *ActionChoice* (Q, s) in the algorithm is the ϵ -greedy method (see Chapter 2).

Before the first episode, the value function is initialized with $Q_0 = 0$. During the experiment, the values of the algorithm's parameters are

- $\epsilon = 0.1$,
- $\gamma = 0.9$,
- $n_{\max} = 10$,

– N equal to the number of previously visited distinct state-action pairs (over all episodes), i.e.

$$N = \text{card} (\{(s, a) \in \mathcal{S} \times \mathcal{A} \mid M(s, a) \neq \emptyset\}). \quad (11.4)$$

11.4.2. Results

The experiment lasted for a little more than 24 hours for a total of 34,134 pushes spread of 100 episodes. The curve of Figure 11.3 represents the number of pushes (actions) performed per episode.

At the beginning of the learning, an average of 400 to 500 pushes are necessary to position the object. The standard deviation is very large (about 500). The policy consists mainly of exploring the state space.

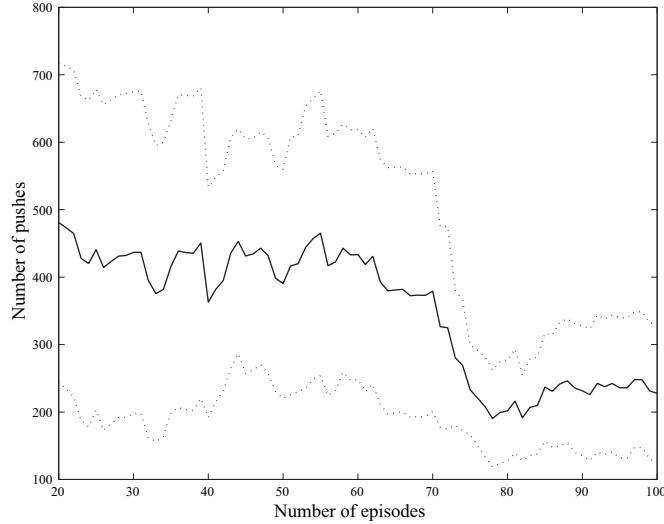
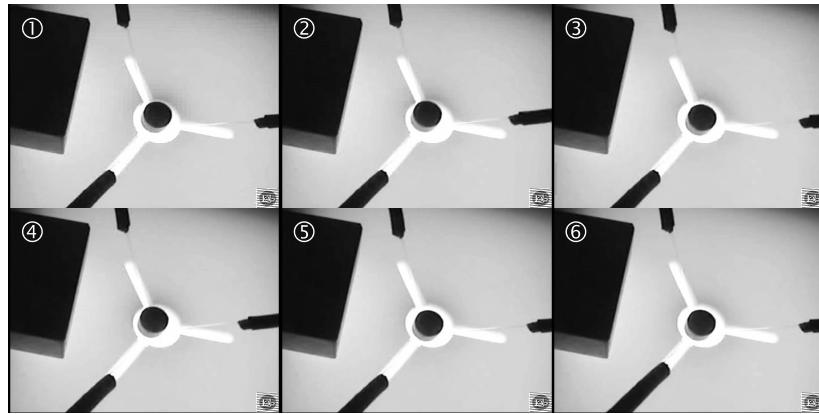


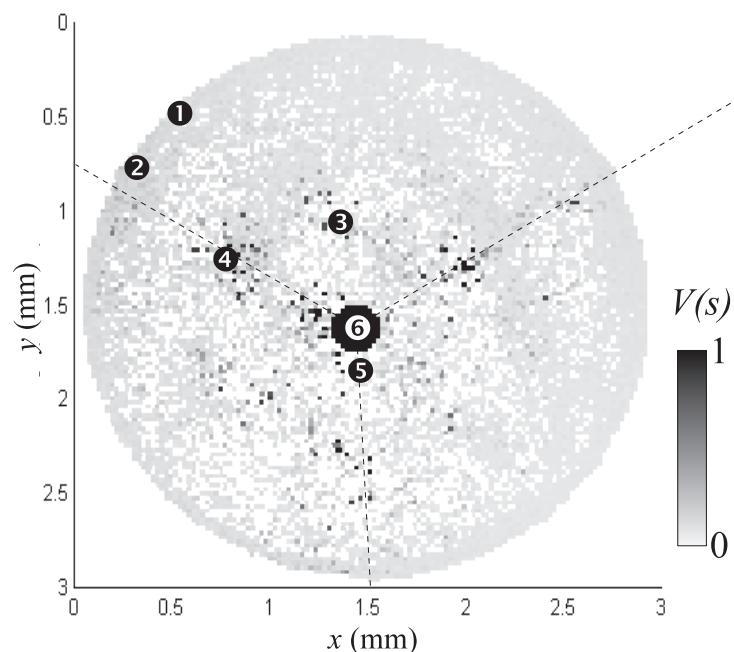
Figure 11.3. Result of the online learning: the solid line shows the mean number of pushes per episode (sliding average over the last 20 episodes). The dotted line shows the standard deviation centered around the mean

After 80 episodes, we observe a significant decrease of the episode length. An average 250 pushes are needed to position the object, but the standard deviation remains high (about 200). We can nevertheless obtain very short episodes as shown in Figure 11.4b. So, the STM-Q algorithm improves the initial random policy, but the performances are not uniform through the state space. When starting from some states, the learned policy is very efficient, and from other states the policy is not.

Indeed, the observation of longer sequences shows that the control is particularly delicate in some states for two major reasons. On one side, when the object is between two micro-tips against the wall, it is difficult to dislodge it from this position. On the other side, to reach the objective it is not sufficient to move the object closer to the center: first, the object must be placed in the axis of a micro-tip, then, the micro-tips pushes the object straight to the goal; if the objective is missed from a short distance, it is often necessary to restart the manipulation “from scratch” by pushing the object back to the wall. These two observations appear in the value function. Figure 11.4b shows that the value function has a higher value on the perimeter than in the middle. It means that the border states are more visited than middle ones. The three nodes located on the micro-tips’ axes at about one push from the target show that these area are near the goal in terms of action. Finally, there is a ring of low Q-values close to the objective. As we observed, the closest states to the goal in terms of distance are not close in terms of action.



(a) Manipulation sequence: ① initial state, ② long push of the right micro-tip, ③ long push of the top micro-tip, ④ long push of the right micro-tip, ⑤ long push of the top micro-tip, ⑥ short push of the bottom micro-tip. The micro-tips are not very visible because they are fine and transparent (optical fibers)



(b) Representation of the value function v and of the successively visited states during the manipulation sequence. The white points represent the non-visited states

Figure 11.4. Example of manipulation episode obtained after learning

	Total	Visited
Number of states	13,500	8,748
Number of state-action pairs	81,000	10,700
Number of actions per state	6	1 action in 7,138 states 2 actions in 1,138 states more than 3 actions in 375 states

Table 11.1. *State and action visit statistics after the online learning (which totals 34,134 actions)*

11.5. Conclusion

This experiment shows that it is possible to use an indirect reinforcement learning algorithm to learn online how to control a real-world process. Nevertheless, regarding the manipulator being studied, the results obtained are still far from a reliable and efficient automation. Admittedly, the manipulation task is hard even for a trained human operator. But these mixed results also confirm a classical need when learning: generalization.

The employed algorithm, *STM-Q*, builds a model of the process so as to update the value function as fast as possible, but does not generalize the value from one state to another close state, as illustrated by the very sparse aspect of the obtained Q-values (see Figure 11.4b). Moreover, visit statistics of the states (see Table 11.1) indicate that just over one out of eight state-action pairs have been visited during the experiment. As it is not reasonable to extend the already important duration of a learning session, a generalization mechanism would be necessary to reuse acquired experience for non-visited pairs. Thus, beyond this micro-manipulation application, learning online to control a real process requires facing a double challenge: use past experience at its best via indirect approaches and generalize acquired experience to adapt as well as possible to an unknown state. Ideas going in this direction are presented in Chapter 4.

11.6. Bibliography

- [ADD 05] ADDA C., LAURENT G J. and LE FORT-PIAT N., “Learning to control a real micropositioning system in the STM-Q framework”, *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA’05)*, pp. 4569–4574, 2005.
- [BAU 98] BAUR C., BUGACOV A., KOEL B. E., MADHUKAR A., MONTOYA N., RAMACHANDRAN T. R., REQUICHA A. A. G., RESCH R. and WILL P., “Nanoparticle manipulation by mechanical pushing: underlying phenomena and real-time monitoring”, *Nanotechnology*, vol. 9, pp. 360–364, 1998.

- [CHE 07] CHENG P., CAPPELLERI D. J., GAVREA B. and KUMAR V., “Planning and control of meso-scale manipulation tasks with uncertainties”, *Robotics: Science and Systems*, 2007.
- [GAU 06] GAUTHIER M., RÉGNIER S., ROUGEOT P. and CHAILLET N., “Forces analysis for micromanipulation in dry and liquid environments”, *International Journal of Micromechatronics*, vol. 3, no. 3, pp. 389–413, 2006.
- [GOR 06] GORMAN J. J. and DAGALAKIS N. G., “Probe-based micro-scale manipulation and assembly using force feedback”, *Proceedings of the International Conference on Robotics and Remote Systems for Hazardous Environments*, pp. 621–628, 2006.
- [HAN 98] HANSEN T. L., KÜHLE A., SORENSEN A. H., BOHR J. and LINDELOF P. E., “A technique for positioning nanoparticles using an atomic force microscope”, *Nanotechnology*, vol. 9, pp. 337–342, 1998.
- [KOR 09] KORAYEM M. H. and ZAKERI M., “Sensitivity analysis of nanoparticles pushing critical conditions in 2-D controlled nanomanipulation based on AFM”, *International Journal of Advanced Manufacturing Technology*, vol. 41, pp. 741–726, 2009.
- [MOL 02] MOLL M., GOLDBERG K., ERDMANN M. A. and FEARING R., “Orienting micro-scale parts with squeeze and roll primitives”, *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'02)*, pp. 11–15, 2002.
- [RES 00] RESCH R., LEWIS D., MELTZER S., MONTOYA N., KOEL B. E., MADHUKAR A., REQUICHA A. A. G. and WILL P., “Manipulation of gold nanoparticles in liquid environments using scanning force microscopy”, *Ultramicroscopy*, vol. 82, pp. 135–139, 2000.
- [SIT 04] SITTI M., “Atomic force microscope probe based controlled pushing for nanotribological characterization”, *IEEE/ASME Transactions on Mechatronics*, vol. 9, no. 2, pp. 343–349, 2004.
- [SUT 90] SUTTON R. S., “Integrated architectures for learning, planning and reacting based on approximating dynamic programming”, *Proceedings of the 7th International Conference on Machine Learning (ICML'90)*, San Mateo, CA, pp. 216–224, 1990.
- [ZES 98] ZESCH W. and FEARING R. S., “Alignment of microparts using force controlled pushing”, *Proceedings of the SPIE Conference on Microrobotics and Micromanipulation*, vol. 3519, pp. 148–156, 1998.

Chapter 12

Conservation of Biodiversity

12.1. Introduction

Globally, biodiversity is undergoing a rapid decline as a result of human activities [PIM 95]. Conversion of native forests and grasslands for agriculture, livestock grazing and forestry, resource extraction, urban and rural development coupled with an ever growing population are resulting in a loss of plants, animals and ecosystems at a rate never before experienced [ANO 07]. In the face of these rapid changes, species must adapt or risk extinction. The field of Conservation Biology has developed out of Ecology in an effort to quantify and predict the impacts of human actions on biodiversity and most importantly develop strategies to conserve and protect these species. Increasing the challenge of this task is the fact that resources to undertake biodiversity conservation are limited [WIL 06]. In response, we have seen the development of cost-effective conservation decision-making [POS 01]. Whether making decisions about the recovery of endangered species or the management of invasive species [REG 09], Markov decision processes (MDP) allow a clear and eloquent formulation of the optimal conservation resource allocation problem. There remains many complex ecological problems without solutions and only through collaboration of computer scientists with conservation biologists can we hope to find good solutions [MAC 09]. The application of MDP methods to conservation problems is in its infancy. Yet, as we will demonstrate in this chapter, their use offers substantial promise in solving complex ecological problems [WIL 09].

In this chapter, we examine two biodiversity conservation applications of Markov decision problems. First, we examine the problem of managing an endangered

Chapter written by Iadine CHADÈS.

species, the Sumatran Tiger (*Panthera tigris sumatrae*), which is difficult to observe (cryptic). For many endangered species, the problem of detection becomes increasingly challenging as populations decline and there are fewer individuals to observe. In this situation managers wish to know what is the optimal management action to conserve the species: When is it best to invest limited resources managing the species? When should we invest in surveying to assess the status of the species? And when, if ever, should we give up?¹ This example is the first application of partially observable Markov decision process models (POMDP) to a biodiversity conservation problem.

Second, we investigate the problem of how to recover two endangered species which interact as predator and prey. Northern abalone (*Haliotis kamtschatkana*) are the preferred prey of sea otters (*Enhydra lutris*), both co-habiting along the pacific northwestern coast of Canada and United States. At present, the recovery strategies for these two species are independent of each other and do not explicitly consider their interactions. We² provide for the first time an optimal recovery strategy for these two species which takes into account their functional relationship using two types of reinforcement learning algorithms over a finite-time horizon.

Finally we will discuss the need for further research development in the MDP community to solve challenging optimization problems in conservation biology.

12.2. When to protect, survey or surrender cryptic endangered species

12.2.1. Surveying and managing the Sumatran tiger

The Sumatran tiger, along with all species of tigers, has suffered dramatic declines in their populations as a result of diminishing food resources, habitat destruction and illegal poaching [LIN 06]. In the Kerinci Seblat region, Linkie *et al.* studied the consequences of increasing anti-poaching patrols on the extinction probability of a population of Sumatran tigers [LIN 06]. The conservation actions examined in their study were the reduction of poaching activity as a result of increased patrols and the surveillance of the status of the population. Approximately 30,000 US dollars were spent each year implementing these two actions, with around 2/3 of the budget allocated to patrols (c_m) and the remainder invested in surveillance (c_s). We estimated the potential cost of failing to conserve the population to be 175,134 US dollars a year (v) on the basis of the amount of funds collected annually by the program for the protection of the Sumatran tiger. We determined that the probability of local extinction of the population of tigers while the park is under protection from poaching to be

1. E. McDonald-Madden, M. McCarthy, B. Wintle, M. Linkie and H. Possingham are co-authors of this study, see [CHA 08].

2. T. Martin, J. Curtis, C. Barreto are the co-authors of this work, see [CHA 07].

$p_m = 0.058$, and otherwise, when not protected, $p_o = 0.1$. Similarly, we calculated the probability of detecting tigers within the reserve (d) to be 0.782 when surveillance was being undertaken and 0.001, if not.

POMDPs are an appropriate modeling framework for solving sequential decision-making problems when there is uncertainty about the state of the system. In this case the uncertainty is whether the species is present or not.

12.2.2. The model

Let $\mathcal{S} = \{\text{extinct}, \text{extant}\}$ define the finite set of states of the system. Let \mathcal{A} be the finite set actions controlling the states of the system $\mathcal{A} = \{\text{survey}, \text{manage}, \text{do nothing}\}$. Each action has an associated transition matrix denoted by P , where the function $P : \mathcal{S} \times \mathcal{A} \rightarrow Pr(\mathcal{S})$ defines for each state-action pair a probability distribution over \mathcal{S} . In our problem formulation we assume that once a species goes extinct there is no recolonization process and thus the population remains extinct indefinitely: $P(\text{extinct} | \text{extant}) = 1$. The cost/reward function $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ defines for each state-action pair the cost of the action and the benefits to the state. Finally, we take into account the incomplete observability of the system and define a finite set of possible observations $Z = \{\text{absent}, \text{present}\}$ and the corresponding observation function O which defines for each action-next_state pair a probability distribution over Z . For example, the probability of detecting the species given the species is extant and the previous decision to manage is defined by $O(\text{present} | \text{extant, manage})$.

The optimization procedure chosen aims to maximize the expected gains across a finite horizon and is defined by equation (7.12) in Chapter 7. To determine the optimal decision we used the incremental pruning [CAS 98] algorithm, which is available from the Cassandra toolbox.³

12.2.3. Results

In most publications and POMDP toolboxes, results are expressed in terms of the value of the expected sum of rewards. The objective is to find a method that maximizes this value and little if any attention is given to the form of the optimal policy. In conservation biology, the value of the expected sum of rewards is often meaningless. We must instead provide guidance to decision managers on the optimal policy and therefore analyze the solution and its sensitivity in detail.

We represent the optimal policy in two ways: a decision graph (Figure 12.1) and a function of two variables representing the optimal policy directly (Figure 12.2). We

3. pomdp-solve <http://pomdp.org/pomdp/code/>.

find that, if the tiger is known to be present in the reserve, it is optimal to manage the Sumatran tiger for 12 years. If after managing the tiger for 12 years the tiger remains undetected, it is optimal to switch all resources to surveying for 3 years. If the species is not seen within these 3 years, then the optimal strategy is to stop investing resources in conserving this species.

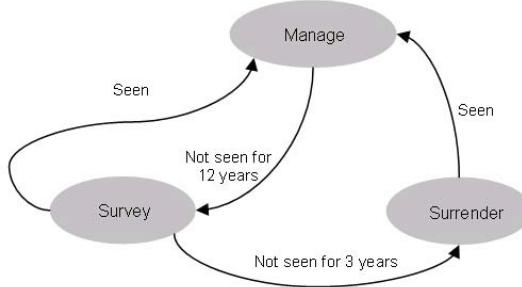


Figure 12.1. Decision graph representing the optimal policy [CHA 08]

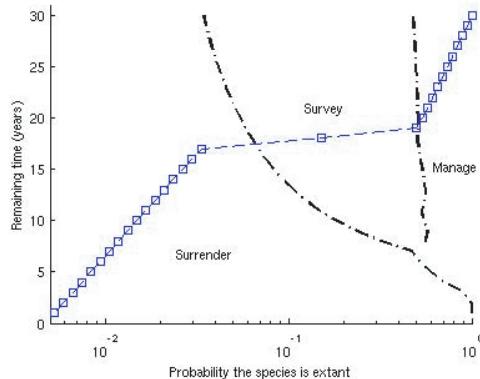


Figure 12.2. Optimal policy as a function of our belief about the extinction probability of the species. The plotted line with squares illustrates an example of the optimal management when the tiger is only seen at time horizon 30 and then remains unobserved [CHA 08]

In Figure 12.2 we represent the optimal policy for a 30 year time horizon. The figure illustrates the optimal decision to take as a function of the remaining time and the state of our belief about the presence of the species. The line denoted by the squares illustrates a scenario where our belief in the persistence of the tiger declines each year the tiger remains unobserved. Our belief declines most rapidly when we survey but fail to detect the tiger. If the tiger is observed at any point during this process, there is no longer any uncertainty about the presence of the species and the optimal decision is to return to the start of the decision process and thus implement management.

To understand the role of the different parameters in our model we undertook a sensitivity analysis examining: the role of the value of the species relative to its management cost, the influence of detection probabilities (Figure 12.3) and the effect of extinction probabilities [CHA 08]. This sensitivity analysis allows us to generalize to other endangered species such as the whooping crane (*Grus americana*) and Mexican spotted owl (*Strix occidentalis lucida*). In our generalization, we find that a similar pattern is observed, where it is always optimal to first manage a species for T_m years before surveying it for T_s years. The higher the probability of extinction of the species, the more important it is to invest in the management of the species rather than spend resources on surveillance. This can be explained by the urgency of the situation where the economic value of the species has less influence on the optimal strategy than the remaining time available to save the species.

Finally it is possible to approximate an analytic solution to this problem. The belief values of the persistence of the species that would result in a change to the optimal decision (e.g. manage vs survey, or survey vs do nothing) can be expressed through equations using the parameters of our problem. We can also derive the corresponding length of time we should undertake a particular action. Interested readers can refer to [CHA 08]. Approximating the solution analytically makes it possible to better understand the role of parameters in the model and importantly produce a simple tool for managers of reserves to assist decision-making about how to best conserve cryptic endangered species without having to undertake a POMDP. The quality of our approximation is represented by the dashed lines in Figure 12.3.

In this example, we formulated the problem of when to manage, survey or stop the management of a reserve for an endangered species. We found an eloquent solution to the problem using POMDPs and we show that it can be approximated analytically. This study paves the way for more complex analyses of the allocation of resources for the conservation of endangered species. Although exact algorithms to solve POMDPs are inefficient for most problems, they can be very useful in solving the problem of when to invest in gaining information versus exploiting this information at the scale illustrated here. On small-size problems we could approximate an analytical solution. Analytical approximation of solutions are valuable in conservation biology as they provide a simple way of deriving a solution for decision managers. Unfortunately, apart from a method developed by Grosfeld-Nir [GRO 96] for uniform observations, no analytical formula has been proposed to resolve the practical issue of computing a stopping rule with POMDPs. New results in that area of research would have a high impact in applied domains.

12.2.4. Extension to more than one population

In the previous example we investigated the use of POMDPs for solving a resource allocation problem for a single population. In reality we need to consider the optimal management of multiple populations across multiple reserves. In Sumatra for

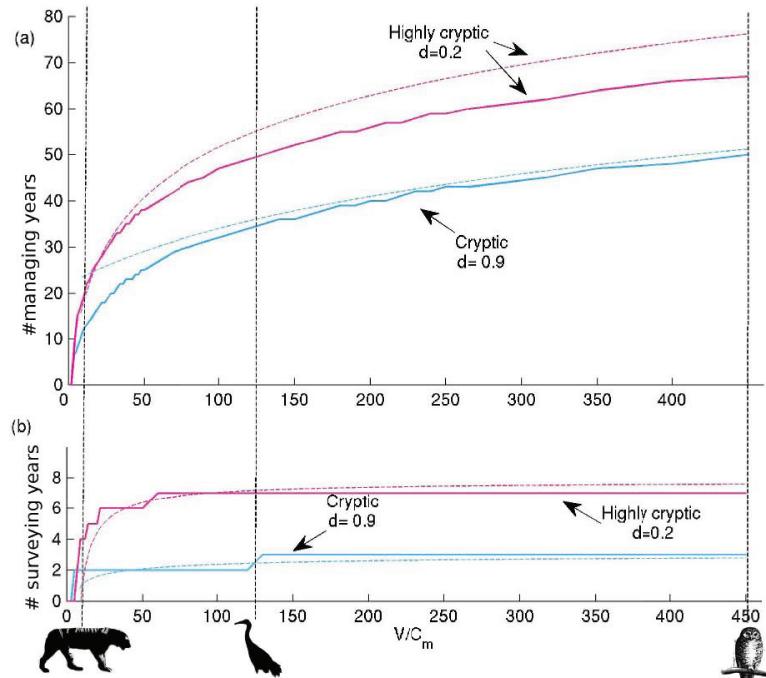


Figure 12.3. Sensitivity of the optimal strategy when the probability of detection (d) varies with the economic ratio value of the species/management cost (V/c_m). The continuous curves illustrate the optimal solution derived from our exact numerical method; the dashed lines represent our analytic approximation

example there are several reserves across the island and resources to manage them are limited. We therefore examined what the optimal management strategy would be when we have two reserves each containing a population of tigers. In this case the efficiency of our actions depends on the task: the simultaneous protection of two populations of tigers is less efficient locally. The resources previously directed at a single population must now be divided between two populations. We examined strategies that maximized the number of viable populations of tigers and how the quality of the habitat influenced the performance of our actions. Large reserves have a higher probability of persistence than small reserves. We therefore needed to find strategies which took into account this information.

We study this problem using a multi-agent POMDP which distinguishes states of population a from states of population b : $\mathcal{S} = \mathcal{S}_a \times \mathcal{S}_b$. The finite set of actions \mathcal{A} comprises 5 actions $\mathcal{A} = \{P_a, P_b, P_{ab}, P_a S_b, S_a P_b\}$ that define our conservation decisions (manage a , manage b , manage a and b , manage a and survey b , manage b and

survey a , and survey both a and b). The transition probabilities associated with each action are represented by a set of transition matrices T_a giving, for each state-action pair, a probability distribution over \mathcal{S} . The reward function takes into account the quality of the state (1 point for each viable population). Finally, we take into account the partial observability of the tiger and represent the probability of detection as a function of the state and action chosen using the observation function O . We denote by $\Omega = \Omega_a \times \Omega_b$ the finite set of observations of the system.

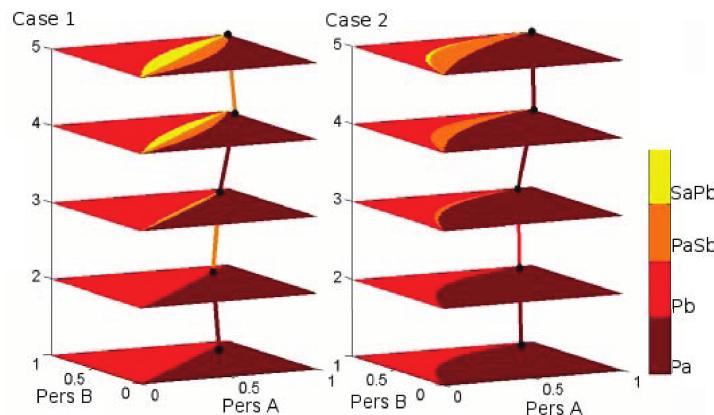


Figure 12.4. Influence of the probability of extinction on the optimal strategy for two cases with a time horizon of 5 years. Pers A and Pers B represent the belief in the persistence of populations A and B

Examining solutions such as those presented in Figure 12.4 allowed us to derive simple rules of thumb for management of multiple reserves. For example, an important result is that it is more efficient to alternate the protection between the two reserves for the Sumatran tiger than trying to manage the two reserves simultaneously but with less efficiency. In Figure 12.4, in the first case the populations have the same quality habitat and probability of extinction; alternating the management between the two is optimal. In the second case, population b has a higher probability of extinction than population a ; the optimal strategy here is to favor the protection of population a at the detriment of population b . Readers interested in learning more about this study are referred to [MCD 08, MCD 09].

12.3. Can sea otters and abalone co-exist?

12.3.1. Abalone and sea otters: two endangered species

Sea otters and northern abalone of British Columbia are protected under the Species at Risk Act (SARA) and plans for their recovery and protection are being

developed. During the 20th century sea otters were hunted to near extinction for their fur. By 1905 sea otters were extirpated from the coast of British Columbia. Fortunately, several small populations survived in Alaska. During the 1980's a plan to reintroduce sea otters from Alaska to British Columbia was put in place. Since the re-introduction of approximately 65 individuals on the western coast of Vancouver Island, the population has grown to over 3,500 individuals. The key threat to sea otter populations since the cessation of hunting are oil spills.

Northern abalone is a mollusc inhabiting near-shore rocky reefs and kelp forest along the pacific-northwest coast. Along with being a prized mollusc for human consumption, northern abalones are the preferred prey of sea otters. After the extirpation of sea otters, populations of abalone increased markedly. This in turn led to the development of a substantial commercial abalone fishery. Stocks began to decline in the 1970's and quotas were set in place. However, this was insufficient to halt the dramatic decline in the abalone population and the fishery was closed in 1993. Density of abalone ranged between 1.2–2 abalone/m² to less than 0.1–0.3 abalone/m². Since the closure of the fishery and legal protection of the species through listing under SARA, a lucrative black market for abalone has resulted in substantial illegal poaching and populations have failed to recover.

The decline in abalone populations coupled with the extirpation of the sea otters resulted in an unforeseen ecological disaster: the explosion of sea urchins which in turn browsed kelp forests, a key abalone habitat, to the point where once lush forests were transformed into the equivalent of a desert. Today even with protection, abalone have not recovered to levels previously recorded. There are several hypotheses as to why this is the case: abalones requiring a minimum density for effective reproduction, competition for habitat with sea urchins, illegal poaching activity. It is estimated that over 40 tonnes of abalone are harvested illegally each year, the equivalent of the quota set in place prior to the fisheries' closure in 1993. It is within the context of an expanding sea otter population that we want to determine strategies to aid the recovery of both species and ask at what densities can sea otters and abalone coexist?

To answer this question we chose to model the population dynamics of both populations as well as the interaction between the populations (their functional response). Using a simulator, we analyzed the effects of conservation decisions used to manage each species. We used two Reinforcement Learning algorithms adapted for a finite-time horizon: QH-learning and RH-learning.

12.3.2. The models

12.3.2.1. Population dynamics of abalone

We chose to model the demographic dynamic of the population following Bardos *et al.* [BAR 06]. This size-structured matrix population model represents

a compromise between a multi-agent systems model and a deterministic model. Elements of the matrix represent growth, survival and fecundity transition probabilities for different size classes of abalone. These values are used to simulate deterministically the average population dynamics. This is a common type of approximation used in the ecological community, where fluctuations in these transition probabilities are ignored. This type of approximation is verified in the limit by large populations [MAY 73].

More formally, the state of the population at instant n is defined by the vector $\mathbf{x}(n) = (x_1(n), \dots, x_7(n))$ with 7 size classes. Class 1 represents the post-larvae stage. Classes 2 to 4 are the juveniles and classes 5 to 7 represent the adult stages. The evolution of abalone populations at instant $n + 1$ is defined by $\mathbf{x}(n + 1) = G \cdot \mathbf{x}(n)$ with transition matrix G :

$$\begin{pmatrix} 0 & 0 & 0 & 0 & f_5 s_5 & f_6 s_6 & f_7 s_7 \\ g_{2,1} s_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ g_{3,1} s_1 & g_{3,2} s_2 & 0 & 0 & 0 & 0 & 0 \\ g_{4,1} s_1 & g_{4,2} s_2 & g_{4,3} s_3 & g_{4,4} s_4 & 0 & 0 & 0 \\ 0 & g_{5,2} s_2 & g_{5,3} s_3 & g_{5,4} s_4 & g_{5,5} s_5 & 0 & 0 \\ 0 & 0 & g_{6,3} s_3 & g_{6,4} s_4 & g_{6,5} s_5 & g_{6,6} s_6 & 0 \\ 0 & 0 & 0 & g_{7,4} s_4 & g_{7,5} s_5 & g_{7,6} s_6 & g_{7,7} s_7 \end{pmatrix}.$$

g_{ij} defines the growth matrix g , or in other words the transition probability of an individual from class j to class i . The survival probability of an individual at size class j is defined by s_j . f_i represents the fecundity of class i multiplied by the probability of fertilization and the larval survival probability. In other terms, f_i is the average number of post-larvae produced by an individual in class i each time step. The f_i s are multiplied by survival probability and is dependent on the density of adults (class 5 to 7). In the absence of threats the abalone population stabilizes at 1 abalone/m² (first 25 years in Figure 12.5).

Poaching of abalone is a key threat to population recovery [GAR 00, JUB 00]. In our model we assume the pressure of poaching to be similar to that of commercial fishing, targeting only the largest size class [BAR 06]. A stochastic process governs the probability of success of poaching removing 90% of class 7 with probability 0.75 and 70% with probability 0.25 every year. Simulation with poaching threat leads to 0.31 abalone/m² in the absence of sea otter predation.

12.3.2.2. Sea otter population model

Sea otter population dynamics were modeled using a Beverton-Holt model described by Gerber *et al.* [GER 04]. This model includes an asymptotic relationship

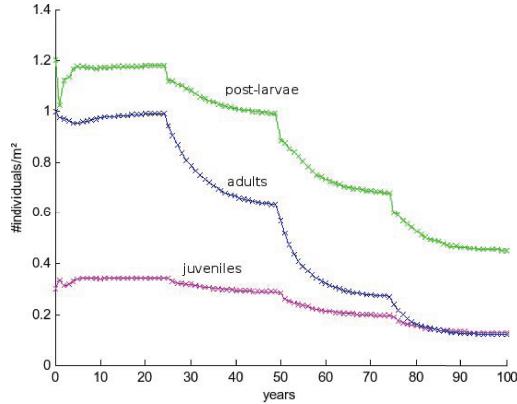


Figure 12.5. Simulation of the abalone population dynamics over time with increasing predation rates ($t = 0$ none $t = 25$ weak, $t = 50$ medium and $t = 75$ strong)

between density and recruitment:

$$N_{t+1} = \frac{e^r K N_t}{e^r N_t - N_t + K},$$

where K is the carrying capacity, N_t is the current population size, and r is the intrinsic rate of increase. The parameters are based on a population of sea otters occurring in Washington State with $K = 612$ and $r = 0.26$. While predators such as orcas, sharks and bald eagles do impact sea otters, oil spills pose the single largest threat to sea otter populations. In our sea otter model we implement a stochastic process by which oil spills may occur every 10 years on average with intensity varying from 0.2 to 0.4, reducing N by 20–40%. This rate of occurrence is an optimistic figure. Figure 12.6 illustrates the dynamics of the sea otter population under stochastic occurrence of oil spills.

12.3.2.3. States

We define S_a the set of states of our Northern abalone population. It is not feasible to consider a continuous state MDP because of the computational complexity required for this problem. We therefore model the adult abalone population (classes 5 to 7) using a finite set of 20 states. Each state represents a range of 0.05 density which corresponds to one of four hypothetical categories of threat (Table 12.1).

We define S_s the finite set of states representing 10 levels of the sea otter population. Each state represents a 10% increment of the population's carrying capacity, e.g. when $K = 612$, the sea otter population is in state zero when its population abundance is in between 0 and 61 individuals. Additionally, sea otters are assumed to be in one of 4 threat classes (Table 12.1).

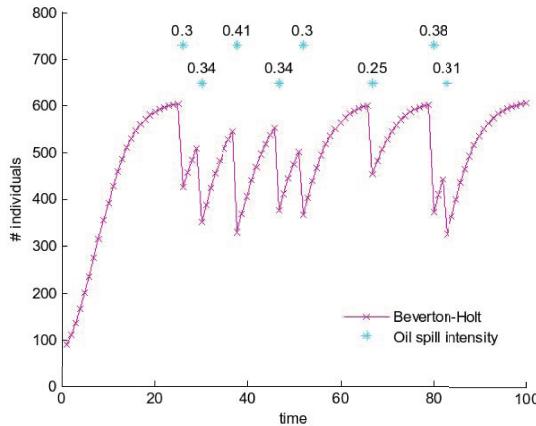


Figure 12.6. Simulation of the sea otter population dynamics under stochastic occurrences of oil spills

Abalone density (m^{-2})	< 0.1	< 0.3	< 0.5	≥ 0.5	
Status	Endangered	Threatened	Special concern	Not at risk	
Sea otters abundance (%K)	0	≤ 30	≤ 40	≤ 60	≥ 60
Status	Extinct	Endangered	Threatened	Special concern	Not at risk

Table 12.1. Classification status following the threat status

Finally we define \mathcal{S} the set of states of our problem $\mathcal{S} = \mathcal{S}_a \times \mathcal{S}_s$, so that $|\mathcal{S}| = 200$. The initial state of our problem assumes that sea otters are not yet present in the area and poaching activity occurs.

12.3.2.4. Decisions

Five management actions are considered and define the set \mathcal{A} of actions: do nothing (N), reintroduce sea otters (RI), enforce anti-poaching (A), control sea otters (C) and the combined action of sea otter control and anti-poaching (AC).

Anti-poaching enforcement is one of our conservation actions for abalone recovery. We model the effects of anti-poaching enforcement by stochastically reducing illegal fishing from 90% to 10% with probability 0.75 or to 30% with probability 0.25. Thus, even when anti-poaching measures are implemented, poaching still occurs but at a reduced intensity.

The decision control of sea otters reduces the sea otter population by 3% of the carrying capacity each year. This action can only occur when sea otters are in a “not-at-risk” state.

12.3.2.5. Interaction between sea otters and abalone

In the absence of a mathematical model describing the interaction between sea otters and abalone, we derived three hypothetical functional responses based on the literature:

- The first response assumes the threat from sea otters on abalone only depends on abalone density (F1): the predation rate increases with abalone density (Figure 12.7).
- The second functional response assumes sea otter quantities drive the response. Here the level of predation imposed on abalone irrespective of abalone density will increase with increasing sea otter abundance (F2).
- Finally the third function is influenced by the abundance of both sea otters and abalone. It resembles a sigmoid response where the predation rate accelerates at first as prey density increases and then decelerates towards satiation at high prey densities (F3). Sigmoid functional responses are typical of generalist predators, like sea otters, which readily switch from one prey species to another and/or which concentrate their feeding in areas where certain resources are most abundant.

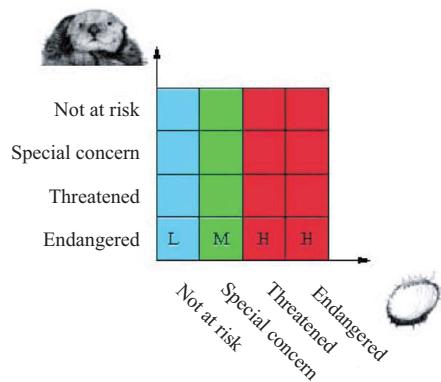


Figure 12.7. Schematic of functional response (F1) between sea otters and abalone (low predation, medium predation, high predation)

The survival rate of classes 3 to 7 is decreased by 5% (L) when predation is low, 15% (M) when predation is medium and 25% (H) when predation is high (Figures 12.5 and 12.7).

12.3.2.6. Multicriteria objective and reward function

To determine an optimal management strategy, we need to define our objective. Here we used two types of optimality criteria: maximize the probability of having both species at “not-at-risk” or “special-concern” levels simultaneously (R1), or independently (R2). The first criterion provides a reward only when both sea otter

and abalone populations are in “not-at-risk” or “special-concern” states, with highest rewards when both are not-at-risk. Vice versa, the second criterion provides rewards when at least one population is in a “not-at-risk” or “special-concern” state and reflects a trade-off that may ensue if both populations cannot be maintained at levels outlined in the respective recovery strategies (Figure 12.8).

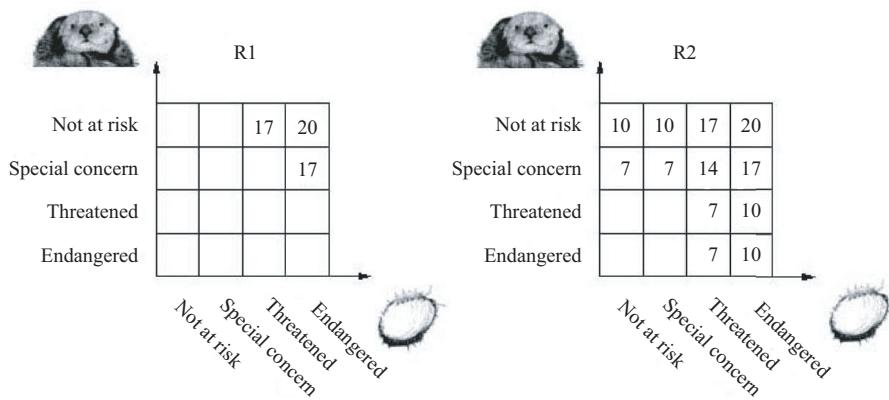


Figure 12.8. R1 represents the joint reward function and R2 represents the individual reward

12.3.3. Methods

We used QH-learning and RH-learning algorithms [GAR 98] to learn optimal conservation strategies over a 50-year time horizon. There is no proof of convergence of these finite-horizon learning algorithms; however, experimental results have shown their efficiency to converge towards good strategies.

12.3.4. Results

Our goal is to determine the management conditions under which both species can co-exist at “not-at-risk” levels over a 50-year time period. Both reinforcement learning algorithms were run 500,000 times with 10 time-steps each to learn near optimal policies. Decisions were taken every 5 years. For all the following results we set the starting state by simulating abalone population dynamics in the presence of poaching but absence of sea otters.

12.3.4.1. Scenario 1: sea otter reintroduction and anti-poaching enforcement

We first examine two management actions currently being used in the northeast Pacific Ocean: reintroduction of sea otters and anti-poaching enforcement.

Under functional response 1, rewards 1 and 2, we find that it is optimal to introduce sea otters at the first time step (RI). Anti-poaching enforcement (A) is then the optimal decision for the remaining time horizon. Table 12.2, columns 2 and 3, lines QL1 and RL1, represent the average performance of this scenario. Sea otters stabilize around “not-at-risk” and “special-concern” varying with oil spills occurrence. The northern abalone population oscillates in between “threatened” and “special-concern” status. These results can be explained due to the switch between predation pressures under functional response 1 (Figure 12.7).

Under functional response 2, R1 (Figure 12.9) and R2, the optimal strategy is to first increase the level of abalone to “not-at-risk”. This level can be reached after 10 to 15 years of anti-poaching enforcement. Sea otters are then reintroduced. For R1 only, rewards +14 and +17 are obtained and occur when both species are at “special-concern” or when abalones are at “special-concern” and sea otters are at “not-at-risk”. Here, the abalone population fluctuates with changes in sea otter population density as a result of oil spills because under functional response 2 predation pressure is related to sea otter density. Performances observed in Table 12.2, columns 4 and 5, represent the low numbers of accumulated rewards under functional response F2.

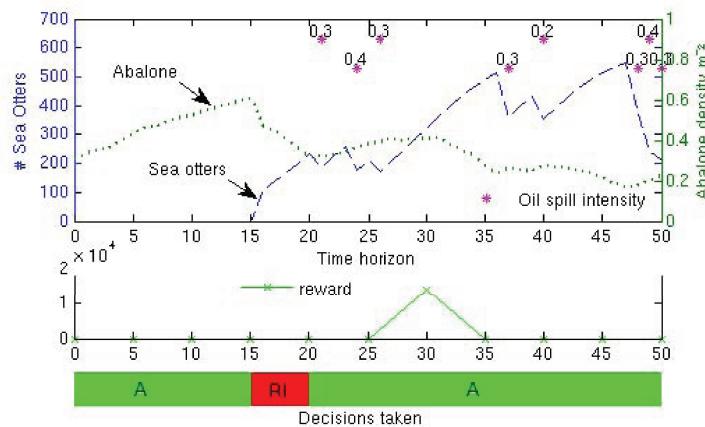


Figure 12.9. Scenario 1: simulation of the best strategy (functional response F2 and reward function R1)

Under functional response 3, rewards 1 and 2, the optimal strategy is always to introduce sea otters at the first time-step. Anti-poaching enforcement is then the optimal decision for the remaining time-horizon. Only two valuable states are reached: when both species are at “special-concern” (+14) or when abalones are at “special-concern” and sea otters are at “not-at-risk” (+17). Abalone density

	F1	F1	F2	F2	F3	F3
	R1	R2	R1	R2	R1	R2
QL1	66.06	111.85	19.90	98.96	53.63	108.28
RL1	66.11	111.44	20.50	98.55	53.94	108.33
QL2	66.29	111.36	19.47	99.11	53.07	108.72
RL2	66.10	111.78	19.48	99.12	54.92	108.65
QL3	66.17	111.56	25.39	100.06	84.67	119.95
RL3	66.05	111.44	24.73	99.90	84.53	119.64

Table 12.2. Performances for each scenario and algorithm using the average sum of expected rewards

decreases with increases in sea otter population until the former reaches threatened or endangered status, when it becomes less impacted by sea otters. As defined by functional response 3, abalone density fluctuates in response to changes in sea otter population abundance which are driven by oil spills. Columns 6 and 7, Table 12.2, give the average cumulated rewards under F3.

12.3.4.2. Scenario 2: control of sea otters

Implementing sea otter reintroduction and removal does not improve the overall performance of optimal strategies (lines QL2, RL2, Table 12.2).

12.3.4.3. Scenario 3: combined action of sea otter control and anti-poaching

The combined actions of sea otter reintroduction, control and anti-poaching outperform all other management scenarios. After sea otters are reintroduced during the first time step, population control is implemented when sea otters are not-at-risk with functional responses 2 and 3 (lines QL3, RL3, Table 12.2; Figure 12.10). This scenario does not change the performance with functional response 1 as the threat to abalone is independent of sea otter density. This combined strategy allows abalone to increase in density until density-dependent effects come into play as defined under functional response 3. Unfortunately this scenario does not allow us to achieve our conservation objective as “not-at-risk” levels remain unachievable for both species.

12.3.5. Conclusion

In the sea otter and abalone case study, we fail to find a management strategy which allows both species to co-exist at “not-at-risk” levels. There are several possible reasons for this.

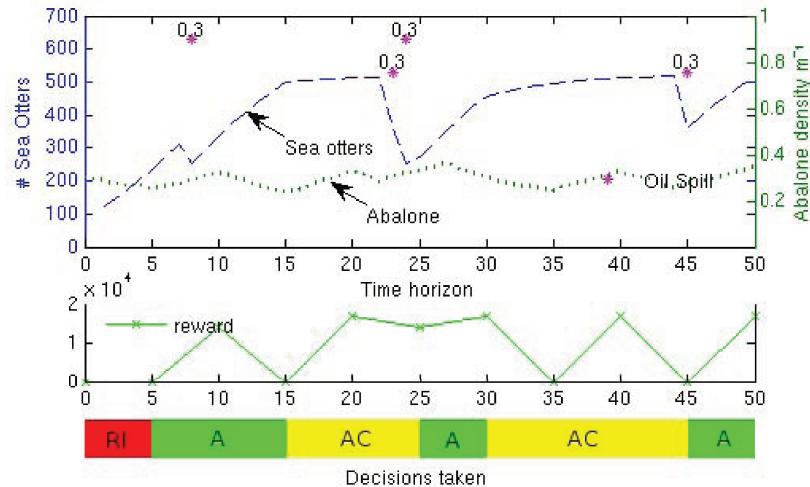


Figure 12.10. Scenario 3: simulation of the best strategy (functional response F3 and reward function R2)

First, our models and assumptions may be unrealistic. In the absence of published information on abalone and sea otter functional responses we have assumed three contrasting responses based on the literature. In reality the functional response could be a mix between these interactions. Due to the rarity of the species studied here it is unlikely that we will be able to build better population models. A sensitivity analysis on the key parameters of the functional responses might help us to circumvent the lack of data. We are currently working on improving the northern abalone population model we used in this work.

A second possibility for not achieving co-existence at “not-at-risk” levels is that the densities specified for recovery of abalone may be unrealistic in the presence of sea otters. Interestingly, Watson [WAT 00] argues that sea otter recovery and abalone fisheries are mutually exclusive corroborating what we find here. We do find, however, that, with anti-poaching enforcement and sea otter removal, co-existence at “not-at-risk” (sea otter) and “special-concern” (abalone) can be achieved and outperforms all other scenarios assessed here.

This work illustrates the interest of reinforcement learning methods to help solve conservation biology problems. To our knowledge this is the first application of reinforcement learning for optimal management of interacting species at risk. There is a need to develop these methods further in order to increase our capacity to solve the complex management problems arising from species interactions such as those described here.

12.4. Other applications in conservation biology and discussions

Increasingly many problems in ecology require the use of new optimization methods. In this chapter we presented two applications of Markov decision problems and optimization methods to problems in conservation biology. In these two problems we provided clear results and useful rules of thumb. While research in the MDP community and methods to solve Markov decision problems generally focuses on increasing the average performance, these results become meaningless when dealing with problem in conservation biology. Indeed, when working in conservation biology the challenge of solving a problem is further complicated by the need to explain the meaning of the results and provide guidance to decision managers. Very fast or efficient algorithms that do not provide understandable results are not useful for ecologists. Through this work we have identified several developments of research for computer scientists who would like to engage or contribute to the field of conservation biology.

In reinforcement learning or planning, it is clear that several strategies might be optimal or near optimal. Amongst that set of strategies some show clear structures and boundaries and therefore can be explained or applied easily. Approximate methods like reinforcement learning often provide a few “good” or near optimal strategies that do not have such a structure due to the lack of convergence or the way they are solved. There is an opportunity here to develop methods that try to build strategies in order to maximize the coherence of the strategy rather than maximizing the sole performance criterion. Development of methods based on policy search algorithms (Chapter 5) and Factored MDP representations (Chapter 4) should be encouraged. An example of a contribution to this research area is [DEG 06], where the authors propose a method to solve an FMDP without prior knowledge of the structure of the problem.

Taking into account expert knowledge prior to online learning has a strong potential [POU 06]. Indeed, under climate change, habitats are changing, conservation biologists need to make smart decisions to manage endangered and invasive species. Should we start translocating threatened species at high cost and risk of failing [ROU 09], or should we wait and risk a deadly decrease in population abundance? In the absence of information about the translocation process, we might need to first rely on expert opinions and integrate their prediction into our optimization model but we should also learn and adapt our conservation strategy as we gather new data. The challenge is to save biodiversity at a minimum cost and funds invested must make a difference.

Another avenue of research that promises to have a high impact in ecology is related to the field of POMDPs. We need more theoretical work on small-size problems that show analytical solutions [GRO 96]. Understanding the conservation mechanisms at a small scale is a key element to derive rules of thumb before we tackle them on a larger scale. Solving large POMDPs has become a reachable objective but again

representing their solution remains a difficult task. The trade-offs between gathering information (surveying) and acting (managing a species) is already attracting attention [MAC 09] and more work need to be done when detection probability varies over time.

Other potential areas of strong interest are linked with robust optimization problems (see Chapter 10, section 10.3) and online decision-making for large state space (see Chapter 6 and [NIC 09]). The combined skills of ecology and computer science are showing great promise in solving complex conservation problems. We encourage researchers in AI to consider problems in conservation biology when looking for insightful applications of their methods or new research avenues.

12.5. Bibliography

- [ANO 07] ANONYMOUS, “IUCN cat projects database”, October 2007, Zoological Society of London (ZSL).
- [BAR 06] BARDOS D. C., DAY R. W., LAWSON N. T. and LINACRE N. A., “Dynamical response to fishing varies with compensatory mechanism: An abalone population model”, *Ecological Modelling*, vol. 192, no. 3-4, pp. 523–542, 2006.
- [CAS 98] CASSANDRA A. R., Exact and approximate algorithms for partially observable Markov decision processes, PhD thesis, Brown University, 1998.
- [CHA 07] CHADES I., MARTIN T., CURTIS J. and BARRETO C., “Managing interacting threatened species: a reinforcement learning decision theoretic approach”, *Proceedings of the International Congress on Modelling and Simulation (MODSIM’07)*, Modelling and Simulation Society of Australia and New Zealand, 2007.
- [CHA 08] CHADES I., McDONALD-MADDEN E., MCCARTHY M. A., WINTLE, B., LINKIE M. and POSSINGHAM H. P., “When to stop managing or surveying cryptic threatened species”, *Proceedings of the National Academy of Sciences*, vol. 105, pp. 13936–13940, 2008.
- [DEG 06] DEGRIS T., SIGAUD O. and WUILLEMIN P. H., “Chi-square tests driven method for learning the structure of factored MDPs”, *Proceedings of the 22nd Conference on Uncertainty in Artificial Intelligence (UAI’06)*, Cambridge, MA, pp. 122–129, 2006.
- [GAR 98] GARCIA F. and NDIAYE S. M., “A learning rate analysis of reinforcement-learning algorithms in finite-horizon”, *Proceedings of the 15th International Conference on Machine Learning (ICML’98)*, Morgan Kaufmann, San Mateo, CA, pp. 215–223, 1998.
- [GAR 00] GARDNER J., GRIGGS J. and CAMPBELL A., “Summary of a strategy for rebuilding abalone stocks in British Columbia”, *Canadian Special Publication of Fisheries and Aquatic Sciences*, vol. 2000, pp. 151–156, 2000.
- [GER 04] GERBER L. R., TINKER M. T., DOAK D. F., ESTES J. A. and JESSUP D. A., “Mortality sensitivity in life-stage simulation analysis: a case study of southern sea otters”, *Ecological Applications*, vol. 14, no. 5, pp. 1554–1565, 2004.

- [GRO 96] GROSFIELD-NIR A., “A two-state partially observable Markov decision process with uniformly distributed observations”, *Operations Research*, vol. 44, no. 3, pp. 458–463, 1996.
- [JUB 00] JUBINVILLE B., “Enforcing the fishery closure for northern (pinto) abalone (*Haliotis kamtschatkana*) in British Columbia”, *Workshop on Rebuilding Abalone Stocks in British Columbia*, vol. 130, p. 52, 2000.
- [LIN 06] LINKIE M., CHAPRON G., MARTYR D. J., HOLDEN J. and LEADER-WILLIAMS N., “Assessing the viability of tiger subpopulations in a fragmented landscape”, *Journal of Applied Ecology*, vol. 43, no. 3, pp. 576–586, 2006.
- [MAC 09] MACKENZIE D. I., “Getting the biggest bang for our conservation buck”, *Trends in Ecology & Evolution*, vol. 24, no. 4, pp. 175–177, 2009.
- [MAY 73] MAY R. M., *Stability and Complexity in Model Ecosystems*, vol. 6 of *Monographs in Population Biology*, Princeton University Press, Princeton, NJ, 1973.
- [MCD 08] McDONALD-MADDEN E., CHADES I., McCARTHY M., LINKIE M. and POSSINGHAM H. P., “Should I spread my risk or concentrate my efforts: is triage of a subpopulation ever the best decision”, *Proceedings of the International Congress on Modelling and Simulation (MODSIM'07)*, Modelling and Simulation Society of Australia and New Zealand, 2007.
- [MCD 09] McDONALD-MADDEN E., CHADES I., McCARTHY M., LINKIE M. and POSSINGHAM H. P., Allocating conservation resources between areas where persistence of a species is uncertain, submitted, 2009.
- [NIC 09] NICOL S., CHADES I. and POSSINGHAM H. P., “Conservation decision-making in large state spaces”, *Proceedings of the International Congress on Modelling and Simulation (MODSIM'09)*, Modelling and Simulation Society of Australia and New Zealand, 2009.
- [PIM 95] PIMM S. L., RUSSELL G. J., GITTELMAN J. L. and BROOKS T. M., “The future of biodiversity”, *Science*, vol. 269, no. 5222, pp. 347–350, 1995.
- [POS 01] POSSINGHAM H. P., ANDELMAN S. J., NOON B. R., S. T. and PULLIAM H. R., “Making smart conservation decisions”, SOULE M. E. and ORIANS G. H., Eds., *Conservation Biology: Research Priorities for the Next Decade*, Island Press, Washington DC, pp. 225–244, 2001.
- [POU 06] POUPART P., VLASSIS N., HOEY J. and REGAN K., “An analytic solution to discrete Bayesian reinforcement learning”, *Proceedings of the 23rd International Conference on Machine Learning*, Pittsburgh, PA, pp. 697–704, 2006.
- [REG 09] REGAN T. J., CHADES I. and POSSINGHAM H. P., Using imperfect information to optimally manage plant invasions, Submitted, 2009.
- [ROU 09] ROUT T. M., HAUSER C. E. and POSSINGHAM H. P., “Optimal adaptive management for the translocation of a threatened species”, *Ecological Applications*, vol. 19, no. 2, pp. 515–526, 2009.
- [WAT 00] WATSON J. C., “The effects of sea otters (*Enhydra Lutris*) on abalone (*Haliotis* spp.) populations”, *Workshop on Rebuilding Abalone Stocks in British Columbia*, vol. 130, pp. 123–132, 2000.

- [WIL 06] WILSON K. A., MCBRIDE M. F., BODE M. and POSSINGHAM H. P., “Prioritizing global conservation efforts”, *Nature*, vol. 440, no. 7082, pp. 337–340, 2006.
- [WIL 09] WILLIAMS B. K., “Markov decision processes in natural resources management: Observability and uncertainty”, *Ecological Modelling*, vol. 220, no. 6, pp. 830–840, 2009.

Chapter 13

Autonomous Helicopter Searching for a Landing Area in an Uncertain Environment

This chapter presents the application of MDPs to a problem of strategy optimization for an autonomous search-and-rescue helicopter exploring a partially known and uncertain environment in search for a landing zone. Online MDP optimization algorithms were developed and implemented in a real-time aerial robotic architecture, thus providing on-board autonomous bounded-time decision-making capabilities for uninhabited rotorcraft systems. Both theoretical aspects and practical implementation constraints are discussed. This work was part of the ONERA¹ RESSAC project [FAB 07]. Figure 13.1 shows the autonomous helicopter exploring an initially unknown environment cluttered with artificial cardboard obstacles. The following functions are performed autonomously on-board the uninhabited rotorcraft: terrain exploration and mapping (perception), online optimization of a conditional exploration strategy (decision), acting according to this strategy, characterization and choice of landing zones (and autonomous landing and take-off). The human operator only intervenes for final validation before landing, or security management during the experimental flights.

13.1. Introduction

The operation of uninhabited systems in partially known dynamic environments requires *anytime* decision and reaction processes in order to enable the system to

Chapter written by Patrick FABIANI and Florent TEICHTEIL-KÖNIGSBUCH.

1. Office National d'Études et de Recherches Aérospatiales: <http://www.onera.fr/english.php>.



Figure 13.1. Autonomous uninhabited helicopter using MDP-based decision-making algorithms for the optimization of its exploration strategy in search for possible landing zones

deal timely and properly with encountered situations. A number of attempts to solve this problem are related to reactive deterministic planning [CHA 05, DAM 05]: a sequential action plan is searched over a bounded time horizon, the length of which depends on the available time, and this action plan is improved if time permits, or replanned according to external events. However, these approaches fail to deal robustly with uncertainties (partially observable states or uncertain effects of actions) and especially tend to anticipate the future quite optimistically, predicting a deterministic evolution of state from the most probable current situation.

On the other hand, *stochastic planning* approaches, like MDPs, do take into account the uncertainties of the environment in the planning process. Their output is a *policy* or a *conditional plan*, i.e. a mapping from every possible states to an optimal action to perform, which optimizes the average of cumulated rewards (or penalties) along the mission's stochastic trajectories. Such a policy is robust to uncertainties of the environment, but the complete optimization process is costly both in terms of computation time and memory. In order to overcome these difficulties, the state space can be structured (see Chapter 4) and algorithms have been developed that use heuristics to guide the solution search (see Chapter 5 and also [FEN 02, TEI 05a]).

However, these approaches are still based essentially on offline optimization processes that are not naturally fitted for implementation on real robotic systems.

In this chapter, we present both an embedded control architecture and an *anytime stochastic planning* algorithm, compromising the robustness and optimality of heuristic probabilistic planning with the adaptivity of reactive deterministic replanning.

The decision-making architecture makes use of the on-board computer operating system's threads in order to introduce parallelism between the execution of the current best policy and the further optimization of this current best policy.

The decision-making algorithm incrementally extends the sub-space of reachable states. In the initial set of reachable states, an initial feasible policy is very rapidly obtained. The sub-space of reachable states is further extended using the current policy, and the current policy is subsequently optimized on this extended sub-space of reachable states, until completion. This approach applies asynchronous dynamic programming while incrementing the size of the search space.

In section 13.2, we present the mission scenario of an autonomous aircraft searching for a landing zone in an ill-known environment. Section 13.3 deals with the embedded control architecture, and specifically with the interaction between the decision-making algorithms and the other embedded control functions. In section 13.4, we present the model and the algorithm that performs the online optimization of MDP strategies by incremental and local search. Lastly, we illustrate this approach with results obtained via real robotic implementation and flight tests on-board an autonomous uninhabited helicopter.

13.2. Exploration scenario

Two remote control Yamaha Rmax helicopters have been equipped by ONERA with an on-board avionics control architecture, embedded sensors and data processing algorithms. This equipment provides the automatic perception and action capabilities in order to perform autonomous flight, navigation, exploration of ill-known

environments in search for a place where to safely land and take-off automatically (see Figure 13.1).

In the Search and Rescue scenario of the ReSSAC project (see Figure 13.2.a), the aircraft is initially given a GPS position close to the assumed position of the person to be rescued, a number of forbidden flight zones and an initial ill-known search region within which it should find a landing site.

Before flying within areas possibly cluttered with obstacles, the aircraft must perform an initial rough exploration at 50 m of height in order to perform a vision-based mapping of the region into sub-zones corresponding to big obstacles or possible landing sites (see Figure 13.2.b). Further exploration at 20 m of height, avoiding obstacles, makes it possible to characterize more precisely the obstacles and confirm potential landing sites. However, fuel autonomy and flight duration must be taken into account: thus the anytime stochastic planning algorithm provides an optimized exploration strategy giving the sub-zones to explore, the perception and motion actions to be performed, including landing, take-off and return to the base.

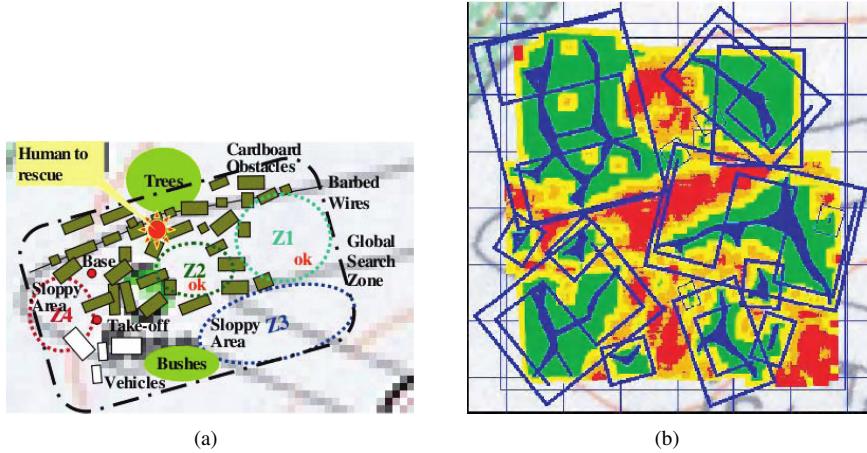


Figure 13.2. (a) Scenario – (b) Vision-based extracted sub-zones
(dark=obstructed, light=free)

13.2.1. Planning problem

The planning problem occurs after the initial rough mapping of the region, which provides a list of sub-zones, each associated with its respective name, 2D reference coordinates, 2D dimensions, number of itinerary waypoints for further characterization and probability of being appropriate for landing (based on the initial rough mapping):

```
(zones (Z1 3153.65 -1348.34 30.9898 56.726 0.731584 6) ...)
```

This list is automatically turned into an exploration planning problem written using PPDDL (*Probabilistic Planning Domain Definition Language*, see Chapter 15) [YOU 04], where states, preconditions and action effects of the *exploration planning domain* are described in first-order logic formalism. The main interest of the PPDDL formalization lies in its *intensional representation* of states and actions: it makes it possible to use state features, or state properties, to designate and manipulate states (*intension* as opposed to *extension*), in the same way as state variables are used in vector spaces, instead of enumerating them (*in extension*). Logic formulae are assigned boolean truth values (TRUE=1; FALSE=0) and apply to features of the planning domain such as the *sub-zones to be explored* in our case, which makes it possible to give once and for all a generic description of actions `goto(zone)`, `land` and `takeoff`. Furthermore, the planning domain is written in an intuitive, compact and easily readable way.

13.2.2. States and actions

The state variables of the planning domain are:

- `human-rescued`: boolean = TRUE if the human has been rescued;
- `on-ground`: boolean = TRUE if the helicopter is on the ground;
- `explored(sub-zone)`: boolean (for each sub-zone) = TRUE if the sub-zone has been characterized at 20m height;
- `landable(sub-zone)`: boolean (for each sub-zone) = TRUE if the sub-zone has been characterized as appropriate for landing;
- `where`: integer indicating the number of the helicopter position in the initial sub-zones mapping;
- `flight-autonomy`: real number indicating the estimated remaining flight time.

The state space size is thus $2^{2(n+1)}(n+1)d$, where n is the number of subzones in the initial rough mapping, and d is the number of discretization ranges of the remaining flight time.

As a matter of fact, the MDP planning algorithms developed for this application (see section 13.4) make use only of discrete state variables. Recently improved planning algorithms make it possible to deal with hybrid state spaces, like in [GUE 04]. Anytime versions of those could be developed for real implementation.

$n + 6$ actions are to be considered:

- `goto(sub-zone)`: in flight motion to sub-zone number `sub-zone`;
- `explore`: exploration of currently flown sub-zone;

- `land`: landing on currently flown sub-zone if `landable(sub-zone)` is TRUE;
- `takeoff`: take-off from the sub-zone where the helicopter is, if `on-ground` is TRUE;
- `fail-safe`: safe return to the base for security landing if the remaining flight time is lower than 10 minutes;
- `end-mission`: put an end to the mission, either after security landing or after `human-rescued` is TRUE.

13.2.3. Uncertainties

Three sources of uncertainty have to be taken into account for the optimization of the action and perception strategy:

- probability P_a (after rough mapping) for a sub-zone to be later confirmed as appropriate for landing after characterization (see Figure 13.2.b):

$$P_a = \frac{\text{number of light pixels}}{\text{number of dark pixels}} \quad [\text{pixels after image processing}];$$

- probability P_s of successfully rescuing the human if the helicopter lands at a distance d_z from the human position in sub-zone z :

$$P_s = \frac{40}{40 + d_z};$$

- probability density f_τ for an action to last τ seconds (μ_a and σ_a depend on the action a):

$$f_\tau = \frac{1}{\sigma_a \sqrt{2\pi}} e^{-\frac{(\tau - \mu_a)^2}{2\sigma_a^2}}.$$

13.2.4. Optimization criterion

An additive criterion is optimized: the mathematical expectation of the additive sum of all rewards or penalties received after the performed actions. A +1000 reward is associated with the rescue of the human (once `human-rescued` is TRUE and the helicopter has landed safely). A penalty of -1000 corresponds to the return of the helicopter to its base without rescuing the human.

The action strategy, and in particular the order in which sub-zones are selected for characterization by the optimized exploration strategy, reflects a decision taking into account the chances for each sub-zone to be eventually appropriate for landing, the chances of rescuing the human by landing in this sub-zone, and the chances of flying back home safely from this sub-zone.

13.2.5. Formalization of the decision problem

The planning problem above can be formally modeled as an FMDP (see Chapter 4) with:

- stochastic Markovian transition model;
- rewards corresponding to the effects of actions;
- additive optimization criterion;
- state space factorized according to state variables.

Furthermore, the PPDDL formalized planning problem can automatically be translated into a DBN, as indicated in [YOU 04]. The obtained DBNs are then encoded as ADDs for better processing efficiency (see section 4.3.2). However, the classical optimization algorithms for FMDPs which use decision diagrams do not guarantee the “*anytime*” behavior which is required by the application on-board a flying robot. For this reason, we show in the remaining of the chapter an algorithmic framework allowing such an “*anytime*” implementation using local and heuristic optimization for FMDPs.

13.3. Embedded control and decision architecture

13.3.1. Global view

As shown in Figure 13.3, our embedded architecture is divided into two parts: the deliberative layer and the reactive layer. Each layer runs on a specific computer. Both layers essentially interact with each other via data exchanges.

The flight control functions are executed on the reactive layer under real-time constraints: they have been validated separately and need to be able to run independently from the deliberative layer in order to permanently ensure the security of the flight – this is a requirement in order to obtain the official flight authorizations.

The deliberative layer can thus be allowed to consume more memory and computing time resources, without penalizing the real-time functions that are vital for maintaining the aircraft in flight. In other words, the main difference between the two layers, and the reason why it is more adapted to separate them, is that the deliberative and reactive layers do not have the same time constraints, which is highly desirable in order to allow the implementation of effective embedded decision-making capabilities.

The deliberative layer is composed of three main functions allowing to perform the classical *perception-decision-action* cycle:

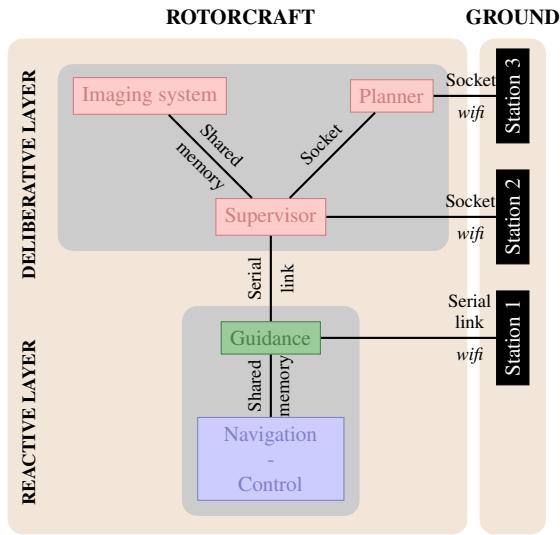


Figure 13.3. *Embedded architecture: the mission is supervised by the supervisor, which is a client of the imaging system and the planner (servers activated on request)*

– *vision*: performs the processing of the acquired images and provides the sub-zones and their characterization;

– *planning*: optimizes the solution strategy of the FMDP whose formal model is built on the basis of the output of the vision;

– *supervisor*: executes state automata allowing the supervision of the mission phases, performs the coordination between the vision and planning functions and sends control orders to be executed by the reactive layer.

The deliberative layer must, however, be reactive enough so as to produce decisions within reasonable time compared to the completion time of an action by the UAV. Otherwise, decisions might arrive with such delays that it could lead to dangerous situations. Furthermore, real autonomous rotorcraft missions are strongly constrained by the flight time, so that it is impossible to wait for the planning function to optimize the complete mission before launching the first action.

Even if security and flight control do not directly rely on it, it is still highly desirable that whatever the current UAV situation, a current best safe action is made available by the deliberative layer for execution by the reactive layer.

The decision and control architecture is thus in charge of providing the appropriate framework for an “*anytime*” behavior of the deliberative layer, which can be guaranteed by the supervisor. Therefore, both the vision and planning functions

are activated by the supervisor. Image processing and planning can thus be performed in parallel. If useful, new planning problems could be automatically generated based on the further exploration and characterization of each sub-zone, in order to optimize more precisely the helicopter's actions within this sub-zone.

13.3.2. Multi-thread planning triggered by the supervisor

The interactions between the planning function and the supervisor is represented in Figure 13.4. The output from the vision function is the mapping of the initial exploration region into sub-zones candidate for landing. Upon reception of it, the supervisor automatically elaborates the stochastic planning problem, triggers the planning server and sends it the problem to solve. The planning server then launches two parallel tasks: one is the optimization of the action policy and the other one is the dialogue with the supervisor.

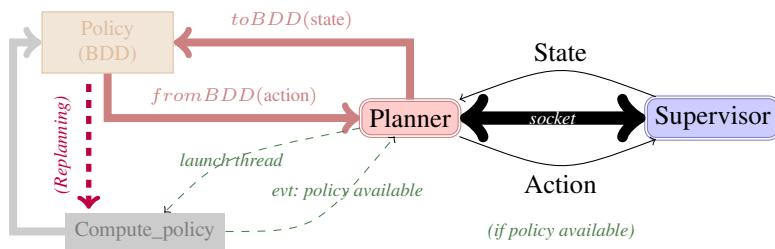


Figure 13.4. Decision and control architecture: the planning server is a multi-thread service triggered by the supervisor

13.3.2.1. Policy optimization

The policy is incrementally generated, as presented in the next section, through successive optimization iterations performed on an incrementally expanded sub-space of the complete state space. Each iteration leads to locally improving the current policy, its value being increased within a sub-space of reachable states comprising the current state and the goal states (i.e. the states providing a positive reward when they are reached). During an iteration, the current policy and the corresponding sub-space of reachable states are stored in a safe memory location, protected by mutex against accidental reading or writing. This makes it possible to optimize the current policy while being able to apply it between two successive iterations.

An applicable policy is available from the end of the first optimization iteration: the supervisor executes the first action without waiting for the algorithm to fulfil the optimization process. The behavior of the planning function is “*anytime*” provided that, as explained later on, the computation time to obtain the first iteration is small enough.

13.3.2.2. Dialogue with the supervisor

The dialogue with the supervisor is fitted to the desired “*anytime*” behavior of the deliberative layer and, therefore, to the implemented multi-thread stochastic planning approach. Complete policies are mappings from states to actions. The policy optimization iterations use BDD and ADD representations [YOU 04] that are inapplicable as such by the supervisor. On the contrary, whenever the supervisor sends the current state to the planning function, the current best action is sent back through the dialogue socket. It is obtained from the currently stored best policy if the current state stays within the sub-space of reachable states corresponding to this current policy. It can be locally optimized starting from the new current state of the UAV otherwise, thus interrupting the current policy optimization iteration, as presented in the following sections.

For instance, `landable (Z0)` can be sent by the supervisor as a request in XML format messages, useful parameters being coded in ASCII characters, and `land` can be returned as an answer, to be executed by the supervisor. The planning function converts ASCII state descriptions into their decision diagram representations, matches it with the currently stored best policy (not forgetting to set properly the reading or writing mutex) and then converts the action decision diagram into its ASCII descriptions for answering.

13.4. Incremental stochastic dynamic programming

As stated above, the desired “*anytime*” behavior of the deliberative layer relies on the one hand on the dialogue between the supervisor, the multi-thread planning function and the reactive layer, but also on the other hand on the capability of the planning function to output a best current safe action quickly enough. For the robotic application on ReSSAC helicopters, safe operation conditions require the online “*anytime*” planning function to be able to produce a decision within a time comparable to the execution time of a UAV action (flight or perception maneuver): that is to say an average 50 seconds in this case.

This section describes the algorithmic framework that makes it possible to achieve MDP optimizations under these constraints. The approach implemented for this application is based on the incremental algorithm `sFDP` (*Stochastic Focused Dynamic Programming*) [TEI 05a, TEI 05b]. Other incremental algorithms could have been used such as `sLAO*` [FEN 02], or `sRTDP` [FEN 03], which also perform two-stage optimization iterations, with a reachable states sub-space incremental expansion stage followed by a local optimization stage.

`sFDP` originally performs the incremental expansion stage of the reachable states sub-space knowing the current UAV state and a goal condition to be satisfied in a goal state (in the end):

$$at(base) \wedge (human_rescued \vee (flight_autonomy \leq 10\text{ mn})).$$

s_fDP then alternates two stages in each optimization iteration:

- The first stage is a computation of the set of states that are reachable by iteratively applying the current policy starting from the current state until reaching the specified goal states.
- The second stage runs a local optimization (Bellman update) of the policy within the obtained set of current reachable states.

Between two optimization iterations, the size of the sub-space of reachable states is incremented.

Furthermore, once an initial policy is obtained, the current best policy can be executed at any time upon request by the supervisor.

13.4.1. Obtaining the initial safe policy quickly

In order to obtain rapidly enough an initial applicable policy, a policy is calculated, without any optimization but such that there exists at least one trajectory leading from the initial state to a goal state. This policy can be obtained using an iterative construction scheme that is very close in nature to a value iteration dynamic programming scheme applied to sets of eligible actions and expanded along state transitions of non-zero probability:

$$\pi_0(s) = \begin{cases} \{\text{actions defined in goal states}\} & \text{if } s \in \text{goal}, \\ \emptyset & \text{otherwise,} \end{cases}$$

$$\pi_{t+1}(s) = \{a : \exists s', T(s' | a, s) > 0 \text{ and } \pi_t(s') \neq \emptyset\},$$

where $\pi_t(s)$ is the set of actions a that can be randomly chosen at step t in state s (eventually empty), and $T(s' | s, a)$ is the probability of transition from state s to state s' by applying action a in state s .

The second line of equation of the above system guarantees that the strategy follows a sort of shortest path between the current state and the goal states in the sense that, in each explored state, an action corresponding to a trajectory with fewer steps from the current state is preferred.

The dynamic programming process stops when there exists a step t such that $\pi_t(\text{initial state}) \neq \emptyset$. This first policy computation iteration only involves logical tests, so that it makes it possible to obtain very quickly a first applicable action in any current state.

13.4.2. Generating the sub-space of reachable states

Knowing the current policy π , the sub-space of reachable states \mathcal{F} is calculated in two phases: a forward propagation phase and a backward one (see Figure 13.5).

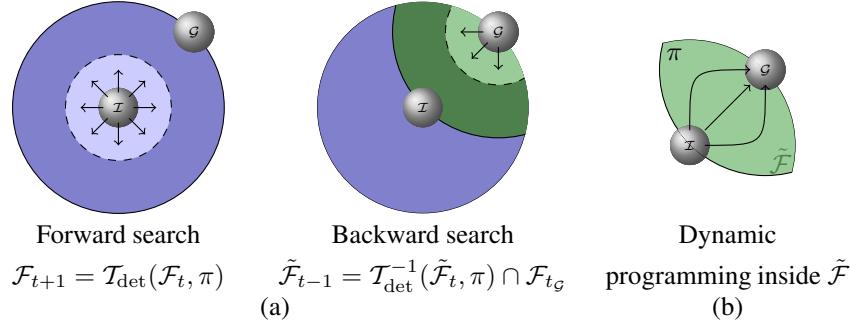


Figure 13.5. SFDP: (a) expansion of reachable states sub-space \mathcal{F} following the current policy π from the initial states \mathcal{I} until goal states in \mathcal{G} ; (b) optimization of π . (\mathcal{T} : transitions)

– Forward propagation: the following recursive system of equations, encoded as BDDs, propagates forward the set \mathcal{F} of states reachable from the set of current (or initial) states \mathcal{I} until there exists at least one reachable state in \mathcal{F} such that the goal condition \mathcal{G} is satisfied:

$$\mathcal{F}_0 = \mathcal{I}, \quad (13.1)$$

$$\mathcal{F}_{i+1} = \mathcal{F}_i \cup \{s' : T(s' | \pi(s), s) > 0, s \in \mathcal{F}_i\}. \quad (13.2)$$

- Backward propagation: the following recursive system of equations, encoded as BDDs, propagates backward the constraint that goal states from \mathcal{G} are reachable from states within the set \mathcal{F} , and thus it eliminates states in \mathcal{F} until there exists at least one state in the set of current (or initial) states \mathcal{I} such that a goal state \mathcal{G} is reachable from \mathcal{I} .

$$\tilde{\mathcal{F}}_0 = \mathcal{G} \cap \mathcal{F}, \quad (13.3)$$

$$\tilde{\mathcal{F}}_{i+1} = \tilde{\mathcal{F}}_i \cup \left(\{s : T(s' \mid \pi(s), s) > 0, s' \in \tilde{\mathcal{F}}_i\} \cap \mathcal{F} \right). \quad (13.4)$$

The reduced sub-space of reachable states $\tilde{\mathcal{F}}$ replaces \mathcal{F} : $\mathcal{F} \leftarrow \tilde{\mathcal{F}}$.

13.4.3. Local policy optimization

Once the sub-space of reachable states \mathcal{F} is calculated, the current policy π is updated using the Bellman equations restricted to \mathcal{F} :

$$V_0(s) = 0, \quad (13.5)$$

$$V_{i+1}(s) = \mathbf{1}_{\mathcal{F}}(s) \cdot \max_a \left\{ \sum_{s'} T(s' \mid a, s) \cdot (\gamma V_i(s') + R(s' \mid a, s)) \right\}. \quad (13.6)$$

The new current policy, restricted to \mathcal{F} , is obtained by applying the Bellman operator within \mathcal{F} :

$$\pi(s) = \mathbf{1}_{\mathcal{F}}(s) \cdot \operatorname{argmax}_a \left\{ \sum_{s'} T(s' | a, s) \cdot (\gamma V^*(s') + R(s' | a, s)) \right\}. \quad (13.7)$$

This new policy replaces the previous one and is stored in the mutex protected memory for application by the supervisor while the planning function proceeds with another incremental two-stage optimization iteration, starting with the reachable states sub-space expansion.

13.4.4. Launching local replanning processes

The advantage of sfDP over other incremental methods is the knowledge of goal states, which reduces the number of states to explore during the local optimization stage. On the other hand, its drawback is the loss of optimality guarantee because not all states reachable from the initial state are explored, but only those leading to goal states. In particular, there is no guarantee that all trajectories of the current policy lead to the goal states, starting at the initial state: the states subspace expansion stops indeed as soon as at least one of these trajectories is found. As a result, the planner needs to launch replanning processes in two cases (see Figure 13.6):

- (a) a low-probability state transition occurs during the policy execution, so that the new current state is outside the reachable states subspace;
- (b) no transitions are defined in the current state, when all stochastic outcomes are outside the reachable subspace.

The replanning time generally decreases at each replanning, since the new initial states of each replanning process are nearer and nearer to goal states.

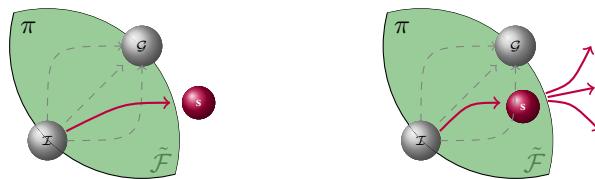


Figure 13.6. Replanning cases: (a) the current state is outside the current reachable subspace; (b) the local policy is not defined in the current state

13.5. Flight tests and return on experience

Real-time flight tests of the implemented decision architecture and continuous planning algorithms were conducted at Esperce in Southern France. A two-day-long,

multi-scenario public demonstration of the embedded artificial perception and autonomous MDP-based online replanning was also achieved in July 2007. The on-board planner implements the multi-thread version of sfDP with the initial stochastic policy computation.

Further performance tests of the decision architecture were also conducted by replaying the mission in real-time in hardware-in-the-loop simulations, but with the helicopter “on the ground”. This was done by using previously recorded vision and mapping data as inputs to the planner. The mission replanning phase is performed with the on-board processor (hardware-in-the-loop) using the embedded real-time implementation of the decision architecture and algorithms.

In this section, we compare the performance of our anytime multi-thread framework with a single-thread implementation using sfDP. We focus on four comparison criteria:

- *total optimization time*: sum of the optimization times for both the initial planning and all the replanning processes;
- *number of local replanning sequences*;
- *maximum answering time*: maximum time to send an action to the supervisor after reception of the current state;
- *landing zone*, obtained by applying the current policy from the rotorcraft’s base (the relevance of the obtained landing zone).

In order to compare these criteria on the same basis, we recorded a list of 10 zones extracted by image processing during one of our flight tests. The resulting planning problem contains 24 state variables: 1 binary variable `human-rescued`, 1 binary variable `on-ground`, 10 binary variables `explored(zone)`, 10 binary variables `landable(zone)`, 1 11-ary variable `at(zone)` or `at(base)`, and 1 288-ary variable `flight-autonomy` (1 hour divided into 288 intervals of 12.5 seconds). The size of the state space is this: $2 \times 2 \times 2^{10} \times 2^{10} \times 11 \times 288 = 13\,287\,555\,072$ states.

The aim of the tests presented in this section is not to assess the efficiency of local heuristic stochastic planning algorithms like sfDP (see [TEI 05b, FEN 03] for such tests). Our goal is rather to demonstrate that our multi-thread anytime approach produces applicable and relevant policies in a very short time.

Table 13.1 represents the comparison between the single-thread version of sfDP and its multi-thread version. All actions are durative so that we can really run the optimization in multi-thread mode and the replanning processes as background batch tasks. UAV actions last 50 seconds on average. All assessments were run on the embedded image processing and decision processor (1 Ghz Pentium) dedicated to

No. of zones	5	5	5	7	7	10	10
Algorithm	Optimal	ST	MT	ST	MT	ST	MT
Total optim. time	1358	2.58	2.8	13.76	13.6	308.29	258.57
No. replannings	0	0	1	0	1	3	4
Max. answer. time	1358	2.58	0.21	13.76	0.29	308.22	5.75
Landing zone	Z1	Z1	Z0	Z5	Z0	Z0	Z1

Table 13.1. Comparison between single-thread sfDP (ST) and multi-thread sfDP (MT) – time is given in seconds

deliberative processes on-board the rotorcraft. The last column reproduces the results obtained on-board the helicopter during one of our real flights.

The second column corresponds to an optimal algorithm (based on the SPUDD library: value iteration using BDDs and ADDs) which performs a complete Bellman optimization over the entire state space. This is not meant to provide a comparison between our implementation and a so-called “brute force” optimization algorithm (in the sense that SPUDD is not using any heuristic), but rather:

- firstly: so as to compare the relevance of the landing zone calculated by sfDP to the optimal one obtained by this optimal algorithm (landing zones are in fact identical);
- secondly: in order to give an order of magnitude of the complexity of the optimization problem, in the sense that applying such a complete and optimal algorithm, when there exist more than 5 sub-zones to explore, leads to more than 1 hour of optimization time, which is the maximum mission’s duration due to the fuel autonomy constraint.

Figure 13.1 shows that the maximum answering time of the multi-thread version of sfDP is negligible in comparison with actions’ average duration (~ 50 s).

The maximum answering time of the single-thread version is significantly higher. With 10 zones, in single-thread mode, the current state is nearly never included in the reachable subspace obtained during the successive replanning sequences because the state space variable `flight_autonomy` decreases exactly as much as the replanning time. As a result, a lot of replanning sequences are necessary before being able to apply a single action, which does not occur with the multi-thread anytime implementation.

Another way to produce anytime strategies could have consisted in splitting an incremental optimization algorithm into small *independent* computation processes, successively run within a single thread, but the implementation of this solution was avoided on the existing architecture to avoid an excessive amount of data transmissions between the different processes.

13.6. Conclusion

In this chapter, we have presented an application of MDP planning algorithms for real-time decision on-board autonomous aerial vehicles. We have also presented a multi-thread decisional architecture for real-time planning under uncertainty. We have explained that under the real-time constraints of this application, the principles of MDP optimization algorithms cannot be used and implemented “as is” for the online optimization of action strategies: the computation time would exceed the acceptable answering time limit for the planner. Yet, it was possible to show that this difficulty could be overcome by adapting the algorithms and the implementation, thus embedding them in an anytime framework for solving real-time stochastic planning problems. The planning process is divided into two communicating threads synchronized on the current applicable policy. The first stochastic policy is calculated quickly enough, such that there exists at least one trajectory leading from the initial state to goal states.

This application was achieved with the complete implementation of the optimization algorithms and the real-time decision architecture on-board the autonomous experimental aerial platforms developed at ONERA. A two-day-long, multi-scenario public demonstration of the embedded artificial perception and autonomous MDP-based online replanning using these autonomous aerial systems was also achieved. Further flight and ground performance tests were conducted, thus providing results obtained under the real-time constraints imposed by the flight control architecture. Those tests results, on-board the rotorcraft embedded processor, were presented in this chapter and show the effectiveness of our approach, in the sense that the answering time is significantly reduced, and optimized policies can be applied in time compatible with actions duration. Further research efforts need to be dedicated to improve tools for the integration and validation of the perception-decision-action loop for autonomous systems within their environment. This includes perception planning, interlaced continuous planning and action execution control, real-time perception, planning for cooperation of multiple heterogeneous assets, mixed initiative planning and automatic generation of safe discrete controllers that are valid by design.

Acknowledgment

This work would not have been possible without the great and essential work of the ReSSAC team project: V. Fuertes, G. Le Besnerais, R. Mampey and A. Piquereau.

13.7. Bibliography

- [CHA 05] CHANTHERY E., BARBIER M. and FARGES J.-L., “Planning algorithms for autonomous aerial vehicle”, *16th IFAC World Congress*, Prague, Czech Republic, 2005.

- [DAM 05] DAMIANI S., VERFAILLIE G. and CHARMEAU M.-C., “A continuous anytime planning module for an autonomous earth watching satellite”, *ICAPS'05 Workshop on Planning and Scheduling for Autonomous Systems*, pp. 19–28, 2005.
- [FAB 07] FABIANI P., FUERTES V., LE BESNERAIS G., MAMPEY R., PIQUEREAU A. and TEICHTEIL F., “The RESSAC autonomous helicopter: Flying in a non-cooperative uncertain world with embedded vision and decision making”, *A.H.S. Forum*, 2007.
- [FEN 02] FENG Z. and HANSEN E., “Symbolic heuristic search for factored Markov decision processes”, *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI'02)*, Edmonton, Alberta, Canada, pp. 455–460, 2002.
- [FEN 03] FENG Z., HANSEN E. and ZILBERSTEIN S., “Symbolic generalization for on-line planning”, *Proceedings of the 19th Annual Conference on Uncertainty in Artificial Intelligence (UAI'03)*, Morgan Kaufmann, San Mateo, CA, pp. 209–216, 2003.
- [GUE 04] GUESTRIN C., HAUSKRECHT M. and KVETON B., “Solving factored MDPs with continuous and discrete variables”, *Proceedings of the 20th Annual Conference on Uncertainty in Artificial Intelligence (UAI'04)*, Banff, Canada, pp. 235–242, 2004.
- [TEI 05a] TEICHTEIL-KÖNIGSBUCH F., Approche symbolique et heuristique de la planification en environnement incertain, PhD thesis, Ecole Nationale Supérieure de l’Aéronautique et de l’Espace, 2005.
- [TEI 05b] TEICHTEIL-KÖNIGSBUCH F. and FABIANI P., “Symbolic heuristic policy iteration algorithms for structured decision-theoretic exploration problems”, *ICAPS Workshop on Planning under Uncertainty for Autonomous Systems*, 2005.
- [YOU 04] YOUNES H. L. S. and LITTMAN M. L., PPDDL1.0: an extension to PDDL for expressing planning domains with probabilistic effects, Report no. CMU-CS-04-167, Carnegie Mellon University, 2004.

Chapter 14

Resource Consumption Control for an Autonomous Robot

This chapter presents a robotic application. The objective is to control a robot's resource consumption using a Markov Decision Process. An autonomous rover has to explore a number of spots in order to gather information. The robot's environment is inaccessible for human beings. In this context, embedded consumable resources like memory, power or time are bounded. At a certain time, a mission is sent to the rover, which can gather different pieces of information in the environment. The rover must do its best to gather as much information as it can before having consumed the resources. All tasks must be performed in several steps, but there are various ways to perform them. Generally, there are not enough resources to complete the whole mission. This chapter describes a high-level controller that selects the best way to consume resources while performing the tasks during the mission as well as possible. This selection takes the importance of each task into account but also the resources required by the rover. The resource consumption control is the motivating point of the chapter. [BRE 02, NUL 05] also propose approaches to plan a mission while taking consumable resources into account, but, in contrast with their work, we do not present a planner or a task organizer, just a resource controller. Additionally, this chapter presents the case of a single consumable resource (the remaining time), but it is possible to deal with multiple resources, as presented in [LEG 08].

Chapter written by Simon LE GLOANNEC and Abdel-Illah MOUADDIB.

14.1. The rover's mission

The rover's objective is to gather information in an inaccessible area. This area could be a remote planet, the ocean depth or rubble after an earthquake. The rover has many sensors in order to collect various types of information (e.g. cameras, infrared sensors, temperature sensors, etc.). Once the mission is over, the rover sends the data to human operators who finally analyze the collected information. At the beginning, the rover has lots of tasks to accomplish but few resources.

The mission is divided into a number of independent tasks. The exploration path is known before the mission starts. Figure 14.1 is a mission example. Here, the robot has to get atmospheric measures, to gather minerals and to take pictures of the environment.

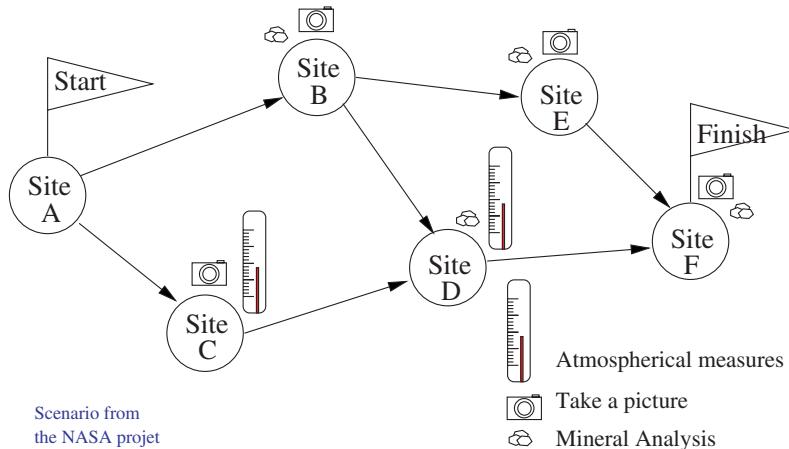


Figure 14.1. The rover has to explore some spots (graph nodes) in the mission (the acyclic graph). The mission ends if the Finish node or a predefined time deadline are reached. The rover has to stay more or less time in each node to gather information. It has to adapt the time spent in each node depending on the current node relevance

Each operation can last more or less time, i.e. it can consume more or less of the time resource. The rover may have several effectors to perform the same task, or these effectors may work in various modes. There are therefore many ways to perform a given task. For example, the rover can take a low or a high resolution picture. As the embedded resources are limited, the robot has to select an appropriate method to save resources for the remaining tasks.

The task structure is the second motivating point of this chapter. An information gathering task is divided into a series of steps (referred to as “levels”). Each level may be accomplished using one of several atomic tasks (referred to as “modules”),

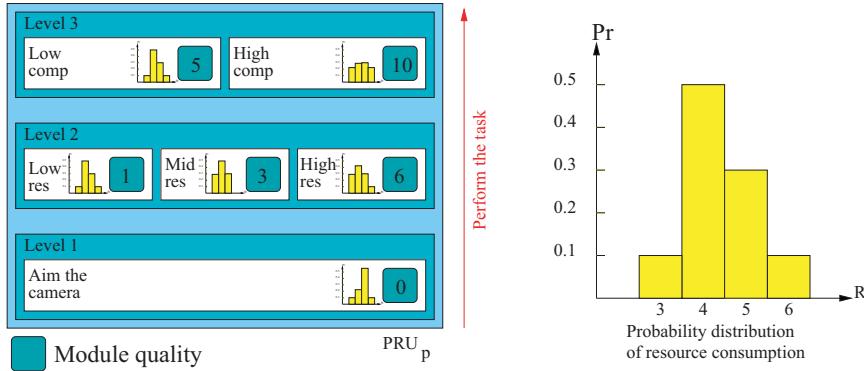


Figure 14.2. The rover has to collect information. In the figure, the left side represents a task model. The rover has here to take a picture and to save it. The task is divided into three levels: first it turns the camera towards the desired subject; second it chooses the picture resolution and finally it picks the file format to save the picture. At each level, the rover can only select and perform one module (or atomic task)

each having different specificities (resource consumption, quality). High level tasks are independent (except for the resources they have to share) but the task levels are dependent on each other: the rover has to finish the first step of a task before moving to the next step in this task. Figure 14.2 illustrates a “take a picture” task. We assume that the task descriptions are known to the rover.

On a given site, the rover can perform one or several tasks. When performing a hierarchical task step-by-step, after each step the rover can decide whether to keep on performing this task. If it realizes that it has spent too much time on a task, it can also skip the remaining levels and move on to another task. This task interruption may save resources, but it is impossible to go back to this task later.

The controller makes it possible to reach a compromise between the resource consumption and the global quality of the mission. The tasks are performed progressively, step by step. We talk about progressive processing, and a task is called a PRU (progressive processing unit). Progressive processing has been used to model exploration problems in [CAR 01, ZIL 02]. The first part of this chapter presents the progressive processing formalization. The second part explains how to control the resource consumption with a Markov decision process (MDP). We will use the hierarchical structure in order to efficiently calculate an optimal policy.

14.2. Progressive processing formalism

A *mission* is a finite sequence of tasks. These tasks are modeled with progressive processing units. Each of the P PRUs has a unique ID number p (see Figure 14.1).

We could extend the mission definition to an acyclic graph. The mission we defined is a particular path in such a graph. In Figure 14.1 the sequence A, B, E, F is a mission. For convenience, we simplify the description to minimize the number of IDs in the formalism. Results shown here are still valid for an acyclic graph.

Each PRU is divided into N *levels* noted l . After having achieved a level, the rover has to decide whether to perform the next level or to leave the task (PRU). Each level is mandatory for a task to succeed: the rover cannot save the picture without taking it.

When the robot is about to execute a level, several methods are available. In this example, it can take a low, medium or high resolution picture. A level is thus composed by several *modules*. Modules are ways to execute a level. The rover can only execute one module per level. A quality Q and a resource consumption probability distribution are part of the module's description.

We denote by $m_{p,n,m}$ the m th module m of level $N_{p,n}$ and by $Q_{p,n,m}$ the quality. The quality is a gain obtained after the module execution.

$P_{p,n,m} : R \rightarrow [0, 1]$ is the *probability distribution* over possible resource consumptions when the agent executes module $m_{p,n,m}$ (R is the resources).

$r \in R$ are consumable resources. The amount of resource is decreasing while the rover executes the mission. As an example, remaining time, battery level and available disk space are resources.

With this presentation of the progressive processing formalism, we are now able to define the MDP that controls resource consumption in this case.

14.3. MDP/PRU model

The decision-making mechanism is made of three steps:

- the MDP creation (offline);
- the policy calculation (offline);
- the mission execution (online), which consists of selecting actions according to the policy.

14.3.1. States

The agent's state consists of the current PRU, its level number and the quality Q accumulated while completing the current PRU's levels (up to the current level). Finally, the available time resource is the fourth state variable as the rover needs to know the amount of available resources to make a decision.

For example, if the rover executes PRU number 3 and has completed the second level, if there remains 153 units of time and if the accumulated quality for this PRU is 15, we denote the current state by $s = \langle 153, 15, 2, 5 \rangle$. In some cases, the agent can consume more resources than available. In such a case, the next state is the failure state.

Formally, the set of states is denoted by $\mathcal{S} = \{\langle r, Q, p, n \rangle\} \cup \{s_{failure}\}$.

Depending on the current state and on the uncertain dynamics, a level may lead to one of several states. Moreover, we add to each PRU a level 0 that corresponds to possible initial states in that PRU. In those level 0 states, the quality is set to 0 as no modules have been performed in the PRU. They only differ in their available resources.

14.3.2. Actions

Two kinds of action are available: $\mathcal{A} = \{E_m, M\}$ (see Figure 14.3):

- action E_m executes module m in the next level;
- action M interrupts the current unit execution. The rover moves to the next PRU.

After each level, the agent has to make a choice. It can go further in the current task by selecting a module in the list of modules for the next level. This execution will consume resources. The other option is for the rover to interrupt the current task and move on to the next PRU. In this case, no resources will be consumed, the next state corresponds to the 0th level in the next PRU, and the accumulated quality is set back to 0. Everything that has been done during the previous task is withdrawn.

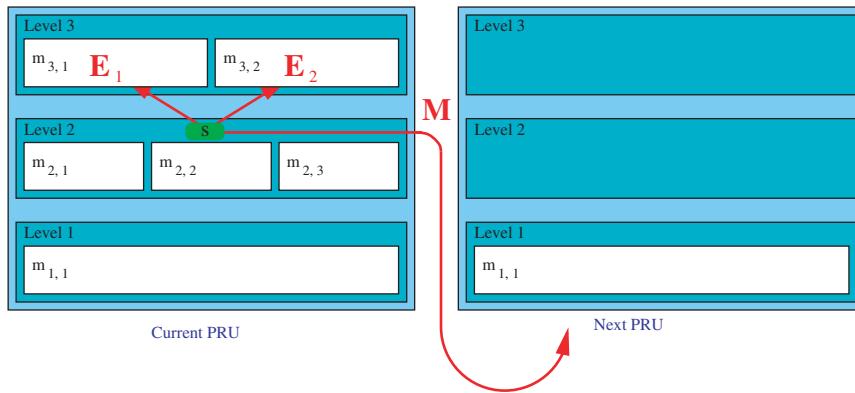


Figure 14.3. The agent is in the level 2 of the current PRU (the picture is taken). Three actions are available: 1- execute module “Save a high resolution picture” (E_1), 2- execute module “Save a low resolution picture” (E_2), and 3- leave the current task and move to the next one (M)

After having completed the last level of a task, the agent automatically moves to the next PRU.

For example, the rover can begin a picture task, but localizing the object in the environment may happen to be difficult. The rover can therefore spend too much time executing the module “aim camera”. After the level, instead of continuing the picture task, the rover can leave the current object and move to the next one. This permits to interrupt a task in order to save resources (time). The saved resources can be used for more important tasks in the future.

14.3.3. Transition function

The transition function $\mathcal{P}: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ models the uncertain dynamics.

Action \mathbf{M} is deterministic. When the rover moves to the next PRU, the amount of available resources stays the same. The new accumulated quality is 0. The new level is 0. Then,

$$\mathcal{P}(\langle r, \mathbf{Q}, p, n \rangle, \mathbf{M}, \langle r, 0, p + 1, 0 \rangle) = 1.$$

Action \mathbf{E}_m is subject to uncertainty. The probability distribution depends on the module (see Figure 14.2). Then,

$$\mathcal{P}(\langle r, \mathbf{Q}, p, n \rangle, \mathbf{E}_m, \langle r - \Delta r, \mathbf{Q} + \mathbf{Q}_{p,n,m}, p, n + 1 \rangle) = \mathcal{P}(\Delta r | m_{p,n,m}).$$

14.3.4. Reward function

The rover receives a reward when a PRU is completed. If the rover interrupts the current PRU the reward is null, no particular penalty is given for the resource consumption. Then,

$$\begin{aligned}\mathcal{R}(\langle r, \mathbf{Q}, p, N_p \rangle) &= \mathbf{Q}, \\ \mathcal{R}(\langle r, \mathbf{Q}, p, n < N_p \rangle) &= 0.\end{aligned}$$

Actually, when the robot consumes resources in a given PRU, they will naturally not be available for future use. This limits the robot’s opportunities to obtain rewards in the future. There is therefore no need to explicitly penalize resource consumption.

14.4. Policy calculation

The global control mechanism is based on an offline policy calculation for the whole mission.

14.4.1. Value function

In order to find an optimal policy, we calculate the corresponding value function.

In our setting, the Bellman update writes

$$V(\langle r, Q, p, n \rangle) = \begin{cases} 0 & \text{if } r < 0 \text{ (fail),} \\ \mathcal{R}(\langle r, Q, p, N_p \rangle) + \max(Q(s, M), \\ \quad \max_m Q(s, E_m)) & \text{otherwise;} \end{cases} \quad (14.1)$$

$$Q(\langle r, Q, p, n \rangle, M) = \begin{cases} 0 & \text{if } p = P, \\ V(\langle r, 0, p+1, 0 \rangle) & \text{otherwise;} \end{cases} \quad (14.2)$$

$$Q(\langle r, Q, p, n \rangle, E_m) = \begin{cases} 0 & \text{if } n = N_p \text{ (last level),} \\ \sum_{\Delta r} \mathcal{P}(\Delta r | m_{p,n,m}) \cdot V(\langle r', Q', p, n+1 \rangle) & \text{otherwise,} \\ \text{where } Q' = Q + Q_{p,n,m}, \text{ and } r' = r - \Delta r. \end{cases} \quad (14.3)$$

NOTE 14.1. The module's quality Q and the Q-value function $Q(s, a)$ (see Chapter 2) are two different notions.

An optimal policy is then obtained with the following equation:

$$\pi(\langle r, Q, p, n \rangle) = \operatorname{argmax}_{E_m, M} (Q(\langle r, Q, p, n \rangle, M), Q(\langle r, Q, p, n \rangle, E_m)).$$

14.4.2. Propagation algorithm

A state value only depends on its successors. All these successors are in the next levels and in the next PRUs (see equation (14.1)). The MDP is a finite-horizon acyclic graph. We can therefore calculate the whole value function using a backward chaining algorithm. To initialize the algorithm, the values of the final states – which correspond to the last level of the last PRU – are set to 0.

Finally, once the policy is calculated, the operator loads it on the robot.

14.5. How to model a real mission

The resource consumption control for an autonomous rover depends on the following assumptions:

- the mission is predefined offline;
- the PRUs are predefined correctly.

In order to model a PRU, we have to estimate the resource consumption. For now, this is achieved by gathering statistics for each module through experiments.

We illustrate the use of Progressive Processing for a mission where a robot has to knock bowling pins over to win points. We use a koala robot (K-team company) equipped with infrared sensors all around it. The pins are made of two plastic glasses and a colored ball (white or black). The middle front sensor detects whether a pin is up or not (see Figure 14.4).

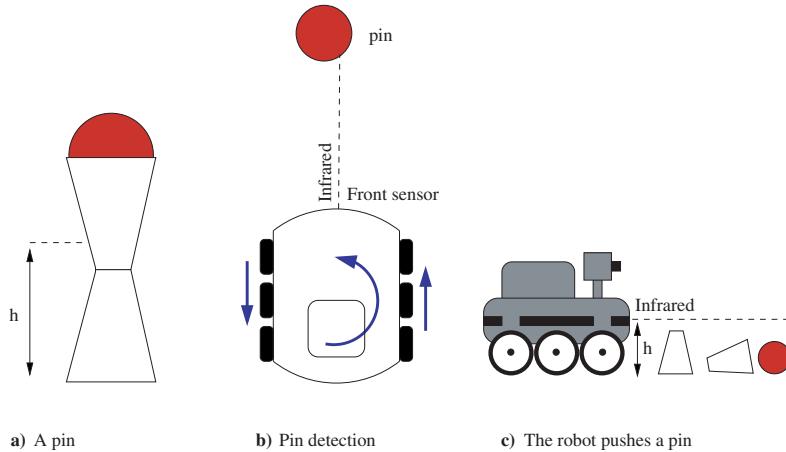


Figure 14.4. How to detect and push a bowling pin

The mission is divided into four sites. For each of them, pins are placed on an arc of a circle by a human. Thus the robot knows exactly the number of pins, their color and their disposition (e.g. here White, White, Black). But it does not know exactly where they stand on the arc. In order to detect each pin's position, the robot goes to the center of the site and turns around until the infrared sensor detects something. Once a pin has been detected, the robot chooses to push the pin or ignores it and keeps turning (see Figure 14.5). On a site, the robot turns only once. After having detected a pin, it can move forward to push the pin and backward to move back to the site's center.

Figure 14.6 illustrates the whole mission. Each arrow represents a robot's movement. When the rover knocks a bowling pin over, it receives a reward that depends on the bowling pin's color. A black one (b) is worth 100 points and a white one (w) is worth 2 points. The rover has not enough time to push all of them. So, it has to select the best ones to hit.

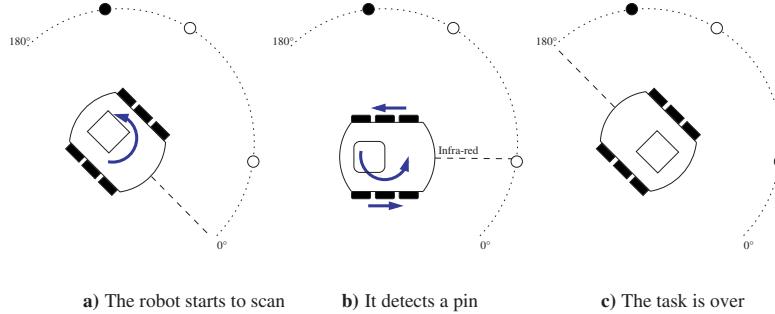


Figure 14.5. On a site, the robot turns only once. After having detected a pin, it can move forward to push the pin and backward to move back to the site's center

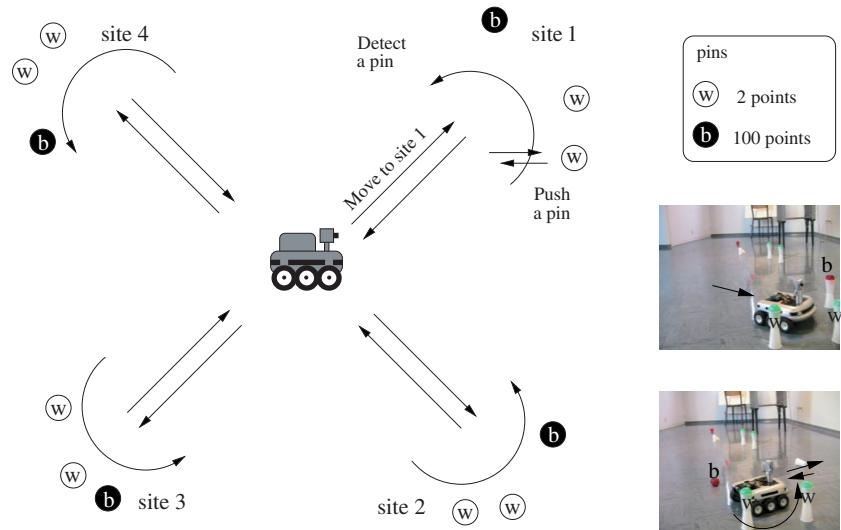


Figure 14.6. Application: a robot must hit bowling pins disposed on 4 sites

We wrote a PRU for each site that corresponds to the robot's actions (see Figure 14.6). The PRU is divided into six levels. Three modules are available, which correspond to actions detect, push and ignore. Action ignore does not consume time, although the two other actions do, as described in the figure. In this experiment, the time spent to move from the center to each site is not deducted.

The policy computation is performed offline on a computer. Then, the policy is loaded on the robot. We have conducted some experiments with the robot and this policy. The robot was given enough time to hit the four black pins, expecting that the

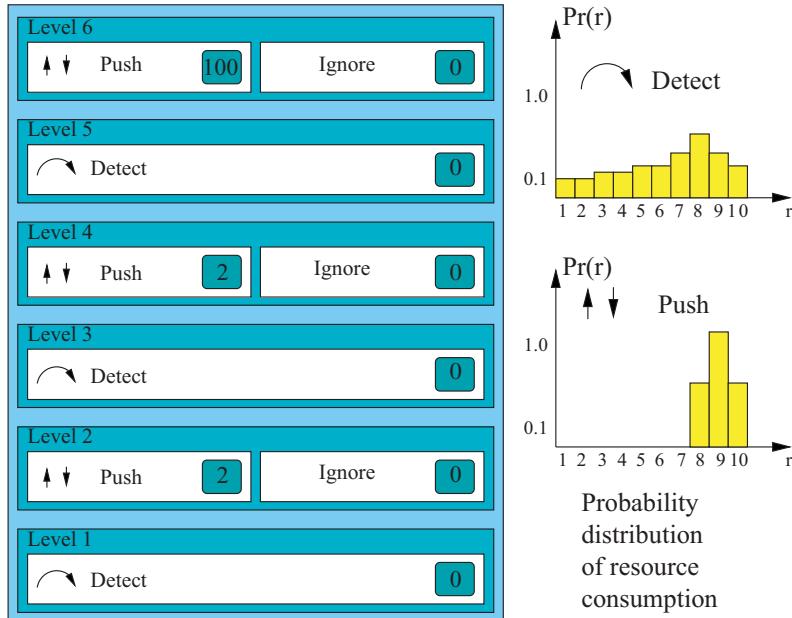


Figure 14.7. A PRU for this mission

robot would push all four black pins and score 400 points. As a result, the robot goes to each site; detects the white pins and ignore them; then it detects the black one and pushes it. Most of the time, the robot has the same behavior for each experiment. But for some experiments, the robot had enough time to push a white pin on the last site before pushing the last black pin. This implies that the robot knew that it would have enough time to push the white and the black pins in order to score 402 points.

14.6. Extensions

In [LEG 07], we propose an extension of this approach to multiple resources. As the state space grows exponentially with the number of resources, a state space aggregation mechanism is proposed so as to save memory and CPU time during the policy calculation.

The policy is calculated offline. If something happens during the mission, it is not possible to calculate another policy. Indeed, the policy calculation can take a lot of time. In [LEG 07] we also propose to reduce the policy calculation time by approximating the value function. When new tasks (PRUS) or interesting objects appear in the environment, the rover takes a decision with this approximate policy.

Experiments show that the rover takes good decisions when it uses the approximate policy.

14.7. Conclusion

In this chapter, we have presented a method to control the resource consumption of an autonomous rover. This rover must achieve a number of complex hierarchical tasks. Progressive processing is a way to model these hierarchical tasks. In order to control the consumption, we model the problem as an MDP and find an optimal policy using dynamic programming. We have presented an application for a real robot. Existing extensions of this work allow for 1- handling multiple resources and 2- approximating the policy online.

14.8. Bibliography

- [BRE 02] BRESINA J., DEARDEN R., MEULEAU N., RAMAKRISHNAN S., SMITH D. and WASHINGTON R., “Planning under continuous time and resource uncertainty: a challenge for AI”, *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence (UAI'02)*, Morgan Kaufmann, San Francisco, CA, pp. 77–84, 2002.
- [CAR 01] CARDON S., MOUADDIB A. I., ZILBERSTEIN S. and WASHINGTON R., “Adaptive control of acyclic progressive processing task structures”, *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI'01)*, Seattle, WA, pp. 701–706, 2001.
- [LEG 07] LE GLOANNEC S., Contrôle adaptatif d'un agent rationnel à ressources limitées dans un environnement dynamique et incertain, PhD thesis, University of Caen Lower Normandy, 2007.
- [LEG 08] LE GLOANNEC S., MOUADDIB A. and CHARPILLE F., “Adaptive multiple resources consumption control for an autonomous rover”, BRUYNINCKX H., PŘEUČIL L. and KULICH M., Eds., *Proceedings of the European Robotics Symposium 2008 (EUROS 2008)*, Springer, Berlin, pp. 1–11, 2008.
- [NUL 05] NULL M., BENAZERA E., BRAFMAN R., MEULEAU N. and HANSEN E. A., “Planning with continuous resources in stochastic domains”, *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI'05)*, Edinburgh, Scotland, pp. 1244–1251, 2005.
- [ZIL 02] ZILBERSTEIN S., WASHINGTON R., BERNSTEIN D. and MOUADDIB A. I., “Decision-theoretic control of planetary rovers”, BEETZ M., Ed., *Advances in Plan-Based Control of Robotic Agents*, vol. 2466 of *Lecture Notes in Computer Science*, Springer, Berlin, pp. 270–289, 2002.

Chapter 15

Operations Planning

15.1. Operations planning

This chapter deals with the application of MDPs to operations¹ planning problems. Such problems arise in areas ranging from space exploration (rovers, satellites, telescopes) to project management and military operations. Automated planning [GHA 04, REG 04] is a branch of Artificial Intelligence. It aims at building general-purpose systems capable of choosing and organizing the operations to perform, in such a way as to reach given objectives at least cost. Here, we examine complex planning problems. These not only require concurrent operation execution and the explicit modeling of time and operation durations. They also make it necessary to account for uncertainty about the effects of operations and about the time at which they occur. Such problems are known as probabilistic temporal planning problems [ABE 06, ABE 07b, LIT 05, MAU 05, MAU 07]. We define them in an intuitive way before providing a formal definition. We then focus on their modeling as an MDP and on their resolution using variants of algorithms presented in the previous chapters.

15.1.1. *Intuition*

We start with an intuitive presentation of the problem illustrated by a space exploration example. In our scenario, a Mars rover must carry out several experiments

Chapter written by Sylvie THIÉBAUX and Olivier BUFFET.

1. We use the term “operation” rather than “task” or “action”. In particular, we avoid the latter to distinguish an “action” in a planning problem from an “action” within an MDP.

on various sites and acquire scientific data which it must transmit to Earth [BRE 02] (see also Chapter 14).

15.1.1.1. Problem features

An operations planning problem is characterized by:

- an *environment/system* described by a set of state variables (discrete or continuous, also called numeric variables), such as the position and orientation of the rover, the energy and data storage available, the state of its instruments (calibrated, on, etc.);
- the *initial state* of the system;
- *operations*, under the control of the planner, which enable acting on the system; for instance, navigating from one location to another, performing an experiment, transmitting data, initializing an instrument. Several operations can be initiated at once and while others are executing; for instance, the rover might need to use certain instruments simultaneously to perform its mission. An operation can be initiated only if certain *preconditions* are satisfied. In this case, we say that the operation is *eligible*; for instance, experiments have setup conditions, such as certain instruments being turned on and calibrated. Some conditions might also need to be maintained throughout a given time interval to guarantee the correct execution of an operation; these are called *invariants*.

An operation has one or more *effects* characterized by changes of values of state variables. For instance, the effects of a navigation operation include the change in the rover's position and the amount of energy consumed. Here, we consider that these effects and the time at which they occur are governed by a probabilistic model. For instance, the effects and duration of a navigation operation depend on the precise features of the terrain, which are unknown or cannot usually be explicitly modeled; we could consider that the operation results in two possible positions with some probabilities: either the rover could or couldn't reach its destination, and in the latter case it returned to its original position. In both cases, we could consider, e.g. that the energy consumption and the action duration follow a normal distribution;

– planning *objectives*: classically, we seek to reach a goal state, described by an assignment of values to certain state variables; in our example, the goal of the rover is to have acquired and transmitted some data. Several utility criteria can be used for a solution plan. In this chapter, we will essentially consider:

- maximizing the probability of reaching the goal,
- if the goal is always reachable, minimizing the time or the resources consumed.

More complex situations could, for instance, affect different utilities to each of several subgoals; typically in our scenario, the data sets the rover must acquire have different priorities and the rover does not have the resources required to acquire them all.

Such a problem can be specified in a probabilistic variant of PDDL (*Planning Domain Definition Language*, [FOX 03, YOU 04a]), see for instance [LIT 05], or via a graphical interface such as Brazil (see Figure 15.1).

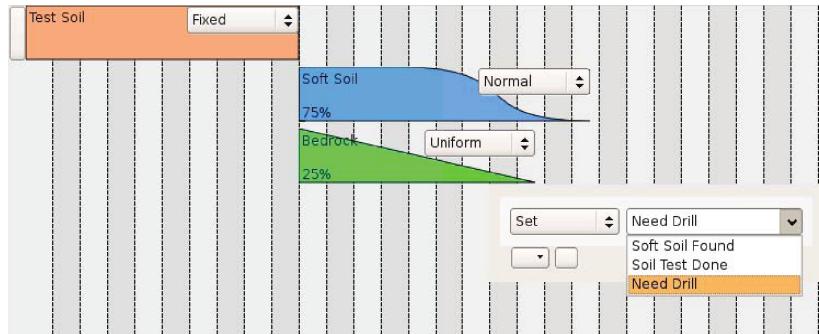


Figure 15.1. Specification, using the Brazil interface, of the operation of soil composition analysis by the Mars rover. After about 10 time units, it becomes known whether the soil is soft (soft-soil) or not (bedrock). Depending on the case, the duration follows either a normal or uniform distribution

15.1.1.2. Plans

There exist several options to deal with uncertainty, and these options determine the type of plans generated as solutions to planning problems.

- *Traditional temporal plans.*

When the risk created by uncertainty is small, the use of a deterministic planner which ignores the uncertainty in the model is common. The planner generates a mere temporal plan, representable as a Gantt chart (as in the bottom of Figure 15.2) which specifies the execution intervals of the operations chosen to reach the goal. If an unexpected event occurs at execution, *replanning* takes place from the current state. Such an approach does not guarantee the optimality of the result nor that the goal be achieved, but it enables the use of very efficient algorithms for deterministic environments. As an illustration, suppose our rover's plan is to navigate to a given site, to take a complete panorama of the landscape, and then to determine the soil composition. If the planner ignores uncertainty about the duration and energy consumption of the navigation operation, the rover might realize, following the panoramic shot, that it lacks time or energy to complete the plan execution and might need to replan or even abort the mission.

- *Robust temporal plans.* If the risk is higher, it is advisable to explicitly take uncertainty into account. If the cost of adopting a conservative attitude (i.e. the resulting loss of opportunity) is acceptable, the planner might generate *robust* temporal plans [SCH 05]. These can take the form of *conformant* plans whose correct execution is guaranteed in all circumstances or, more realistically, of temporally flexible plans

or of plans that are robust in terms of resource consumption. A robust plan for the example above might omit the panoramic shot in favor of the soil composition analysis (higher priority) if the probability of insufficient time or resources is non-negligible. Here again, this approach is not optimal.

– *Contingent temporal plans.* Finally, in general, the planner generates *contingent temporal plans* [ABE 04, ABE 07b, DEA 03, LIT 05, MAU 05]. At each time point of interest, the operations they prescribe depend on the state of the environment. For our example, a contingent plan could examine the resources available upon termination of the navigation operation, and prescribe either a complete panoramic shot, a simple forward panorama, or a skip to the soil composition analysis. Figure 15.2 illustrates the concept of a contingent temporal plan through a visualization interface. The top of the figure shows the tree² of contingencies (possible executions) covered by the plan. At the root of the tree, one or more operations are initiated. Then, the next event that occurs (typically the effect of an operation currently executing) determines the branch of the tree to be followed next. At the next node, the plan can again prescribe the initiation of new operations and so on. A leaf of the tree represents either a success situation (the goal is reached) or a failure of the plan (a situation from which the goal cannot be reached, e.g. by lack of resources). Each path through the tree corresponds to a traditional temporal plan representable as a Gantt chart such as in the bottom of the figure.

In this chapter, we focus on the generation of contingent temporal plans.

15.1.2. Formal definitions

We now formally define the notions of problem and plan, but in a framework that is not yet that of MDPs.³ We make several simplifying assumptions with respect to the intuitive presentation. Firstly, we only consider Boolean variables. Secondly, we assume that operations have preconditions but no invariant conditions which must hold throughout the operation execution. Most of the approaches we describe easily generalize to multivalued variables and invariant conditions. The third assumption is that the time points at which decisions are made to initiate operations are those at which the effect of an already executing operation might take place. This limits the combinatorial explosion. Even though work is currently being done to relax this assumption, it is made in most existing probabilistic temporal planners. The last assumption is the absence of continuous distributions. We will remove it in section 15.3.

2. For infinite horizon problems, it is best to consider a cyclic graph rather than a tree. Visualizing plans becomes difficult in the presence of uncertainty about continuous quantities such as operation durations or resource consumption.

3. We will return to MDPs in section 15.2.

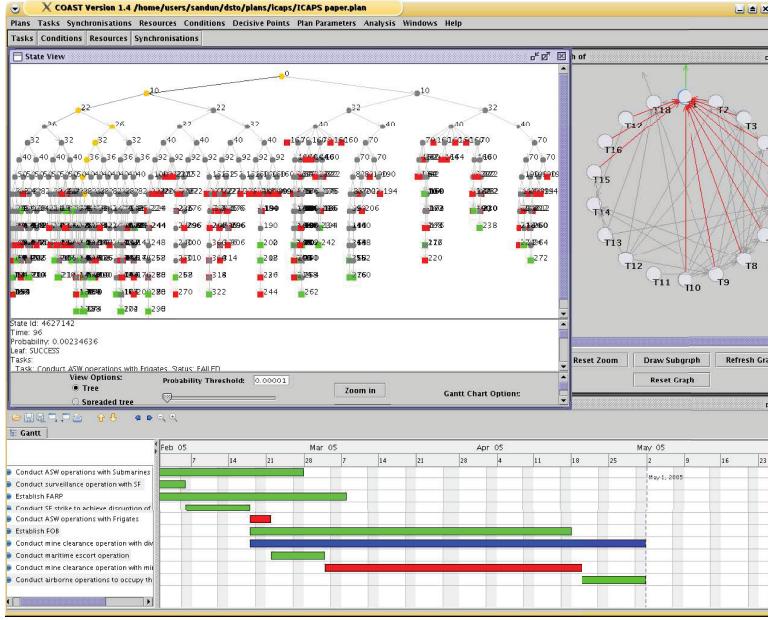


Figure 15.2. Interface for visualizing contingent temporal plans. In this particular case, the random events are end of operations which lead either to failure (left) or success (right)

15.1.2.1. Planning problem, operations

DEFINITION 15.1. An operations planning problem is defined by a four-tuple $\langle B, m_0, Op, \Phi \rangle$ where:

- $B = \{b_1, \dots, b_{|B|}\}$ is a set of Boolean variables;
- m_0 is the initial state of the system, defined by an assignment to variables of B ;
- $Op = \{op_1, \dots, op_{|Op|}\}$ is a set of operations (described below);
- Φ is a Boolean formula over B describing success situations.

This definition is incomplete, not only because the notion of operation is not described (see below), but also because it does not precisely specifies the objective of the planning problem (the optimization criterion) which is discussed later.

DEFINITION 15.2. An operation op is defined by a triple $\langle \text{Pre}(op), c(op), A(op) \rangle$:

- $\text{Pre}(op)$: a Boolean formula over B describing the preconditions of op ;
- $c(op) \in \mathbb{R}$: the cost of executing op ;
- $A(op)$: a tree defining the possible effects, described by:

- arcs which represent a temporal progression: each arc is labeled with a probability distribution P_d (over \mathbb{R}) describing the uncertain duration between a node and its successor;
- internal “conjunction” nodes (denoted by “&”): when such a node is reached, all its successors must be executed;
- internal “chance” nodes (denoted by “?”), each associated with a probability distribution $P_?$ over its successors: when such a node is reached, exactly one of its successors is chosen at random according to distribution $P_?$ and executed;
- “effect” leaves, each associated with a couple $(b, v_b) \in B \times \{0, 1\}$: when such a leaf is reached, the value of variable b becomes v_b .

When an operation is initiated, which is only possible when $\text{Pre}(op)$ is satisfied, its tree $A(op)$ is executed starting from the root. This tree describes the effects of the operation, and models the uncertainty about their occurrence and the interactions – including the time elapsed – between them. Figure 15.3 shows the tree corresponding to the soil composition analysis operation mentioned in Figure 15.1.

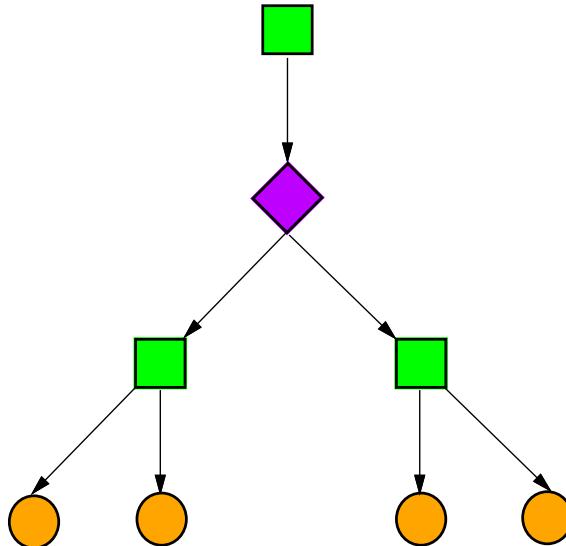


Figure 15.3. Tree specifying the effects of the soil composition analysis operation in Figure 15.1

NOTE 15.1. Except when otherwise specified, we consider, to simplify, that durations are deterministic so that the probability distribution P_d is replaced by a fixed duration d .

15.1.2.2. Execution

Several operations can execute concurrently. This execution can be managed via an event queue where each event corresponds to a node in the effect tree of an operation currently executing. Events are ranked in the queue in chronological order. The execution process looks at all events to occur at the next time step t : e_1, \dots, e_n , and partitions them into $E_{\&/?}$ which contains conjunction nodes and chance nodes, and E_e which contains leaves (effects) nodes, and then:

- 1) for each $e \in E_{\&/?}$, e is removed from $E_{\&/?}$ and:
 - if e is a conjunction node, all its successors are added to the queue or (for those that must execute immediately) to one of the two sets $E_{\&/?}$ or E_e ;
 - if e is a chance node, one of its successors is chosen at random according to the node's distribution and this successor is added either to the queue or to $E_{\&/?}$ or E_e ;
- 2) once $E_{\&/?}$ is empty, E_e is tested to detect any conflicting effects, i.e. effects that disagree on the value a variable should take; such a conflict leads to an execution failure;
- 3) if E_e is conflict free, each of its effects is executed (Boolean variables are set to the specified values).

The description of this process lacks the mention of how new operations are initiated at a given time point. An operation can only be initiated if its preconditions are satisfied. For several operations to be initiated concurrently, their immediate effects must be compatible and do not violate any of the preconditions. This concurrency model is based on the concept of *independence* [GHA 04, pages 119–120] whereby several operations can take place in parallel if and only if their execution is possible in any order, yielding the same result in all cases (here immediate result). Therefore, initiating a set of operations requires:

- 1) testing whether the preconditions of all the operations to initiate are satisfied;
- 2) adding the root of these operations' effect trees to the queue, handling immediate events as described previously;
- 3) testing again whether all preconditions above are satisfied.

We say that two operations are *mutually exclusive (mutex)* if and only if they are not independent, that is, if and only if their preconditions are incompatible or no execution of their immediate events can succeed.

Other definitions of mutual exclusion proposed in the literature account for the entire execution of the operations and forbid any potential conflict with one of the effects (not only the immediate effects) of the operations to initiate or currently executing [MAU 07]. In that case, two operations that have even a very small

probability of conflicting cannot be combined, even if that is the only way to obtain a reasonable plan.

15.1.2.3. *Decision epochs, states, plans*

In our framework, a decision can only be made at the time points at which events from the queue must be processed. Such time points are called decision epochs. At each decision epoch, one initiates a (possibly empty) set of operations, and any future initiation must wait for the next decision epoch. As mentioned earlier, this assumption is restrictive. Even in the absence of chance events, it can lead to a loss of optimality and completeness of the planner [CUS 07]. There exist, however, interesting cases where optimality is preserved, for instance, for operations whose preconditions are also invariant throughout the action execution interval and whose effects only occur at the end of the interval [MAU 06].

DEFINITION 15.3. *A state $s \in S$ is described not only by an assignment to the Boolean variables, but also by the queue of events to occur at decision epochs. The initial state is $s_0 = \langle m_0, \emptyset \rangle$.*

Time can be left out of the state description provided there is no reference to an absolute date in the problem (deadline to reach the goal, exogenous events occurring at a precise date). Consequently states that are identical up to such timing information can be aggregated to dramatically reduce the size of the state space (by introducing cycles). This is the motivation for representing plans as graphs with cycles.

DEFINITION 15.4. *A plan $\pi : S \rightarrow 2^{Op}$ is an application which maps each state encountered during its execution to a set of operations to initiate.*

The number of states that can be encountered is frequently much lower than the number of possible states. Hence, the domain of the plan depends on the plan itself, and can be constructed starting from the initial state s_0 .

15.1.2.4. *Objective*

In planning, we typically look for a plan optimizing one or more criteria. Several optimization criteria can be used and combined, using various quantities featuring in the definitions given above.

– *Success probability:* a first criterion is the *maximization of the probability of reaching a goal state* (a state satisfying Φ , and perhaps with an empty event queue), that is, the avoidance of execution failures and infinite loops.

At equal success probability, plans can be ranked using other criteria such as those described below.

– *Execution cost:* another criterion is the *minimization of the expected execution costs of the operations*.

– *Plan duration*: a final, more classical plan criterion is the *minimization of the expected duration of a plan*, which depends on the individual operation execution durations. This criterion favors the exploitation of concurrency during plan execution.

15.2. MDP value function approaches

The planning problem can be modeled as an MDP whose states are those of the problems and whose actions correspond to the initiation of operation sets. Since both the number of decision epochs and possible decisions increase exponentially with the number of eligible operations, the main difficulty is to cope with the size of the MDP obtained. For that reason, research has focused on the use and extension of heuristic search algorithms which, like (L)RTDP,⁴ perform successive trials and explore a fraction of the MDP, and on the design of appropriate heuristics. These algorithms and heuristics aim at an optimal or near-optimal solution to the MDP. This section details those three points in turn: formalization as an MDP, algorithms and heuristics.

15.2.1. Formalizations, CoMDP

15.2.1.1. States, actions, transitions

To turn an operations planning problem into an MDP, we start by identifying the *states* of the former with those of the latter; an MDP state consists therefore in an assignment to the Boolean variables, an event queue and optionally, a current time point. An *action* simply is the decision, in a given state, to initiate a (possibly empty) *set* of eligible and non-mutually exclusive operations. For this reason we often speak of a CoMDP (concurrent MDP) [MAU 04, MAU 05]. The associated *transition probabilities* can be calculated given the states and actions of the problem [ABE 04, MAU 05]. In practice, given a state and an action, this requires:

- 1) incrementing time up to the date of the next queue event;
- 2) simulating the initiation of the action as described in section 15.1.2.2, so as to enumerate all possible successor states;
- 3) computing the probability to reach each successor state which, for a given state, is the product of the probabilities of the successors of the chance nodes inserted in this state's queue; identical states reachable via several executions should be aggregated and their probabilities summed.

AND/OR graph representations are often used as an alternative to MDPs. Prottle [LIT 05] is a probabilistic temporal planning system which uses an AND/OR graph formalization of the problem which is very close to CoMDPs. As previously, the nodes

4. RTDP is described in Chapter 6, section 6.2.3.2. *Labeled RTDP* (LRTDP) is a variant that controls the convergence of the algorithm.

of the graph are the states of the planning problem. Transitions correspond to the choice of an operation to initiate at the current date, to advancing time up to the date of the next event in the queue, and to the processing of a queue event. A node is an OR node if it corresponds to the choice of an operation to initiate or to the processing of a deterministic event, and an AND node if it corresponds to the processing of a chance event. Hence, a difference with the COMDP model is that an MDP transition is decomposed in as many transitions as initiated operations. This decomposes and therefore simplifies the calculation of transition probabilities and enables the use of finer heuristics.

15.2.1.2. Rewards, costs

The reward function depends on the chosen criterion. Here we review solution algorithms appropriate for a number of criteria:

- Success probability: a reward of zero is given by default, and a unit reward is given when a success state is reached. The value of a state-action pair (s, a) is the success probability when executing action a in state s and following the optimal policy thereafter. Making success states absorbing (without reward once such a state is entered) yields an indefinite horizon MDP which can be solved by value iteration with the total criterion ($\gamma = 1$). It is additionally possible to restrict the states to be considered to those accessible since s_0 .

Rather than using a reward function, it can be advantageous to view the problem as a stochastic shortest path problem, where we seek to minimize the expected cost to reach a terminal state (success or failure). In that case, the default cost is zero, and failure states cost one. If time is included in the state, it is always possible to guarantee that a terminal state will be reached by imposing an upper bound on plan duration and by declaring any state exceeding it as a failure state. Traditional stochastic shortest path algorithms such as (L)RTDP [BAR 95, BON 03] and LAO* [HAN 01] presented in Chapter 6 can then be applied.

- Execution cost, plan duration: with those two criteria, the problem becomes a cost minimization problem. The cost $c(s, a, s')$ of a transition $(s, a) \rightarrow s'$ is in one case the sum of the costs of the operations initiated by action a and in the other case the time elapsed between s and s' . If a terminal state can be reached from any reachable state, the MDP obtained is again a stochastic shortest path problem to which the above algorithms can be applied.

- Combination of criteria: ideally such criteria should be combined, emphasizing first minimum failure probability. Risk-adverse algorithms [GEI 01], which optimize an ordered list of criteria (see Chapter 10), could be extended in this direction. A simpler alternative, but one which does not provide general guarantees, is a combination which assigns an exponentially larger weight to criteria of greater

importance: for n criteria, let $c_i(s, a, s')$ be the cost function relative to criterion i , the global cost function is $c(s, a, s') = \sum_{i=1}^n c_i(s, a, s') \alpha^i$, with α sufficiently large [ABE 04].

15.2.2. Algorithms

Value function approaches from the literature mainly belong to the framework of stochastic shortest path problems which they solve using variants of (L)RTDP and (L)AO*. As noted in Chapter 6, these algorithms are an efficient alternative to classical algorithms such as value iteration. Here, they contribute to cope with the CoMDP size explosion by constructing and exploring it on the fly, guided by an admissible heuristic.

15.2.2.1. (L)RTDP

As mentioned in Chapter 6, RTDP is an asynchronous version of value iteration, which updates states (whose values are initialized via an admissible heuristic) as frequently as they are visited by the greedy policy [BAR 95]. Therefore, RTDP focuses on the most probable states, quickly yields a good policy and totally ignores the states that are unreachable from the initial state s_0 . Convergence might be slow if the heuristic is not informative enough, because important but low-probability states are too rarely visited. LRTDP [BON 03] remedies those convergence problems by labeling the states according to whether they have converged.

15.2.2.2. Memory management

The size of CoMDP obtained for practical operations planning problems is such that it is necessary to modify (L)RTDP to increase its efficiency, at the cost of a loss of optimality and convergence. In fact, even if the majority of states is not visited, relatively benign scenarios involving a mere 20 operations already require exploring millions of states.

In particular, additional measures are required to limit the memory consumption of the algorithm. When a value $Q(s, a)$ indicates that an action is too costly, the corresponding states will not be part of the optimal solution. Such states form the majority of those kept in memory and are obsolete, even if they can occasionally be visited by the current policy. This motivates the following memory management policy: all states that are not reachable *via* the current policy, and whose recent occurrence frequency is below a certain threshold, are removed from the table that memorizes states [ABE 04]. If such a state is revisited later on, its value will need to be re-learnt.

15.2.2.3. Reduction of the number of updates

It is also important to limit computation time, which is dominated by the fact that the number of updates to be performed⁵ for each visited state is linear in the number of actions of the CoMDP and therefore exponential in the number of operations of the planning problem.

A first option is to *eliminate* some of the actions, i.e. certain combinations of concurrent operations, as soon as it can be proved that they are suboptimal. For example, combo-elimination [MAU 04] relies on the following principle: if, at a given iteration k of RTDP, the Q -value $Q_k(s, a)$ of a state-action pair – which we know will remain a lower bound on the optimal Q -value provided Q -values were initialized in an admissible way – becomes greater than a known upper bound $U(s) \geq V^*(s)$ on the optimal value of this state, then action a can be eliminated for the remaining iterations. $U(s)$ can be obtained by solving the sequential MDP obtained by forbidding concurrent operations. Mausam and Weld also describe another method called *combo-skipping*, which is less costly but only eliminates actions for a given RTDP iteration [MAU 04, MAU 07].

Another option to limit computation time is to perform a restricted *random* set of updates, following a distribution that favors the most “important” operation combinations. Mausam and Weld [MAU 04, MAU 07] use a distribution that favors (1) combinations of operations whose Q -values, as singleton actions, are the best for the given state and (2) the combinations of operations that had the best Q -value at the previous iteration.

15.2.2.4. Hybrid algorithms

Operations planning has also motivated the use of hybrid algorithms which deploy an optimal algorithm for states frequently visited and a quick but suboptimal algorithm for the other states [MAU 05]. For example, we can hybridize the RTDP algorithm operating in the CoMDP defined above with a version of RTDP operating in an “aligned” CoMDP in which a decision can only be made once all operations already initiated have terminated. Figure 15.4 illustrates the concept of aligned CoMDP. This CoMDP is much simpler than our “intertwined” CoMDP because the states do not need to maintain an event queue.

With the aligned CoMDP, RTDP quickly converges because the state space is much smaller, but it produces a suboptimal policy for the original problem. With the intertwined CoMDP, RTDP produces the optimal policy but convergence is very slow. Therefore, the hybrid algorithm uses RTDP in the intertwined CoMDP long enough to generate a “good” policy for the frequent states but interrupts RTDP well before

5. The computation of the min in the line $V(s) \leftarrow \min_{a \in A} Q(s, a)$ of the RTDP algorithm.

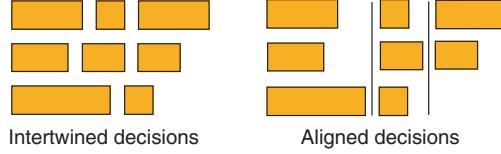


Figure 15.4. *Intertwined and Aligned Decisions. In the aligned case, decisions only happen at the time points represented by vertical lines*

it converges for less frequent states; from each state whose visit frequency under the intertwined policy is insufficient, it creates an aligned policy by letting RTDP converge on the aligned CoMDP. The hybrid policy returned consists of the intertwined policy for frequent states and in the aligned policy for the others. Moreover, to guarantee that a terminal state will always be reached, an aligned policy is also generated for each state preceding a sink state of the intertwined policy, as well as for any state belonging to a cycle of the intertwined policy.

The hybrid algorithm thus alternates k RTDP trials on the intertwined CoMDP and executions of RTDP until convergence on the aligned CoMDP for infrequent states, sinks and cycles. The alternation stops when the cost of the hybrid policy approaches that of the optimal policy. This can be implemented as follows [MAU 05, Figure 4]. Note that the estimated cost $C_i(s_0)$ of the initial state returned by RTDP in the intertwined CoMDP is always lower than the cost of the optimal policy (provided admissible heuristics are used), and that the real cost $C_h(s_0)$ of the hybrid policy (which can be evaluated by simulation) is always higher than the optimal cost. Therefore, we can stop when $\frac{C_h(s_0) - C_i(s_0)}{C_i(s_0)} < r$ for a given ratio r .

15.2.2.5. Algorithms with upper bounds

Operations planning has inspired variants of AO* and RTDP that maintain not only a lower bound $L(s)$ on the cost of each state s , but also an upper bound $U(s)$ [LIT 05]. These algorithms have become popular to accelerate convergence and provide performance guarantees [MCM 05, SMI 06].

For example, the search algorithm of the Prottle planner [LIT 05] operates via successive trials like (L)RTDP, and maintains such two bounds. In Prottle, the lower bound is initialized by heuristics based on the planning graph and discussed in the next section. The cost of a state converges when $U(s) - L(s) \leq \epsilon$. The algorithm updates the bounds and labels as having “converged” the states in a trial only at the end of the trial (a terminal state – success or failure – is reached). The formulae that update the bounds of a state given the bounds of its successors $\text{Succ}(s)$ depend on the type of bound (L - U) and on the type of state (AND-OR) considered (remember that Prottle operates on AND-OR graphs):

$$L_{OR}(s) := \max \left(L(s), \min_{s' \in \text{Succ}(s)} L(s') \right),$$

$$\begin{aligned}
U_{OR}(s) &:= \min \left(U(s), \min_{s' \in \text{Succ}(s)} U(s') \right), \\
L_{AND}(s) &:= \max \left(L(s), \sum_{s' \in \text{Succ}(s)} \Pr(s') L(s') \right), \\
U_{AND}(s) &:= \min \left(U(s), \sum_{s' \in \text{Succ}(s)} \Pr(s') U(s') \right).
\end{aligned}$$

LRTDP randomly selects the next state of the path to explore among those resulting from the execution of the greedy policy. Prottle's search algorithm selects the next state in a deterministic way, and implements a strategy that aims at generating a path to the goal as quickly as possible, and at robustifying known paths to the goal thereafter. Formally, the selected state is the successor s (among those that are not yet labeled as having converged) maximizing $\Pr(s)U(s)$, and in case of tie, the state maximizing $\Pr(s)L(s)$.

15.2.3. Heuristics

Search algorithms such as (L)RTDP and (L)AO* rely on a heuristic estimate to initialize the cost of a newly encountered state. A heuristic is a function $h(s) \geq 0$ which estimates the optimal cost $C^*(s)$ to reach a terminal state from state s . LRTDP is guaranteed to converge to the optimal policy provided that h is admissible: $h(s) \leq C^*(s)$ for all state s . We now detail some of the heuristic functions used in the framework of operations planning. The first two subsections assume that all effects happen at the end of operation execution intervals. We write $\Delta(op)$ for the duration of the operation op . All the heuristics we describe assume that the problem goal Φ is a conjunction, that is, a set of subgoals.

15.2.3.1. Basic heuristics

The MOP system [ABE 04] implements several simple admissible heuristics that can be quickly calculated, and which, respectively, estimate the plan failure probability, the total duration expectation of the plan and the expectation of the cost of its operations. The two latter are described below. Those heuristics are then combined to estimate the total cost as described in section 15.2.1.2 – via an exponential weighting scheme. In the following, we write $\text{goals}(\Phi, s)$ for the set of subgoals of Φ which are not yet satisfied in state s , and $\text{prod}(\phi)$ for the potential “producers” of ϕ , i.e. for the set of operations that have ϕ as one of their possible effects.

We start by formulating a lower bound on plan duration: the maximum of all durations required to establish each of the subgoals. The required duration for a given

subgoal is the minimum of the durations of the operations capable of establishing the subgoal. This yields

$$h_{\Delta}^{el}(s) = \max_{\phi \in \text{goals}(\Phi, s)} \min_{op \in \text{prod}(\phi)} \Delta(op) \leq C_{\Delta}^*(s).$$

This is indeed a lower bound since 1) only the cost of producing the most critical subgoal is considered, 2) interactions between subgoals, i.e. the fact that one subgoal can destroy another, are ignored, and 3) we assume that the probabilistic outcomes of operations are under control. A lower bound on plan cost (for instance, on its resource consumption) can be constructed in a similar way, by summing the minimum costs required to produce each subgoal. The exact calculation of the least-cost set of operations that can produce all subgoals is NP-hard, but an under-approximation can be constructed by dividing the cost of an operation by the number of subgoals it produces. This way, if an operation's cost is the smallest for all the subgoals it produces, the operation will contribute its exact cost to the sum. If it does not have the smallest cost for a subset of the subgoals it produces, its contribution will be less than its exact cost:

$$h_c^{el}(s) = \sum_{\phi \in \text{goals}(\Phi, s)} \min_{op \in \text{prod}(\phi)} \frac{c(op)}{|\{\phi' \in \text{goals}(\Phi, s) : op \in \text{prod}(\phi')\}|} \leq C_c^*(s).$$

15.2.3.2. Heuristics obtained by relaxation of the CoMDP

More informative heuristics can be obtained by optimal resolution of a relaxation of the initial CoMDP, that is, by optimally solving a simpler problem. Such a relaxation is easier to solve and produces a policy that is less costly than the original, and which can therefore be used as an admissible heuristic.

For instance, the DUR system [MAU 05] implements two relaxation heuristics to estimate the expected duration of the plan. The first is called “maximum concurrency” heuristic, and is based on the observation that the expected duration of the optimal policy for the sequential MDP (no concurrency allowed), divided by the maximum number of operations that can happen concurrently at any instant, is a lower bound on the expected duration of the CoMDP’s optimal policy. This heuristic can be calculated as follows. Firstly the sequential MDP is solved. Its states do not have an event queue but only represent an assignment to the Boolean variables. Its actions are the operations in the problem and the cost of a transition is that of the corresponding operation. Let $C_{seq}^*(m)$ be the cost of the optimal policy for a state m of the sequential MDP. The original CoMDP can then be solved using the heuristic:

$$h_{\Delta}^{mc}(s) = \frac{C_{seq}^*(m(s))}{maxconc} \leq C_{\Delta}^*(s),$$

where $maxconc$ is the maximum number of operations that can execute in parallel at any point, and $m(s)$ is the Boolean assignment resulting from the execution in state s of all actions in the event queue of s , in chronological order.

The second heuristic implemented in DUR is the “eager effects” heuristic. The idea is to pretend that we know all effects of the operations last initiated as early as the next decision point, without waiting for these effects to actually occur. The state of the resulting COMDP is represented by a pair (m, δ) , where m is the state of the system accounting for the eager effects, and δ is the duration until the latest termination date of an operation currently executing. Intuitively, this means that m will be reached after a duration of δ . Figure 15.5 gives an example in which, from state m of the system, 3 operations, respectively, terminating after 2, 4, and 8 time units are initiated, whose accumulated effects lead to system state m' . The resulting state of the relaxed COMDP is $(m', 6)$, because effects are known as soon as operation b terminates, i.e. after 2 time units, and there are 6 time units remaining until the latest operation (a) terminates. The cost of a transition of the relaxed COMDP represents the duration between two states and corresponds to the duration until the next action currently executing terminates.

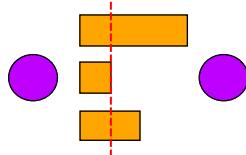


Figure 15.5. Eager effects

The resulting COMDP is indeed a relaxation because (1) it makes information available in advance of the reality and (2) since it does not keep track of the time at which the various operations terminate, it authorizes the initiation of operations whose preconditions are not necessarily satisfied. From an optimal solution of cost $C_{ee}^*(\sigma)$ for each state $\sigma = \langle m, \delta \rangle$ of the relaxed COMDP, a heuristic for the original MDP can be obtained in the following way:

$$h_{\Delta}^{ee}(s) = \sum_{m' \in 2^B} p(m' | m(s), ex(s)) C_{ee}^*(\langle m', \text{last}(s) \rangle) \leq C_{\Delta}^*(s),$$

where $ex(s)$ is the set of operations under execution in state s and $\text{last}(s)$ is the duration until the termination of the latest operation in $ex(s)$.

We refer to [MAU 06, MAU 07] for an extension of this type of heuristics and of hybrid algorithms in the case of stochastic durations.

15.2.3.3. Planning graph heuristics

The planning graph is a data structure often used to build heuristics [BRY 06]. It can be seen as an approximation of the state space of the problem. This graph originated with the deterministic a-temporal planner Graphplan [BLU 97], which uses

it to determine whether it is possible to reach a goal state from the initial state in less than n steps (each step consists of the execution of a set of non-mutex operations which all terminate before the next step).

The graph provides, in polynomial time, a necessary but sufficient condition for reachability. An extension of the planning graph provides a lower bound on the probability of failure for probabilistic temporal planning problems we consider here [LIT 05]. In our framework, the heuristic estimation consists of 3 steps: (1) building the graph, (2) associating lower bounds to nodes of the graph and (3) combining those bounds to obtain the failure probability estimate at a given state of the CoMDP.

Step 1: building the graph

The original planning graph (deterministic, a-temporal) consists of alternating layers of two types of nodes: nodes that represent propositions (Boolean variables), and nodes that represent operations. The graph is built from the operation descriptions. The successors of an operation node at a given layer are the propositions of the next layer representing the positive effects of the operation (the variables which are set to 1 by the operation), and its predecessors are the proposition nodes of the previous layer that represent its preconditions.⁶ We refer to [GHA 04, Chapter 5] for details of the construction and properties of the graph. Let us simply mention here that the graph's size is polynomial in the problem's size and that the presence of a proposition or an operation in a given layer n indicates that it is *not impossible* that the proposition be reachable or that the operation be executable after n steps.

Compared with the original planning graph, the probabilistic temporal planning graph has an additional node type: *chance* nodes. These correspond to the chance nodes in the tree defining the effects of operations. With this extension, operation nodes are now linked to chance nodes, chance nodes to proposition nodes or other chance nodes, and proposition nodes to operation nodes. The temporal aspects of the problem are handled by associating a date to each layer. The successors of a chance node are not necessarily located at the “next” layer, but at the layer with the appropriate date.

Step 2: computation of the cost of nodes

The graph is generated once and for all from the problem's initial state and up to a given temporal horizon (date). Then, for each node n of the graph, a vector of costs $c_n[i]$ is built by back propagation, which reflects the capacity of n to contribute to reaching the i th subgoal Φ_i . A cost of zero indicates that the node (possibly in combination with others) makes the subgoal inevitable, whilst a cost of 1 indicates that the node is irrelevant to reaching the subgoal. The cost vectors for the nodes in the last

6. We assume that $\text{Pre}(op)$ is a conjunction of atomic propositions.

layer of the graph, which are proposition nodes, are initialized as follows: $c_n[i] = 0$ if $n = \Phi_i$ and $c_n[i] = 1$ otherwise. These costs are then propagates backwards, according to rules that guarantee admissibility and depend on the type of node, chance (h), operation (o) or proposition (p):

$$\begin{aligned} c_n^h[i] &:= \prod_{n' \in \text{Succ}(n)} c_{n'}^{p,h}[i], \\ c_n^o[i] &:= \sum_{n' \in \text{Succ}(n)} \Pr(n') c_{n'}^h[i], \\ c_n^p[i] &:= \prod_{n' \in \text{Succ}(n)} c_{n'}^o[i], \end{aligned}$$

where $\text{Succ}(n)$ is the set of successors of node n of the graph.

Step 3: estimation of the cost of a CoMDP state

Finally, the components of the cost vectors are combined in an admissible way to estimate the cost of a given state of the CoMDP. For this, the graph nodes that are relevant to the state are first identified. These are the proposition nodes or chance nodes of the graph (at the layer whose date is appropriate) representing the Boolean variables that are true in the state, and the effects of the chance nodes in the state's event queue [LIT 05]. The final estimate is the maximum component of the product of the cost vectors of the relevant node. This represent the value associated with the most difficult subgoal to reach:

$$h_{\text{Pr}}^{gp}(s) = \max_{i \in |\Phi|} \prod_{n \in \text{relevant}(s)} c_n[i] \leq C_{\text{Pr}}^*(s).$$

Finally, let us note that the planning graph is not limited to estimating reachability and success probability, but that it is often used to estimate the duration or the cost of a plan [BRY 06].

15.3. Reinforcement learning: FPG

15.3.1. Employing approximate methods

All the algorithms presented so far to solve operation planning problems require estimating the utility of state-action pairs. As in most approaches to solve MDPs, this enables determining the best action to perform in each state, or at least in each state visited by the optimal solution found.

But the number of states involved is often very large, what makes such algorithms very memory consuming, even if good heuristics are employed to limit their exploration. Some overcome the exponential growth of the memory

requirements through approximations, as in the case of hybrid algorithms discussed in section 15.2.2.4. Yet one approach that has only been proposed recently is to use function approximators either to approximate a value function (Chapter 3) or to define a policy (Chapter 5).

The FPG algorithm (*Factored Policy-Gradient*) [ABE 06] indeed proposes to employ one of the stochastic gradient methods seen in Chapter 5 to solve operation planning problems. The principle is to let one of these reinforcement learning algorithms interact with a simulator of the planning problem (assuming such a simulator is available). As a reminder, in this setting a policy is viewed as a parameterized function, learning being equivalent to optimizing the parameters. To present FPG in more details, we will mainly discuss the two major points in its design:

- 1) the choice of the form of the parameterized policy,
- 2) the choice of the most appropriate optimization algorithm.

15.3.2. Parameterized policy

To choose the form of the parameterized policy, let us look at the constraints posed by the required inputs and outputs. The objective is to find a good compromise so that the form depends on a small number of parameters but can still represent efficient policies.

15.3.2.1. Inputs

As an operation planning problem has some structure, it is natural to take as inputs for the function approximator not a simple number identifying the current state, but a vector that depends on this current state. In our framework, a complete information is given by 1) the Boolean variables of the model and 2) the queue of upcoming events. Because of its variable length, an event queue does not lend itself well to a vector-based representation, short of keeping a reduced number of elements (the closest in time for example). In the FPG planner, the choice is to restrict the input vector \mathbf{o} to the Boolean variables.

15.3.2.2. Outputs

At each decision point, it has to be determined for each eligible operation whether it should be executed or not. But ideally, a parameterized policy returns a probability distribution over possible actions, an action corresponding to triggering a set of eligible operations. This poses a problem since:

- the number of possible actions grows exponentially with the number of eligible actions;
- a probability distribution over a variable number of actions seems difficult to represent with a function approximator.

To solve this problem, FPG employs a controller *factored* as one sub-controller per operation. For the current \mathbf{o} input, FPG calculates for each eligible operation op a probability of being executed $Pr[op \mid \mathbf{o}; \theta_{op}]$, then samples a subset of these operations. But mutex operations should not be triggered together. To that end, the solution employed by FPG simply consists of identifying conflicting operations and randomly removing them until all conflicts are solved.

15.3.2.3. Function approximator

Having defined inputs and outputs, various function approximators can still be employed. In practice, implementations of FPG so far have used decision trees and, principally, artificial neural networks such as perceptrons. The best results have been obtained with perceptrons with no hidden layer, also called linear networks (see equation (5.4)).

Figure 15.6 shows such a controller based on linear networks. In this particular case, a state comprises not only an event queue and Boolean variables (called “predicates”), but also the current time and resources. The controller is here interacting with a simulator via the $simulateTillHappening(s_t, \mathbf{a}_t)$ function which samples the next state s_{t+1} depending on the current state s_t and the chosen action vector \mathbf{a}_t .

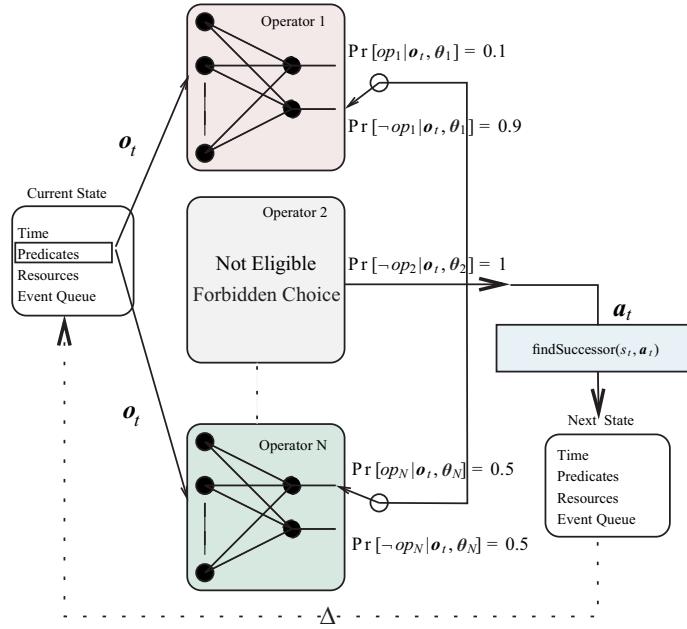


Figure 15.6. High level overview of the FPG controller

We can observe that these gradient methods can, if needed, handle continuous inputs and outputs. We can therefore, in theory, address more complex problems than the ones presented here. A typical example is project planning when task durations are uncertain.

15.3.3. Gradient methods

15.3.3.1. Terminating an execution

Such operation planning problems end when a terminal state is reached, may it be a success or a failure. But it is also possible that some executions of a policy never end. In particular, it is possible to end up in a part of the state space with no way out, whatever the policy. To prevent a reinforcement learning algorithm from remaining stuck in such a way, we have to define a maximum duration T_{\max} where any state is considered a failure. This duration can be measured in wall clock time (seconds, hours, days, etc.) or in number of decision points.

15.3.3.2. Choice of OLPomdp

Because we guarantee the termination of any execution, we can adopt a gradient method for regenerative processes (see section 5.2.4.1). However the current implementations of FPG use the OLPomdp algorithm (Algorithm 5.3) to benefit from the efficiency of its online learning. OLPomdp is also preferred over other more computationally intensive algorithms, such as “actor-critic” methods seen in section 5.3, because we benefit here from very cheap samples as far as a fast simulator exists.

15.3.3.3. Optimized criterion

As described here, OLPomdp optimizes the average reward per simulation step. Yet, in the case of operation planning, the criterion that should be optimized is the average reward per complete execution (per complete path from an initial state to a terminal state). Instead of maximizing for example the success probability, OLPomdp optimizes the frequency at which success states are encountered, making a compromise between probability of success and short executions. To fix this phenomenon, we can modify OLPomdp as follows:

- delete the eligibility trace each time one restarts from the initial state;
- add fictitious decision steps so that all executions have the same length;
- accumulate rewards during an execution to consume them only when a terminal state is reached.

In practice, OLPomdp has mainly been used without these corrections. A reason for that is that learning is harder when executions are long. We prefer not adding fictitious decision steps if this enables learning policies favoring short executions.

15.3.4. Improving FPG

As it is based on a gradient method, FPG can benefit from some improvements.

First, FPG suffers from the fact that its initial exploration is random, as a Brownian motion. Let us take the famous blocksworld [SLA 01] where numbered blocks must be stacked according to a given configuration. We observe that, if a random policy is applied, the goal configuration is encountered with a frequency that grows exponentially with the number of blocks considered. It would therefore be useful to direct FPG's searches. Two possible approaches are:

- using a progress estimator: this consists of giving, after each decision, a reward indicating whether we seem to be getting closer to or further away from the goal; a difficulty is to estimate the distance to the goal; but a simple estimator has already proved to be very efficient [BUF 09];
- following a heuristic's decisions: instead of starting with a random policy, we can envision benefiting from decision rules known to be efficient, such as the ones developed in the field of classical (deterministic) planning.

Moreover, the computations performed by `OLpomdp` usually remain rather expensive. Now, in many operation planning problems, the received reward is null most of the time. It is then possible 1) to modify the θ vector only in case of a non-zero reward and 2) to decrease an operation's eligibility trace only when a reward is received or when this operation is used.

15.4. Experiments

This experiments, taken from [ABE 07a], compare MOP, Prottle and FPG. We present results according to three criteria: the probability of reaching a goal, the average execution time (the result being a success or a failure) and the average long-term reward (for FPG). The problems considered are:

- *Probabilistic Machine Shop (MS)* [MAU 05],
- *Maze (MZ)*,
- *Teleport (TP)* (a science-fiction scenario in which slow teleportation is safer than fast teleportation) [LIT 05],
- *PitStop* (a car race with pit stops management and uncertain action durations) [ABE 07a].

For the first three problems, we use the versions given in [LIT 05]. The experiments use: FPG with linear networks, MOP, Prottle, a random policy triggering actions at random and a naive policy that tries to execute *all* eligible actions. These last two algorithms enable verifying that optimizing is necessary to obtain good results.

<i>Prob.</i>	<i>Opt.</i>	<i>% fail</i>	<i>PL</i>	<i>R</i>	<i>Time</i>
MS	FPG	1.33 (0.02)	6.6 (5.5)	118 (166)	532 (600)
MS	FPG	0.02	5.5	166	600
MS	Prottle	2.9			272
MS	MOP		Out of memory		
MS	random	99.3	18	0.1	
MS	naive	100	20	0.0	
MZ	FPG	19.1 (14.7)	5.5 (6.9)	134 (130)	371 (440)
MZ	FPG	14.7	6.9	130	440
MZ	Prottle	17.8			10
MZ	MOP	7.92 (7.15)	8.0 (8.2)		71 (72)
MZ	MOP	7.15	8.2		72
MZ	random	76.5	13	16.4	
MZ	naive	90.8	16	8.6	
TP	FPG	34.4 (33.3)	18 (18)	298 (305)	340 (600)
TP	FPG	33.3	18	305	600
TP	Prottle	79.8			442
TP	MOP		Out of memory		
TP	random	99.6	15	1.0	
TP	naive	100	19	0.0	
PitStop	FPG	0.0	20180	142	41
PitStop	random	29.0	12649	41.0	
PitStop	naive	100	66776	0.0	

Table 15.1. Results over 3 trial domains. Experiments for MOP and FPG have been repeated 100 times. The Opt. column gives the optimization engine used. % fail=percentage of failed executions, PL=plan length, R is the average long-term reward, and Time is the optimization time in seconds

All experiments have a maximum duration of 600 seconds. Other parameters are described in Table 15.2. In particular, the constant gradient step α has been chosen as the greatest value guaranteeing the convergence over 100 executions on all domains. The experiments have been conducted on a Pentium IV 2.4 GHz with 1 GB of memory. The results are summed up in Table 15.1. Except for Prottle, failure probabilities and average execution lengths have been estimated based on 100,000 simulated executions of the optimized plan. Prottle's results come from [LIT 05], citing the smallest failure probability results. Experiments with FPG and MOP have been repeated 100 times to take into account the stochastic nature of the optimization process. The repeated experiments with FPG are important to measure the impact of local minima. For FPG and MOP are presented the average results over the 100 optimizations and, in

Parameter	Value	Opt.
θ_{init}	0	FPG
α	1×10^{-5}	FPG
β	0.95	FPG
ϵ	1	MOP
ϵ	0.0 to 0.6	Prottle

Table 15.2. Parameter settings not discussed in the text

parentheses, the best optimization over the 100 (according to the failure probability criterion). The small differences between average and best results indicate that local optima have not been severe.

In general, Table 15.1 shows that FPG at least compares with Prottle and MOP, and is better on the hardest problem: *Machine Shop*. The bad performances of Prottle in the *Teleport* problem – 79.8 % of failures compared to FPG’s 34.4 % – come from the fact that, here, it considers execution lengths of at most 20 time units.

Table 15.1 shows that Prottle obtains good results faster on *Maze* and *Machine Shop*. The seemingly slower optimization with FPG or MOP is due to the asymptotic convergence. For FPG, the criterion is being optimized until the average long-term reward does not improve in 5 consecutive estimates (of 10,000 steps each). In practice, good policies are often found long before convergence to this criterion [ABE 07a]. Experimental results for the continuous time problem *PitStop* show FPG’s optimization capability in a framework where random variables are discrete or continuous.

15.5. Conclusion and outlook

This chapter has presented a particular application of Markov decision processes: operation planning. In these problems, state and action spaces (an action being a set of operations) are very structured. We therefore try to exploit this structure to overcome the combinatorial explosion of the spaces. The algorithms presented usually make use of (and therefore calculate) the value function, one of them (FPG) preferring an optimization based on a gradient method. These algorithms are made more efficient by exploiting classical approaches such as:

- using a heuristic known to often indicate a good direction to follow;
- solving a simplified problem first, before solving the original problem (we speak of constraint relaxation);
- or restricting the space of explored solutions (at the risk of losing optimality).

All the work presented here is recent as these are among the first approaches to address the difficult problem of operation planning with both probabilistic and temporal aspects. But they have been preceded by a lot of research on simplest cases of probabilistic planning, cases where operations cannot be concurrent and time does not appear. A good place to look for references on this topic is the international planning competition (IPC), during which a “probabilistic planning” track is organized since 2004.

In addition to the MDP approach, the research in automated planning studies numerous other models and algorithms. They range from representation based on transition systems and heuristic search algorithms, to logical or constraint-based representations and algorithms for satisfiability, model-checking, theorem proving and constraint propagation, through graph-based representations and algorithms [GHA 04].

The compression of the state space to explore remains an important research direction. In this framework, the planning graph mentioned in section 15.2.3.3 is not only a source of heuristic generations, but also a compact space which can be explored (traditionally with a backward search) to generate a concurrent plan [GHA 04, pages 125–129]. Graphplan [BLU 97] was the first to exploit this idea that is now present in many planners. Paragraph [LIT 06] is an extension of Graphplan to probabilistic systems equivalents to CoMDP. Paragraph creates an optimal contingent plan by concatenating sub-trajectories generated by Graphplan. Paragraph is a concurrent probabilistic planner, but not temporal. Research is currently conducted to extend it to the temporal case, by compiling temporal operations into instantaneous operations that Paragraph can already process and by handling time constraints between its last operations via a linear programming algorithm. This way of combining planning graph and linear programming is inspired by the deterministic temporal planner LPGP [LON 03] and constitutes a very promising direction for temporal probabilistic planning.

Taking into account all possible contingencies is very costly. Another research direction is to *incrementally* generate contingent plans [DEA 03]. Here, the goal is not to generate an optimal plan, but a plan covering the most useful contingencies, depending on the available computation time. Identifying these contingencies, that is the estimation of their utility, relies on a backward propagation of the utility functions in the planning graph. Recent work studies the generation of mixed plans (classical, conformant, contingent) which allow for an efficient management of uncertainty depending on the needs [FOS 07].

Finally, an important research direction concerns more expressive formalisms, allowing the modeling of operation planning problems even more generic than the ones we considered in this chapter. Generalized semi-Markov process (GSMP) [MAT 62] is a powerful formalism to describe discrete event systems made of

asynchronous processes operating in continuous or discrete time and with uncertainty. As its name suggests, a generalized semi-Markov decision process (GSMDP) [YOU 03] is a generalization of semi-Markov decision process (SMDP) based on GSMPs. This generalization allows for modeling durations whose probability distributions (which cannot rely on a memory, as in SMDPs) depends not only on the current state, but also on the trajectory followed by the system. A GSMDP can be seen as the asynchronous composition of concurrent SMDPs. GSMDPs can model planning problems taking into account continuous time, exogenous events (events out of the control of the planner), and generalized distributions, while allowing for concurrent events and actions. Techniques for solving these problems include generate-test-repair methods based on an evaluation of the current policy by probabilistic sampling [YOU 03], simulation methods like policy iteration with approximation [RAC 08] or the approximation of the GSMDP with an MDP by approximating generalized distributions with phase-type laws [YOU 04b]. Although these techniques do not guarantee optimality, the GSMDP formalism is a promising research avenue for probabilistic temporal planning.

15.6. Bibliography

- [ABE 04] ABERDEEN D., THIÉBAUX S. and ZHANG L., “Decision-theoretic military operations planning”, *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS’04)*, Whistler, Canada, pp. 402–412, 2004.
- [ABE 06] ABERDEEN D., “Policy-gradient methods for planning”, *Advances in Neural Information Processing Systems 18 (NIPS’05)*, MIT Press, Cambridge, MA, pp. 9–16, 2006.
- [ABE 07a] ABERDEEN D. and BUFFET O., “Temporal probabilistic planning with policy-gradients”, *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS’07)*, Providence, RI, 2007.
- [ABE 07b] ABERDEEN D., BUFFET O. and THOMAS O., “Policy-gradients for PSRs and POMDPs”, *Proceedings of the 11th International Conference on Artificial Intelligence and Statistics (AISTATS’07)*, San Juan, Puerto Rico, 2007.
- [BAR 95] BARTO A. G., BRADTKE S. and SINGH S., “Learning to act using real-time dynamic programming”, *Artificial Intelligence*, vol. 72, pp. 81–138, 1995.
- [BLU 97] BLUM A. and FURST M., “Fast planning through planning graph analysis”, *Artificial Intelligence*, vol. 90, pp. 281–300, 1997.
- [BON 03] BONET B. and GEFFNER H., “Labeled RTDP: “Labeled RTDP: Improving the convergence of real-time dynamic programming”, *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS’03)*, Trento, Italy, pp. 12–21, 2003.
- [BRE 02] BRESINA J., DEARDEN R., MEULEAU N., RAMAKRISHNAN S., SMITH D. and WASHINGTON R., “Planning under continuous time and resource uncertainty: a challenge for AI”, *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence (UAI’02)*, Morgan Kaufmann, San Francisco, CA, pp. 77–84, 2002.

- [BRY 06] BRYCE D. and KAMBHAMPATI S., “A tutorial on planning graph based reachability heuristics”, *AI Magazine*, vol. 27, no. 4, 2006.
- [BUF 09] BUFFET O. and ABERDEEN D., “The factored policy-gradient planner”, *Artificial Intelligence*, vol. 173, no. 5-6, pp. 722–747, 2009.
- [CUS 07] CUSHING W., KAMBHAMPATI S., MAUSAM M. and WELD D. S., “When is temporal planning really temporal?”, *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, Hyderabad, India, pp. 1852–1859, 2007.
- [DEA 03] DEARDEN R., MEULEAU N., RAMAKRISHNAN S., SMITH D. and WASHINGTON R., “Incremental contingency planning”, *Proceedings of the ICAPS-03 Workshop on Planning Under Uncertainty*, Trento, Italy, 2003.
- [FOS 07] FOSS J., ONDER N. and SMITH D., “Preventing unrecoverable failures through precautionary planning”, *Proceedings of the ICAPS'07 Workshop on Moving Planning and Scheduling Systems into the Real World*, 2007.
- [FOX 03] FOX M. and LONG D., “PDDL2.1: an extension to PDDL for expressing temporal planning domains”, *Journal of Artificial Intelligence Research*, vol. 20, pp. 61–124, 2003.
- [GEI 01] GEIBEL P., “Reinforcement learning with bounded risk”, *Proceedings of the 18th International Conference on Machine Learning (ICML'01)*, Morgan Kaufmann, San Francisco, CA, pp. 162–169, 2001.
- [GHA 04] GHALLAB M., NAU D. and TRAVERSO P., *Automated Planning: Theory and Practice*, Morgan Kauffman, San Mateo, CA, 2004.
- [HAN 01] HANSEN E. A. and ZILBERSTEIN S., “LAO*: a heuristic search algorithm that finds solutions with loops”, *Artificial Intelligence*, vol. 129, pp. 35–62, 2001.
- [LIT 05] LITTLE I., ABERDEEN D. and THIÉBAUX S., “Prottle: a probabilistic temporal planner”, *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI'05)*, Pittsburgh, PA, pp. 1181–1186, 2005.
- [LIT 06] LITTLE I. and THIÉBAUX S., “Concurrent probabilistic planning in the graphplan framework”, *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS'06)*, pp. 263–273, 2006.
- [LON 03] LONG D. and FOX M., “Exploiting a graphplan framework in temporal planning”, *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS'03)*, Trento, Italy, pp. 51–62, 2003.
- [MAT 62] MATTHES K., “Zur Theorie der Bedienungsprozesse”, *Transactions of the 3rd Prague Conference on Information Theory, Statistical Decision Functions, Random Processes*, Publishing House of the Czechoslovak Academy of Sciences, Liblice, Czechoslovakia, 1962.
- [MAU 04] MAUSAM M. and WELD D. S., “Solving concurrent Markov decision processes”, *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI'04)*, San Jose, CA, pp. 716–722, 2004.
- [MAU 05] MAUSAM M. and WELD D. S., “Concurrent probabilistic temporal planning”, *Proceedings of the 15th International Conference on Planning and Scheduling (ICAPS'05)*, 2005.

- [MAU 06] MAUSAM M. and WELD D. S., “Probabilistic temporal planning with uncertain durations”, *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI'06)*, Boston, MA, pp. 880–887, 2006.
- [MAU 07] MAUSAM M. and WELD D. S., “Planning with durative actions in stochastic domains”, *Journal of Artificial Intelligence Research*, vol. 31, pp. 33–82, 2007.
- [MCM 05] MCMAHAN H. B., LIKHATCHEV M. and GORDON G., “Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees”, *Proceedings of the 22nd International Conference on Machine Learning (ICML'05)*, ACM Press, New York, pp. 569–576, 2005.
- [RAC 08] RACHELSON E., FABIANI P., GARCIA F. and QUESNEL G., “Une approche fondée sur la simulation pour la résolution des processus décisionnels semi-markoviens généralisés”, *Actes de la Conférence d'Apprentissage (Cap'08)*, 2008.
- [REG 04] REGNIER P., *Algorithmique de la planification en IA*, Cépaduès, Toulouse, 2004.
- [SCH 05] SCHAEFFER S., CLEMENT B. and CHIEN S., “Probabilistic reasoning for plan robustness”, *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI'05)*, Edinburgh, Scotland, pp. 1266–1271, 2005.
- [SLA 01] SLANEY J. and THIÉBAUX S., “Blocks world revisited”, *Artificial Intelligence*, vol. 125, pp. 119–153, 2001.
- [SMI 06] SMITH T. and SIMMONS R., “Focused real-time dynamic programming for MDPs: squeezing more out of a heuristic”, *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI'06)*, Boston, MA, 2006.
- [YOU 03] YOUNES H. L. S. and SIMMONS R. G., “A framework for planning in continuous-time stochastic domains”, *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS'03)*, Trento, Italy, pp. 195–204, 2003.
- [YOU 04a] YOUNES H. L. S. and LITTMAN M. L., PPDDL1.0: an extension to PDDL for expressing planning domains with probabilistic effects, Report no. CMU-CS-04-167, Carnegie Mellon University, 2004.
- [YOU 04b] YOUNES H. L. S. and SIMMONS R. G., “Solving generalized semi-Markov decision processes using continuous phase-type distributions”, *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI'04)*, San Jose, CA, pp. 742–747, 2004.

Index

Symbols

- AO^* , 160
- LAO^* , 160
- R_{\max} , 62
- $SARSA(\lambda)$, 55
- CoMDP, 433
- ADD, *see* algebraic decision diagrams
- DEC-MDP, 280
- NOMDP, 285
- POMDP, 187, 188
 - transient, 193
- DEC-POMDP, 280
- BDD, *see* binary decision diagrams

A

- abalone, 382
- action, 3
 - continuous, 141
- actor-critic, 143
 - natural (NAC), 147
- adaptive dynamics learning (ADL), 265
- adaptive play, 252
- admissible heuristic, 159
- agent, 3
 - playing, 230
- algebraic backward induction, 349
- algorithm
 - envelope, 158
 - offline, 155
 - online, 155
 - Shapley's algorithm, 249
 - simulation-based, 163

approximation

- capacity, 68
- operator, 71
- power, 68
- theory, 73

B

- Bayesian networks, 103
 - dynamic, 103
- Bellman
 - operator, 69
 - optimality principle, 9
- Bellman equation, 18
 - algebraic, 347, 348
 - for POMDPs, 195
 - possibilistic, 335
- bias optimality, 28
- bias-variance trade-off, 92, 139, 170
- biodiversity, 375

C

- co-evolution, 305
- communication, 292
- controlled stochastic process, 4
- covering number, 89
- criterion
 - γ -discounted, 8
 - adapted average, 201
 - average, 8
 - finite, 8
 - total reward, 8

D

decision diagrams
 algebraic, 113
 binary, 113
 decision trees, 108
 depth-width trade-off, 170
 deviation matrix, 26
 discount factor, 10
 dyna, 60
 dyna-Q, 61
 focused, 61
 dynamic programming, 17, 202
 approximate, 67
 asynchronous, 30, 159
 exact methods, 67
 for DEC-POMDPs, 296
 possibilistic, 333

E

E^3 , 61
 ED-Dec-MDP, 289
 efficient play, 264
 eligibility trace, 51
 empirical error, 72
 equilibrium, 236
 in dominating strategies, 237
 in mixed strategies, 239
 Nash equilibrium, 238, 247
 Nash equilibrium in mixed strategies, 239
 error
 generalization, 89
 learning, 89
 temporal difference, 46
 expected payoff, 233, 244
 expected utility
 generalized, 345
 qualitative binary possibilistic, 350
 qualitative optimistic, 331, 346, 350
 qualitative pessimistic, 331, 346, 350
 experience replay, 61
 exploration/exploitation trade-off, 42

F

factored policy gradient (FPG), 443
 feature, 71
 vector, 131

fictitious play, 250

finite difference method, 132
 focused reinforcement learning, 168, 173
 function approximation, 67

G

game, 230, 327
 Bayesian, 307
 chance, 231
 dynamic, 240
 finite, 232
 game theory, 229
 in extensive form, 240, 241
 in strategic form, 232
 iterated, 243
 Markov, 245
 matching pennies, 238
 normal form, 232
 prisoner's dilemma, 237, 243
 repeated, 241, 243
 sequential, 231
 simultaneous, 231, 232
 static, 232
 stochastic, 245, 246
 with complete information, 231
 with imperfect information, 231
 with incomplete information, 231
 with perfect information, 231, 241
 zero-sum, 231

Gauss-Seidel algorithm, 30

generative model, 85

Go, 180

gradient, 128
 instant, 139
 natural, 147
 policy, 128

gradient ascent

infinitesimal (IGA), 258
 generalized infinitesimal (GIGA), 266
 stochastic, 130

H

heuristic sampling, 62
 history, 191
 game history, 241
 horizon
 finite, 5, 202

indefinite, 5
 infinite, 5, 203
 reasoning, 154, 155

I

importance sampling, 149
 influence diagram, 5
 possibilistic, 342
 iterative pruning, 219

J

joint action learners (JAL), 255
 joint equilibrium based search for policies (JESP), 306
 dynamic programming (DP-JESP), 306

K

kernel methods, 73

L

learning
 model, 59
 supervised, 71
 least squares temporal differences (LSTD), 82
 least-squares methods, 82
 likelihood ratio, 133
 linear programming, 28

M

MAA*, 299
 Markov chain, 13
 ergodic, 137
 valued, 14
 Markov decision problem, 3, 8
 Markov decision process, 3
 algebraic, 342
 decentralized, 281
 factored, 99
 multicriteria, 346, 349
 partially observable, 188
 possibilistic, 329, 346, 350
 possibilistic multicriteria, 350
 possibilistic partially observable, 340
 qualitative, 346
 robust, 327
 subjective, 306
 Markov process, 13

valued, 14
 Markov reward process, 14
 matrix

 limiting, 24
 stochastic, 6
 micro-manipulation, 364
 micro-positioning, 364
 micro-robotics, 363
 minimax, 234
 Monte Carlo, 45, 174
 multi-armed bandit, 165, 171
 mutual exclusion (mutex), 431

N

neural networks, 73
 norm
 L^p -norm, 88
 seminorm span, 35

O

observability, 278
 joint, 278
 local, 278
 partial, 188
 observation, 188
 OLpomdp, 139
 operation, 429
 operations planning, 429

P

Pareto dominance, 240
 algebraic, 347
 Pareto optimality, 240
 PDDL, 427
 performance criterion, 8
 plan, 432
 conditional, 194
 planning, 3
 robust, 327
 planning graph, 440
 plausibility measure, 344
 decomposable, 344
 player, 230
 policy, 4, 193
 adapted, 196
 cyclic, 194
 finite state automata, 194
 greedy, 69, 159

- optimal, 9
- parameterized, 130
- tree, 194
- policy hill climbing (PHC), 260
 - PHC-Exploiter, 264
- policy iteration, 30
 - approximate, 77, 154
 - for POMDPs, 222
 - possibilistic, 338
- possibility, 330
 - distribution, 330
 - theory, 320
- predator-prey, 381
- predictive state representation (PSR), 225
- prioritized sweeping, 61
 - prioritized sweeping*
 - possibilistic, 339
- probably approximately correct, 91, 168
- process
 - Markov (see “Markov process”), 13
 - regenerative, 138
 - stationary, 6
- Prottle, 433, 437
- Q**
 - Q-learning, 49
 - $Q(\lambda)$, 56
 - adapted, 200
 - decentralized, 256
 - distributed, 257
 - dyna-Q, 61, 368
 - friend-or-foe-Q, 253
 - friend-Q, 252
 - hyper-Q, 264
 - hysteretic, 257
 - lenient, 257
 - minimax-Q, 253
 - Nash-Q, 253
 - OMQ, 255
 - QH-learning, 387
 - STM-Q, 368
 - team-Q, 252
 - quadratic norm, 72
- R**
 - R-learning, 58
 - RH-learning, 387
- real-time, 155
- real-time dynamic programming (RTDP), 159
- regression, 71
- regret, 265
 - no regret property, 266
- reinforcement learning, 39
 - focused, 168, 173
 - possibilistic, 339
- relaxation, 439
- reward, 3
- reward shaping, 150
- risk, 353
- robustness, 327
- rollout, 165
 - controlled, 168, 178
 - parallel, 167
- S**
 - saddle point, 236
 - scalability, 262
 - sea otter, 382
 - search
 - heuristic, 299
 - online, 154
 - tree, 154
 - security levels, 235
 - semiring, 343
 - canonic preorder, 343
 - idempotent, 343
 - sequential decision under uncertainty, 3
 - Shapley’s algorithm, 249
 - shortest path, 3
 - stochastic, 5, 158
 - simulation, 154
 - state, 3, 188
 - absorbing, 5
 - belief, 191
 - complete information, 191
 - goal, 5
 - information, 190
 - terminal, 5
 - stationary distribution, 25
 - statistics
 - exhaustive, 191
 - sufficient, 191
 - stochastic environment, 3

- strategy
 - best response, 238
 - grim trigger, 243
 - mixed, 233, 244
 - pure, 233
- support vector machines (SVM), 73
- supremum norm, 16
- T**
 - TD-gammon, 154
 - temporal difference, 46
 - trajectory model updates, 61
- U**
 - UCT, 180
- V**
 - value function, 9, 195
 - ϵ -optimal, 203
 - POMDP, 195
- adapted, 198
- piecewise linear, 203
- relative, 25
- representative vector, 206
- value iteration, 29, 70
 - approximate, 70
 - for POMDPs, 207
 - possibilistic, 334
 - robust, 328
- VC dimension, 89
- vector
 - dominated, 207, 213
 - neighborhood, 216
 - useful, 207
 - value function, 206
- W**
 - win or learn fast (WoLF), 260
 - Witness, 216, 218